⌛ 20 min

CSRF

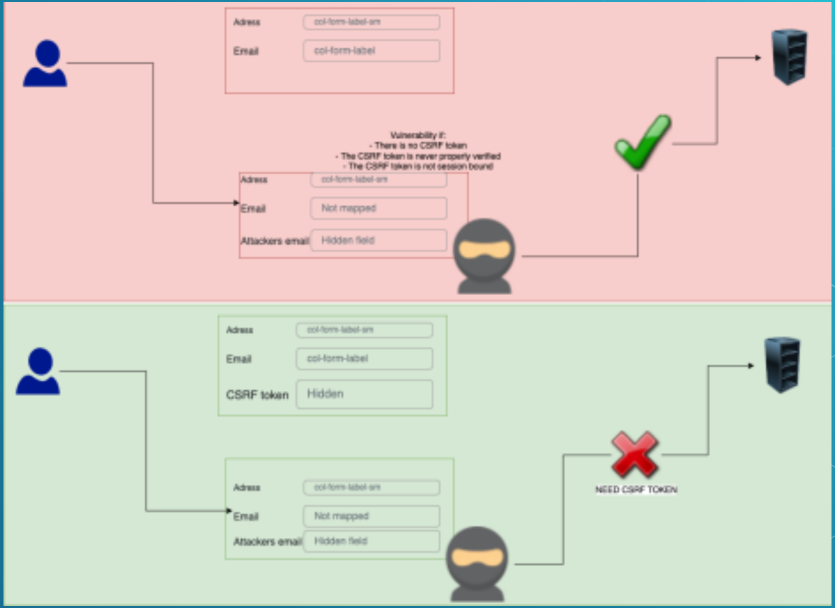
Build It Yourself

DAY 04

AGENDA

Introduction

In this practical guide, you will be building a banking application. You are creating the money transfer form and it has a recipient and amount field. This is very oversimplified and in most cases, you will notice banks add extra security measures such as requiring the user to re-enter their password when making a transaction and the use of MFA
(Multi-factor authentication).
In our example, an attacker will be able to emulate the form on his own website first before we try to put a stop to it.



Make a connection

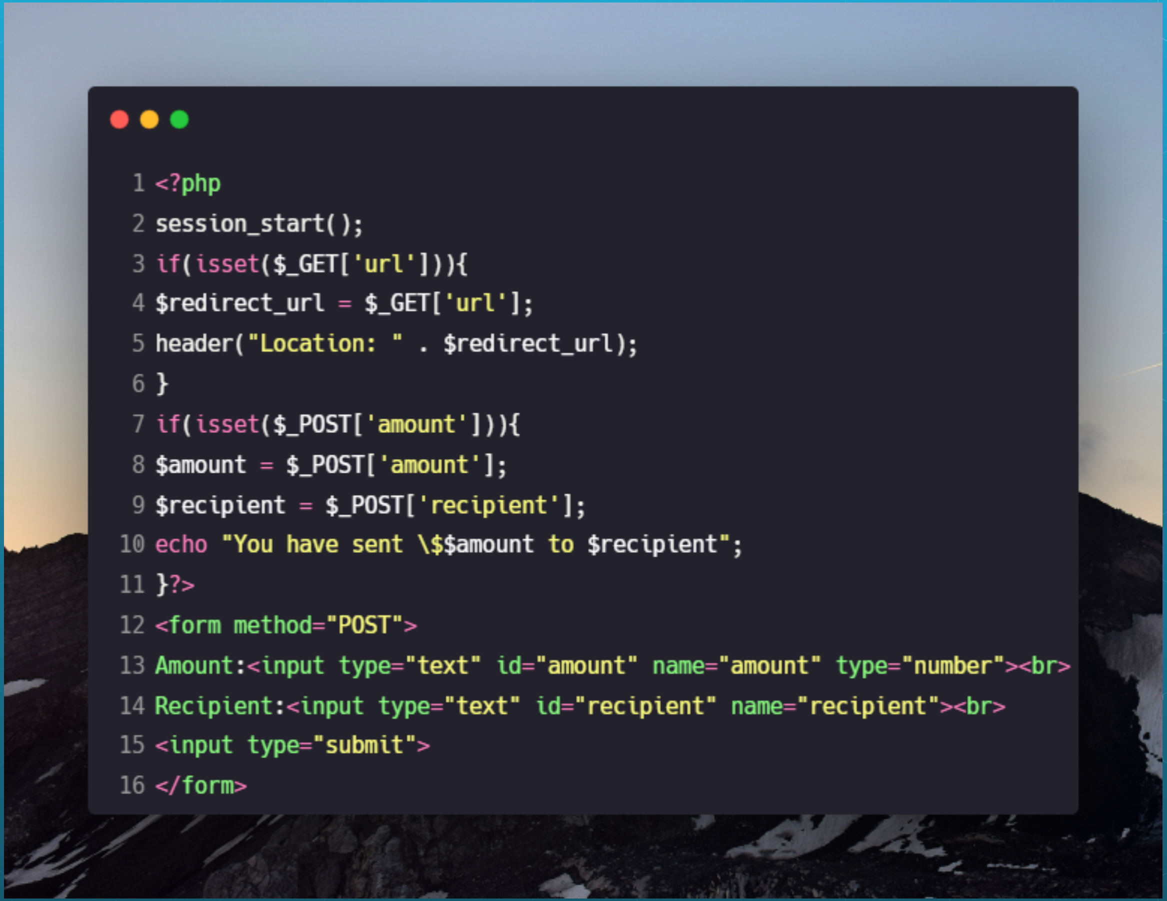
🎯 FTP connection: hackxpert.com

- 🧊 User: Training
- 🧊 Password: test

🎯 Create a new file on the server

📌 Use “nickname.php” for example “rat.php” where the nickname can be anything, as long as you can copy and paste it

⚠️🚨 THE SERVER GETS ERASED EVERY 24 HOURS



```php
<?php
session_start();
if(isset($_GET['url'])){
$redirect_url = $_GET['url'];
header("Location: " . $redirect_url);
}
if(isset($_POST['amount'])){
$amount = $_POST['amount'];
$recipient = $_POST['recipient'];
echo "You have sent \$$amount to $recipient";
}?>
<form method="POST">
Amount:<input type="text" id="amount" name="amount" type="number"><br>
Recipient:<input type="text" id="recipient" name="recipient"><br>
<input type="submit">
</form>
```

🧊 Let's create a CSRF

📌 **Enter the following code in your file and upload it to the server.**

Let’s hack it

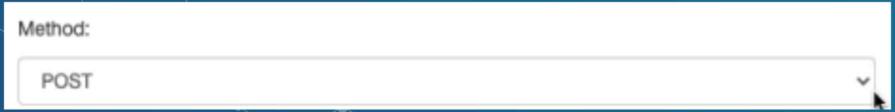
Now we are going to surf to our page at hackxpert.com/Training/YOURFILE.php and try to create a CSRF PoC. There are several tools such as burp suite pro's CSRF PoC creator but for the free option I always prefer:

🎯 https://security.love/CSRF-PoC-Genorator/

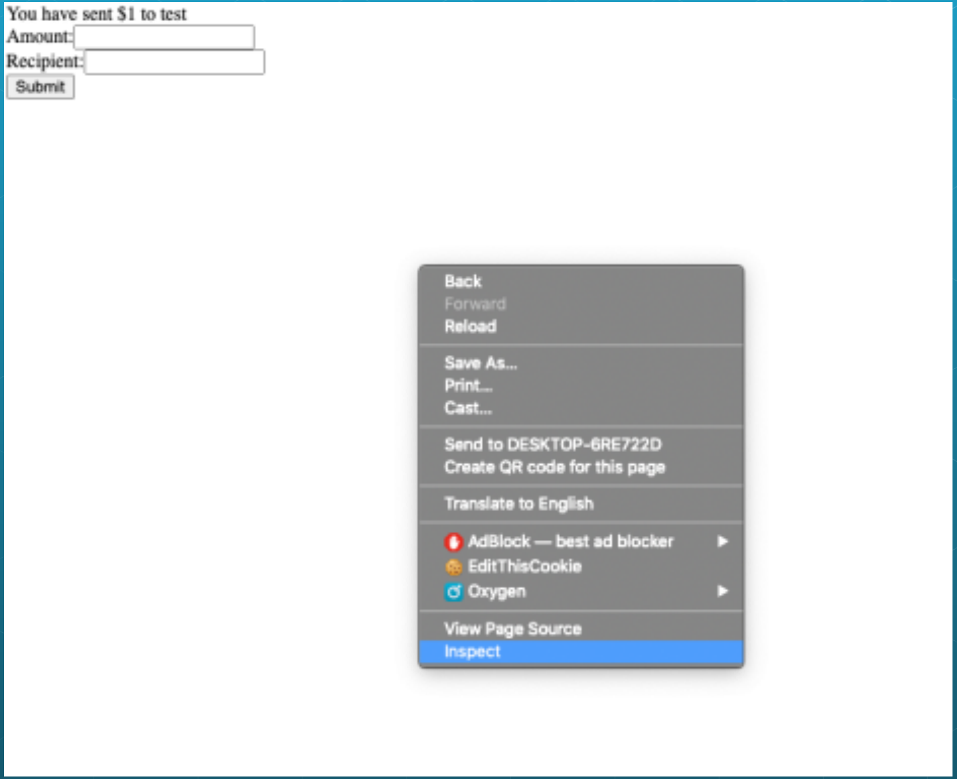
We know it's a POST request from the part:

<form action=YOUR_FILE.php method="POST">

so that should be the easy part

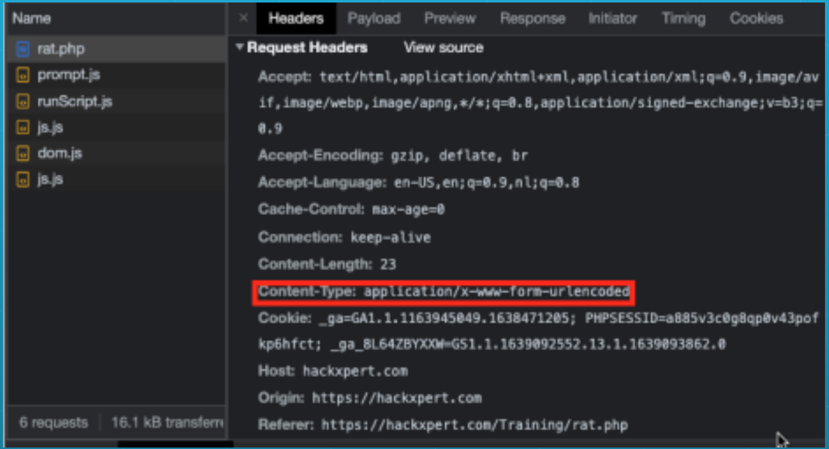

Method:
POST

🧊 The rest we can figure out in the developer console, so go to inspect the page:


You have sent $1 to test
Amount:
Recipient:
Submit

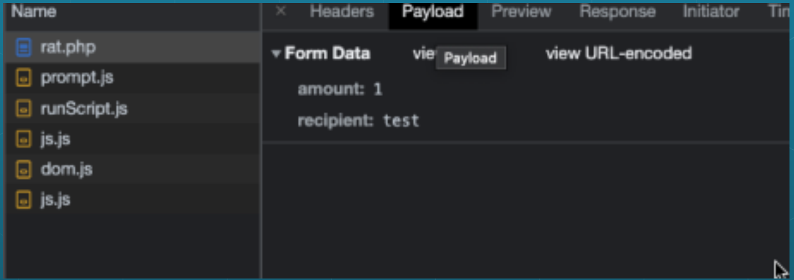
📌Go to the network tab and submit the form again but only after going to the network tab because otherwise you request will not be captured.

📌Here in the "request header" section, you can find the content type



```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/av
if,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=
0.9
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,nl;q=0.8
Cache-Control: max-age=0
Connection: keep-alive
Content-Length: 23
Content-Type: application/x-www-form-urlencoded
Cookie: _ga=GA1.1.1163945049.1638471205; PHPSESSID=a885v3c0g8qp0v43pof
kp6hfct; _ga_8L64ZBYXXW=GS1.1.1639092552.13.1.1639093862.0
Host: hackxpert.com
Origin: https://hackxpert.com
Referer: https://hackxpert.com/Training/rat.php
```

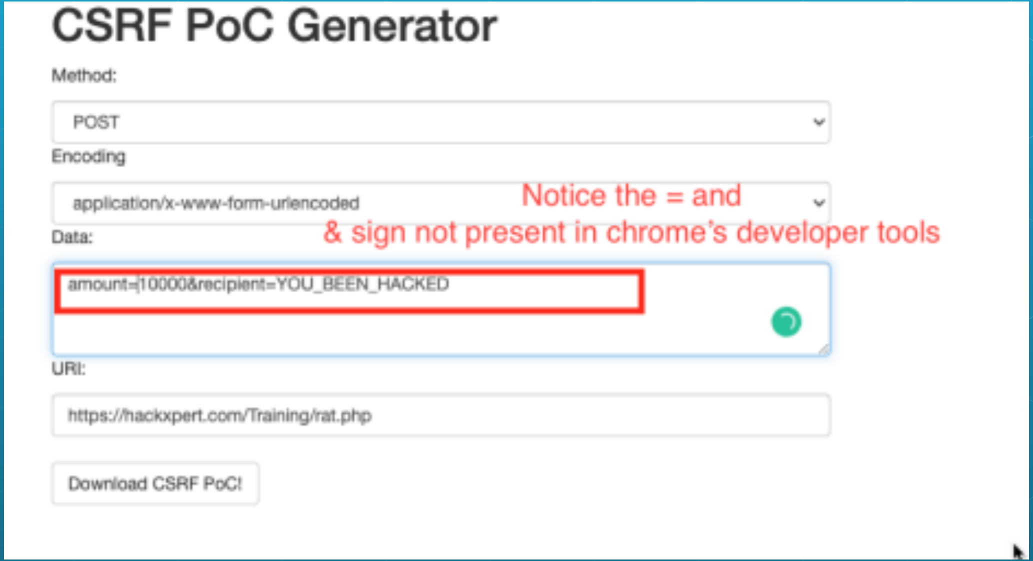
📌 And in the payload tab we can find the parameters we need.



```
amount: 1
recipient: test
```

📌 Let's fill these in but remember that in our tool, we need to enter parameter=value not parameter:value like chrome is displaying here.

📌The URI is simply the value of where the original page resides that the attacker wants to emulate



CSRF PoC Generator

Method:

POST

Encoding

application/x-www-form-urlencoded

Data:

```
amount=10000&recipient=YOU_BEEN_HACKED
```

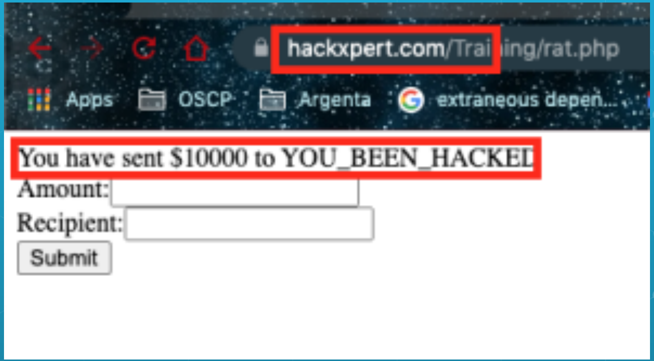
URI:

https://hackxpert.com/Training/rat.php

Download CSRF PoC!

📌Now download the PoC, open it from our PC or webserver, click the button and you should see on you

training file that a transfer was executed that came from an attacker.



hackxpert.com/Trai ing/rat.php

Apps · OSCP · Argenta · extraneous depen...

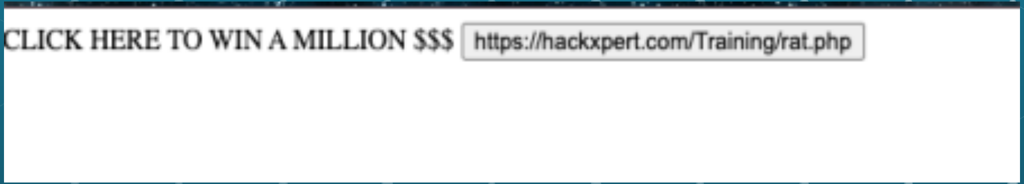
You have sent $10000 to YOU_BEEN_HACKED

Amount:

Recipient:

Submit

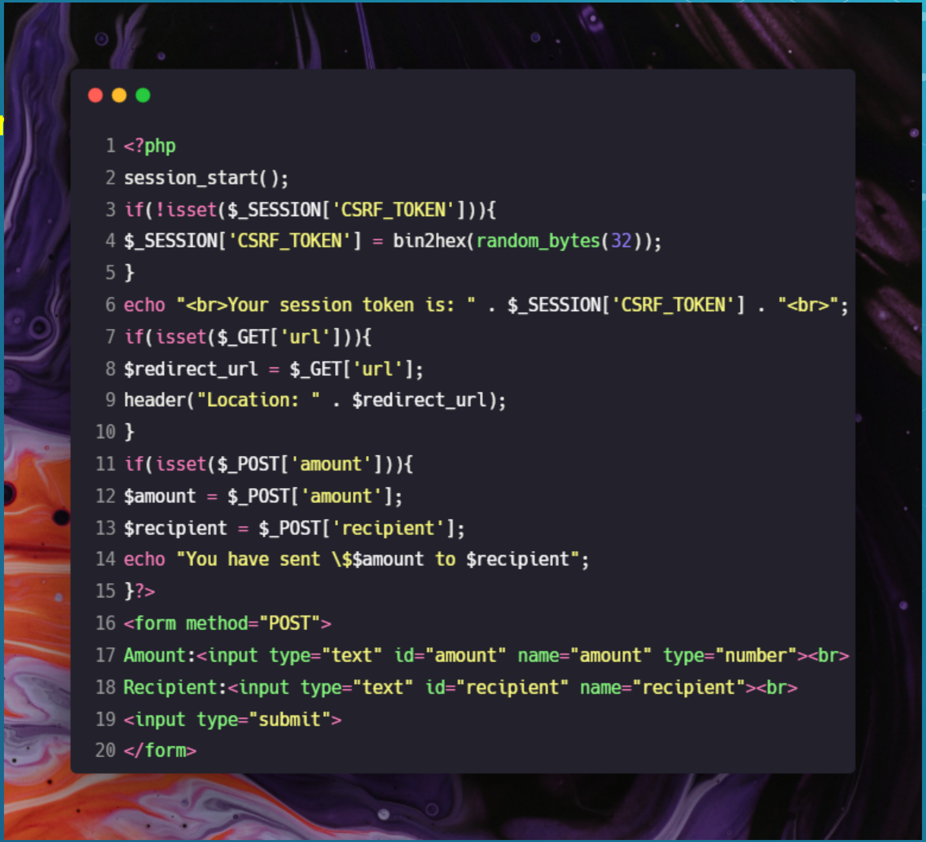
📌Now let's make it even spookier, since you have that HTML file for the PoC anyway, try to hide the input fields and simply display a text like "click here to win a million $$$"



CLICK HERE TO WIN A MILLION $$$ https://hackxpert.com/Training/rat.php

let’s secure it

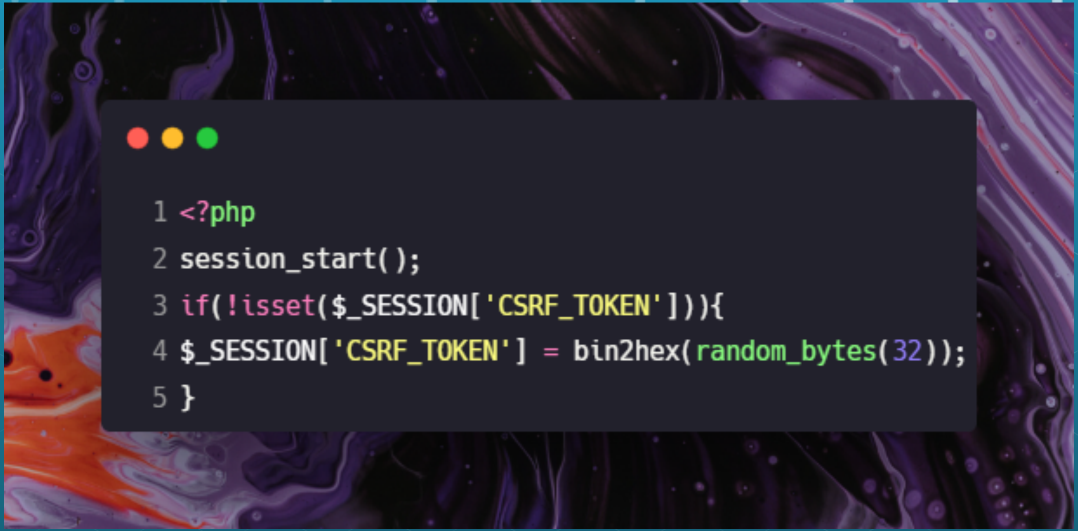
🎯 **Step 1: Generate the toker**



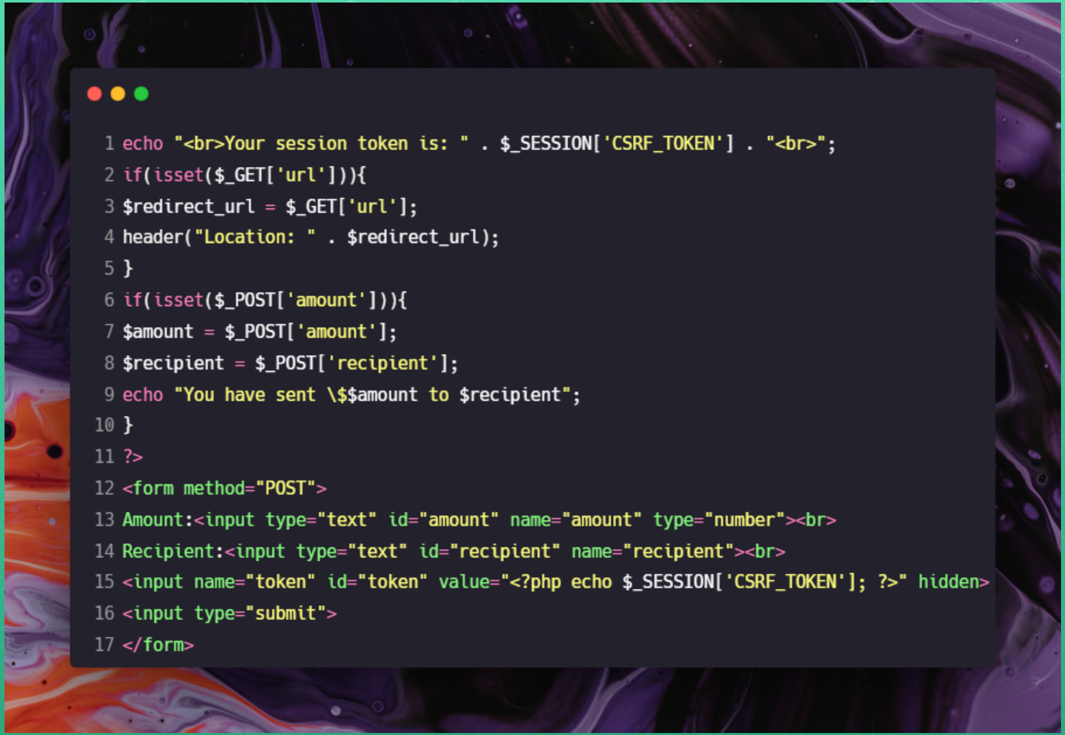
```
<?php
session_start();
if(!isset($_SESSION['CSRF_TOKEN'])){
$_SESSION['CSRF_TOKEN'] = bin2hex(random_bytes(32));
}
echo "<br>Your session token is: " . $_SESSION['CSRF_TOKEN'] . "<br>";
if(isset($_GET['url'])){
$redirect_url = $_GET['url'];
header("Location: " . $redirect_url);
}
if(isset($_POST['amount'])){
$amount = $_POST['amount'];
$recipient = $_POST['recipient'];
echo "You have sent \$$amount to $recipient";
}?>
<form method="POST">
Amount:<input type="text" id="amount" name="amount" type="number"><br>
Recipient:<input type="text" id="recipient" name="recipient"><br>
<input type="submit">
</form>
```

🎯Now add the token to the form

📌We usually add these tokens in a hidden field so let's do that here as well:



```php
<?php
session_start();
if(!isset($_SESSION['CSRF_TOKEN'])){
$_SESSION['CSRF_TOKEN'] = bin2hex(random_bytes(32));
}
```

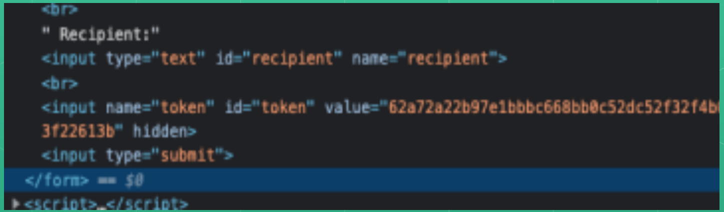


```
echo "<br>Your session token is: " . $_SESSION['CSRF_TOKEN'] . "<br>";
if(isset($_GET['url'])){
$redirect_url = $_GET['url'];
header("Location: " . $redirect_url);
}
if(isset($_POST['amount'])){
$amount = $_POST['amount'];
$recipient = $_POST['recipient'];
echo "You have sent \$$amount to $recipient";
}
?>
<form method="POST">
Amount:<input type="text" id="amount" name="amount" type="number"><br>
Recipient:<input type="text" id="recipient" name="recipient"><br>
<input name="token" id="token" value="<?php echo $_SESSION['CSRF_TOKEN']; ?>" hidden>
<input type="submit">
</form>
```

📌Here you can see a new field:

<input name="token" id="token" value="<?php echo $_SESSION['CSRF_TOKEN']; ?>" hidden>

Which will contain our token when the PHP page is rendered.



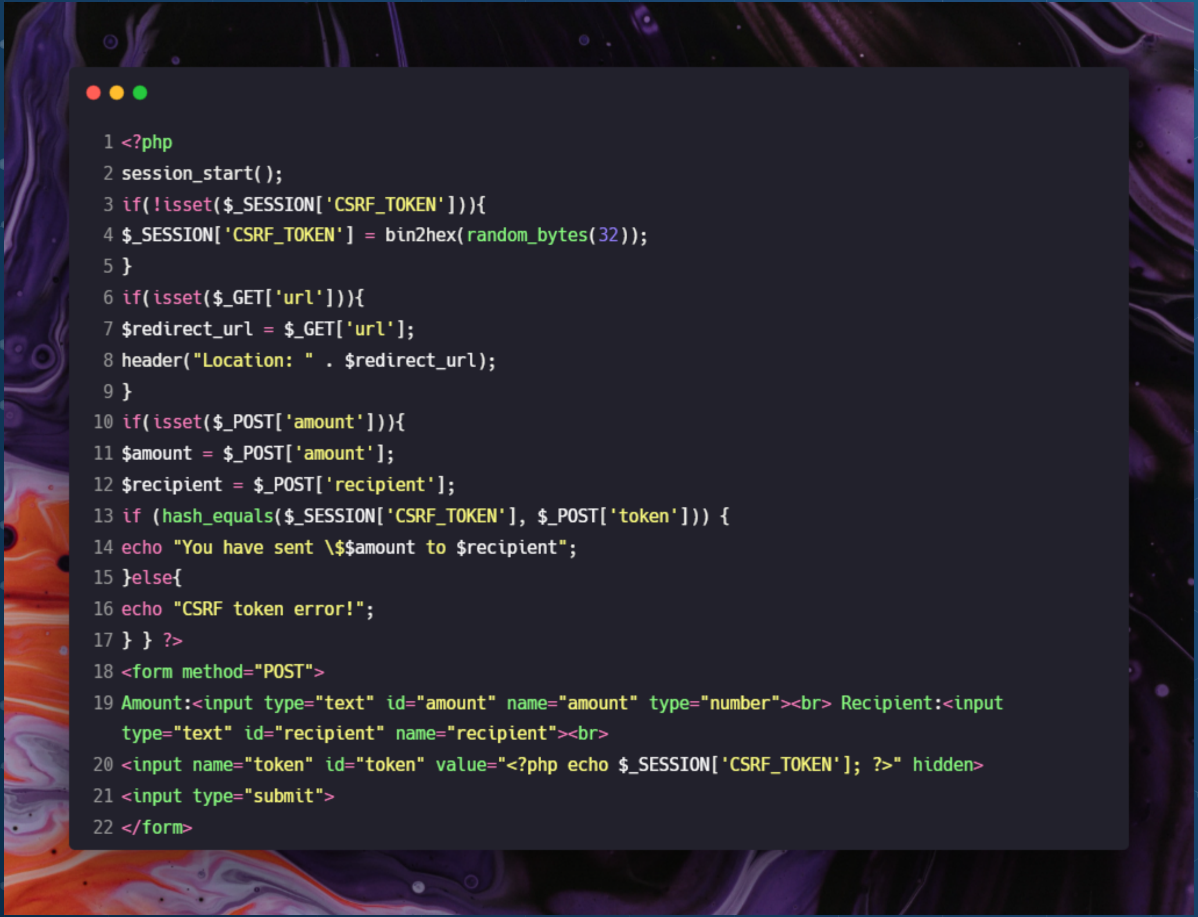
```
    <br>
    " Recipient:"
    <input type="text" id="recipient" name="recipient">
    <br>
    <input name="token" id="token" value="62a72a22b97e1bbbc668bb0c52dc52f32f4b
    3f22613b" hidden>
    <input type="submit">
  </form> == $0
 ▸<script>…</script>
```

🧑‍💻 Try to hack it

- ❑ So now that we have a CSRF token, it should be secure right? Try the same method of hacking the system as we tried before with the PoC generator. It works right? Why?

🧑‍💻 So what is the solution?

- ❑ Of course, you still need to check if the CSRF token is valid so change the code as follows:



```php
<?php
session_start();
if(!isset($_SESSION['CSRF_TOKEN'])){
$_SESSION['CSRF_TOKEN'] = bin2hex(random_bytes(32));
}
if(isset($_GET['url'])){
$redirect_url = $_GET['url'];
header("Location: " . $redirect_url);
}
if(isset($_POST['amount'])){
$amount = $_POST['amount'];
$recipient = $_POST['recipient'];
if (hash_equals($_SESSION['CSRF_TOKEN'], $_POST['token'])) {
echo "You have sent \$$amount to $recipient";
}else{
echo "CSRF token error!";
} } ?>
<form method="POST">
Amount:<input type="text" id="amount" name="amount" type="number"><br> Recipient:<input
type="text" id="recipient" name="recipient"><br>
<input name="token" id="token" value="<?php echo $_SESSION['CSRF_TOKEN']; ?>" hidden>
<input type="submit">
</form>
```

📌You can notice the new IF clause around the action of making a transaction. This is proper CSRF protection but things can still go wrong, this is why I urge you to look at the tips.

TIPS

- ❑ CSRF can go wrong if there is no token where one is needed, this is the most
- ❑ overlooked issue because testing existing things is easy but realizing something is
- ❑ not there when it should be is hard. Use automated tooling to check all the forms
- ❑ with creating, update or delete actions except for registration forms and things
- ❑ where a user is not logged in yet.
- ❑ Use a hash comperating function because a normal comperator (such as ==) might
- ❑ open you up to type mismatching attacks
- ❑ Always use 1 central CSRF generator and validator you include in all pages
- ❑ Check the full parameter and not just part of it

Resources

- ❑ https://security.love/CSRF-PoC-Genorator/
- ❑ https://portswigger.net/burp/documentation/desktop/functions/generate-csrf-poc
- ❑ https://owasp.org/www-community/attacks/csrf
- ❑ https://cheatsheetseries.owasp.org/cheatsheets/Cross-
- ❑ Site_Request_Forgery_Prevention_Cheat_Sheet.html