**Introduction to Information Security**

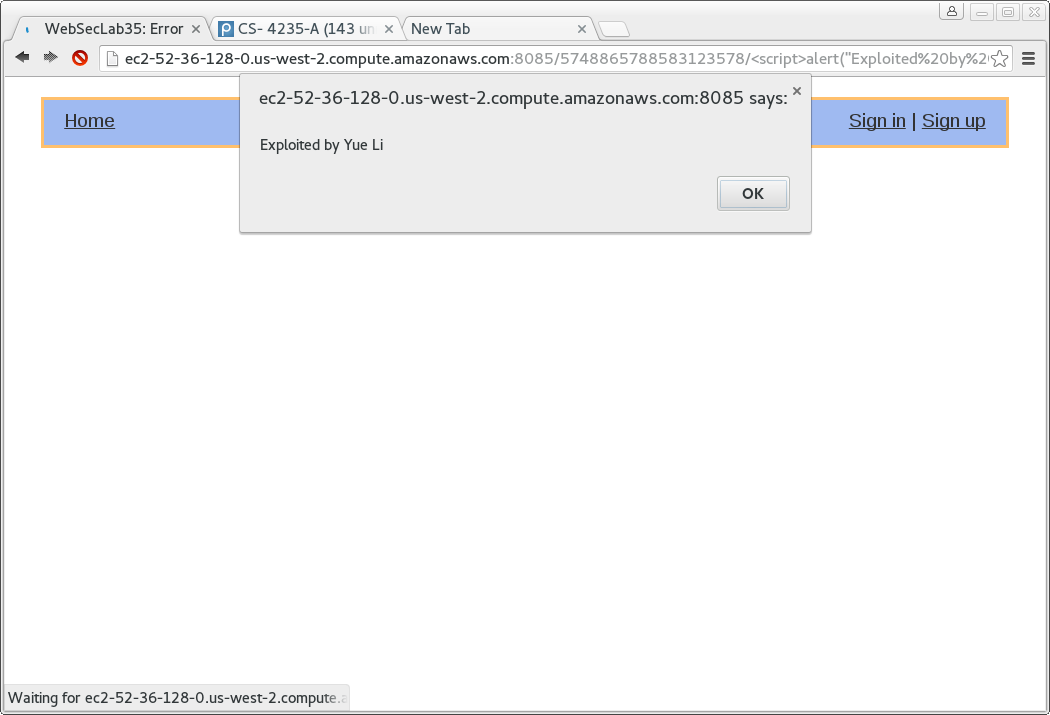
**Lab 2 Report**

Yue Li 903145936 Shujun Bian 9031XXXXX

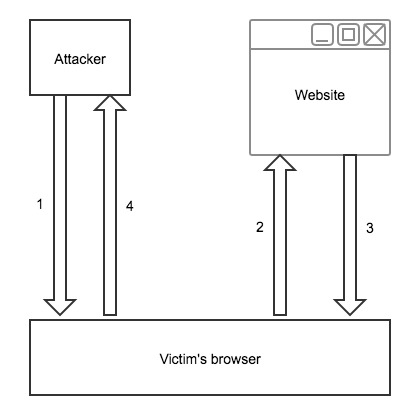
Problem One: Exploits (15 Points Total)

Subproblem 1: Reflected XSS (5 Points)

1. Try the exploit yourself and provide snapshots. The snapshot should show your signature (something about you uniquely). (1.5 points)



1. Draw a diagram of the exploit and explain the procedure. (1 point)



1.Attacker would send some links to the victim which including some malicious code in the URL. For example, the attacker would send a email saying “Check this out: http://ec2-52-36-128-0.us-west-2.compute.amazonaws.com:8081/5272665705106637995<script>… </script>”. And the keyword is <script>…</script>

2. The victim is tricked by the attacker into requesting the URL from the website.

3. The malicious part would response from that URL which normally considered as a trusted server.

4. The victim's browser would execute the malicious script inside the response, because it comes from some trusted website. As long as the browser executed the malicious code, the victim's cookies would be sent to the attacker's server, where could include a lot of important information like username, password and so on.

List 2 other forms of payload (e.g., encoding) that give the same result of the exploit (i.e., a popped dialog defined by the attacker when the victim accesses a link). (1 points)

1. Tag Attribute Value

Sometimes the web application could use the user input value to fill an attribute, which doesn’t contain the <script> tag in the URL. For example, put the **<input type="text" name="state" value="Input your username">** in the URL.

2. Using code encoding

We may encode our script in base64 and place it in <meta> tag.

For example, <meta http-equiv ="refresh"

content="**0;url=data:text/html;base64,PHNjcmlwdD5hbGVydCgndGVzdDMnKTwvc2NyaXB0Pg**">. In this case, we do not need any tags likes <script> or functions like alert();

1. What is the cause of this exploit? Any quick fix? (1.5 points)

Because HTML documents have a flat, serial structure that mixes control statements, formatting, and the actual content, any non-validated user-supplied data included in the resulting page without proper HTML encoding, may lead to markup injection. Reflected XSS occurs when user input is immediately returned by a web application in an error message, search result, or any other response that includes some or all of the input provided by the user as part of the request, without that data being made safe to render in the browser, and without permanently storing the user provided data. One quick fix could be the validating (or sanitizing) user input to ensure that input is safe. For example, we could filter the input with certain patterns (using the regular expression) like starting with <script> tag and ending with the <script> tag. Another quick fix could be the character encoding, so that when a web browser encounters the entities like <scrip> tags, they will be converted back to HTML and printed but they will not be run. In the other word, the browser considers users’ input as data instead of executable code.

Subproblem 2: Stored XSS (5 Points)

|  |
| --- |
| Bug report |
| Name: Stored XSS |
| Description, payload (2 points):  We can apply a stored XSS attack by putting the alert message into onmouseover attributes in the snippets.  Payload:  <a onmouseover="alert('Exploited by Yue Li :D')">Hover Here!</a> |
| Snapshots (Try to be explicit, highlight the key parts) (2 points): |
| Pseudo Code to fix the bug from server side (1 point): |

Subproblem 3: Cross-Site Request Forgery (5 points)

|  |
| --- |
| Bug report |
| Name: CSRF |
| Description, payload (2 point):  We can apply a CSRF by putting the snippets deletion link (/123…/deletesnippet?index=0) into the snippets, and for any user click the link, the most recent snippets he added will be deleted.  Payload:  <a href='/4944131894132192352/deletesnippet?index=0'>Don't click me!</a> |
| Snapshots (Try to be explicit, highlight the key parts) (2 points):  1) Logged in as Ian, before click the link:    2) Click the link provided by user YueLi:    3) After the click: |
| Detail the common practice to thwart CSRF (1 points): |

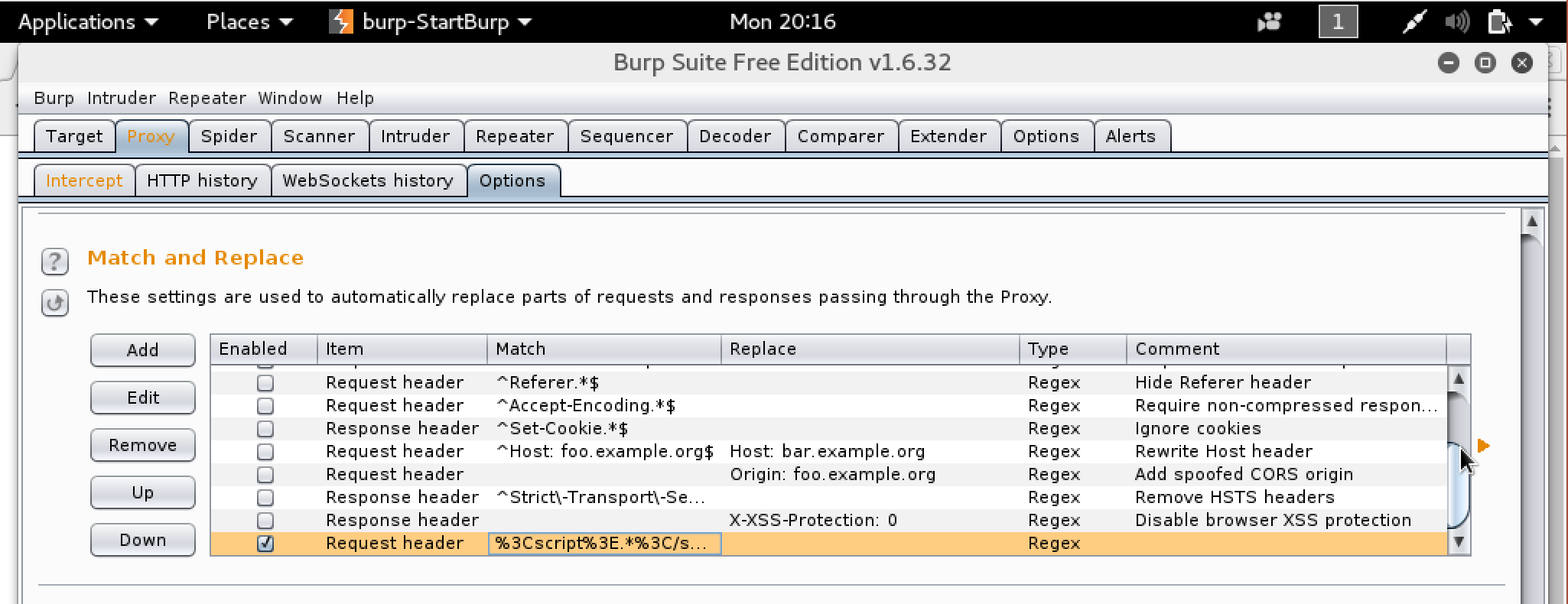
Problem Two: Sanitizer (5 Points Total)

1. What is the detailed procedure for setting this rule? (1 point)

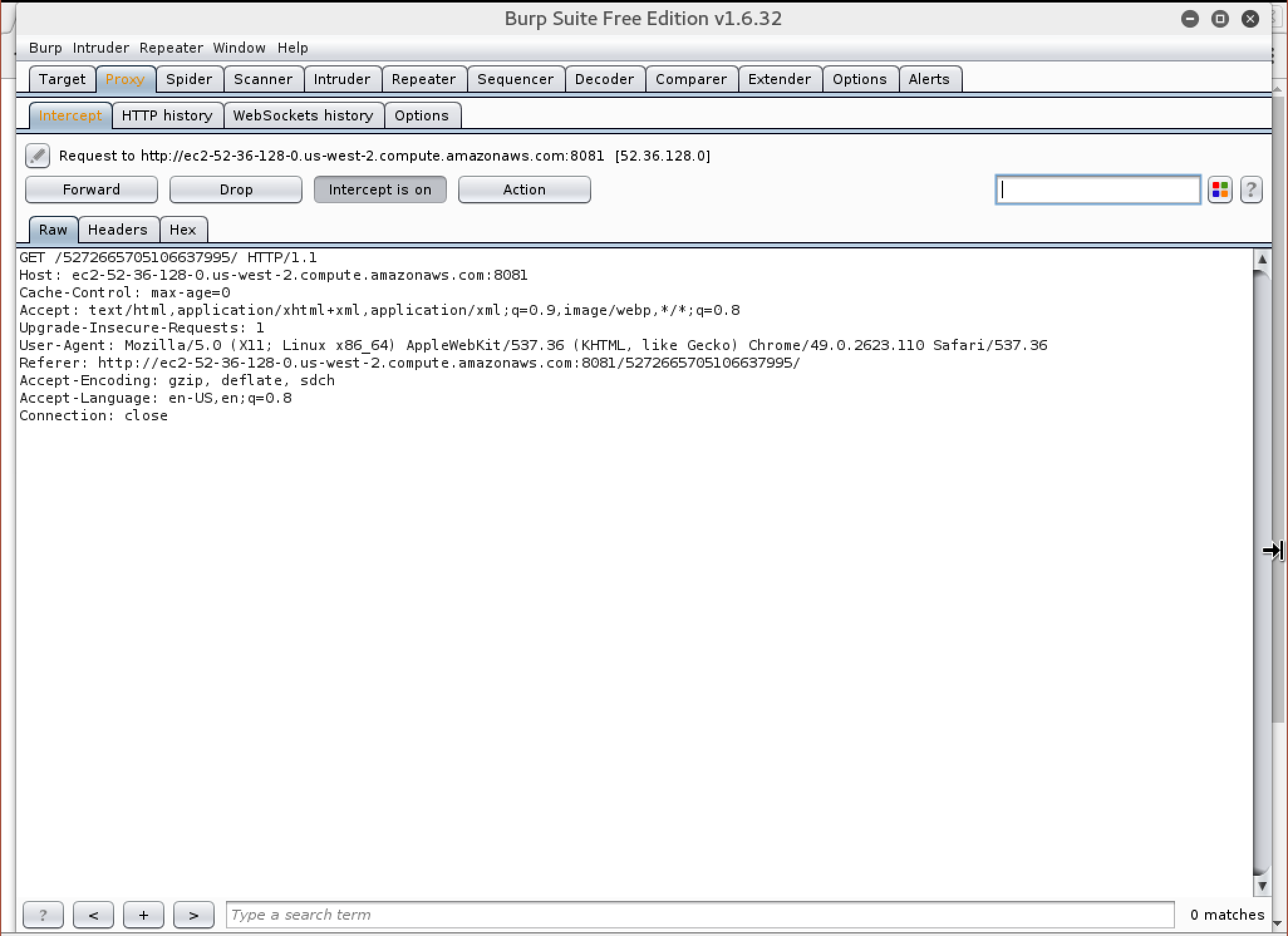
Open Burpsuite, then go to the *Options* tab in *Proxy* tab. Click the *Add* button in the left. Add a new intercept rules for the reflected XSS.

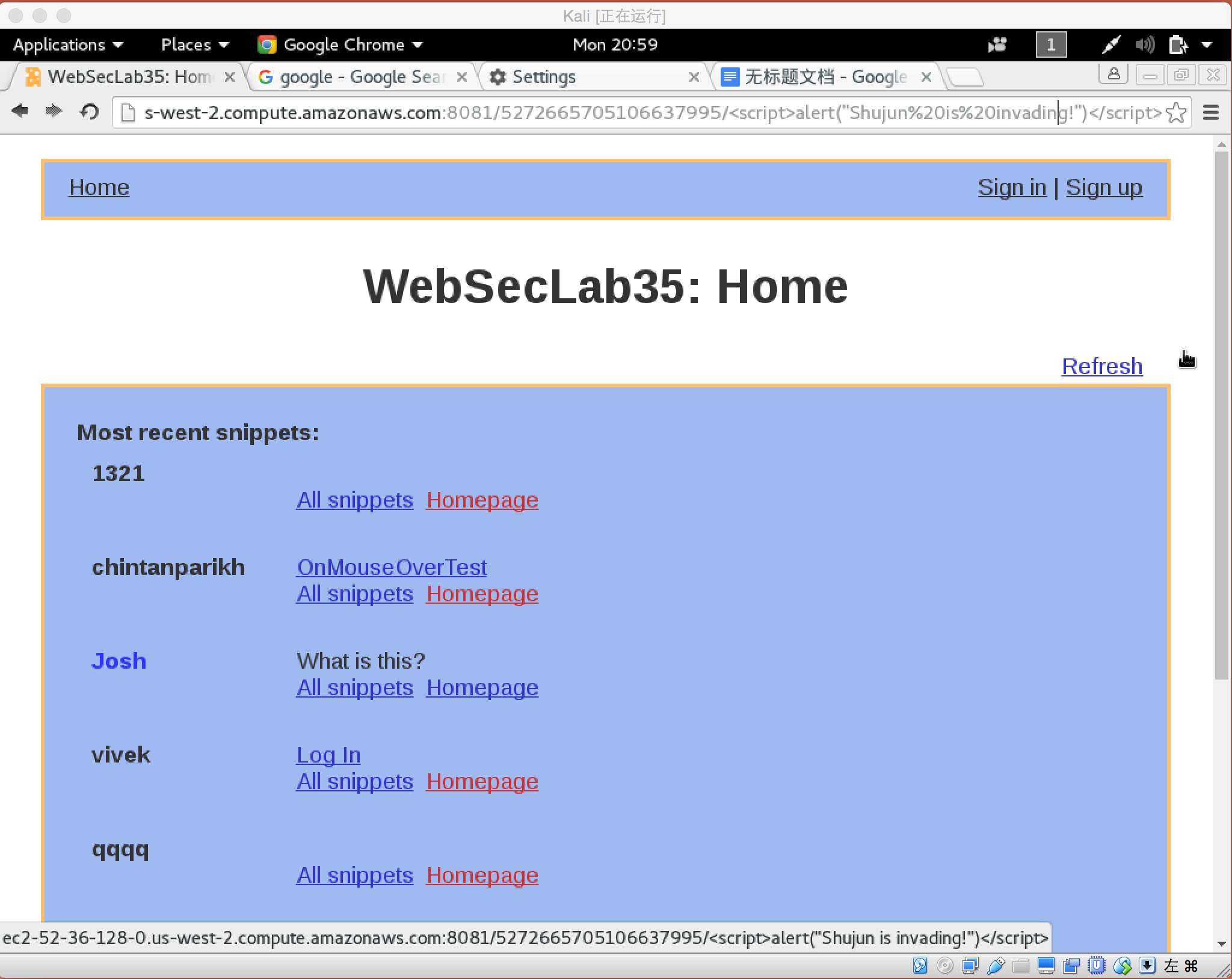
For example, In Subproblem 1 of Problem One, the malicious link is like: http://ec2-52-36-128-0.us-west-2.compute.amazonaws.com:8081/5272665705106637995/%3Cscript%3Ealert(%22Shujun%20is%20invading!%22)%3C/script%3E

We could find that the malicious part is “%3Cscript%3Ealert(%22Shujun%20is%20invading!%22)%3C/script%3E”, which starts from “%3Cscript%3E”(<script>) and ends at %3C/script%3E(</script>), so we could write a regular expression to detect this kind of malicious link. Then the Burpsuite would remove the matched part from the whole link. In this case, the Burpsuite would remove the “%3Cscript%3Ealert(%22Shujun%20is%20invading!%22)%3C/script%3E”, then the link turns to be http://ec2-52-36-128-0.us-west-2.compute.amazonaws.com:8081/5272665705106637995.



1. Provide screen captures that show that your rule stopped an attempt at the exploit. (2 points)



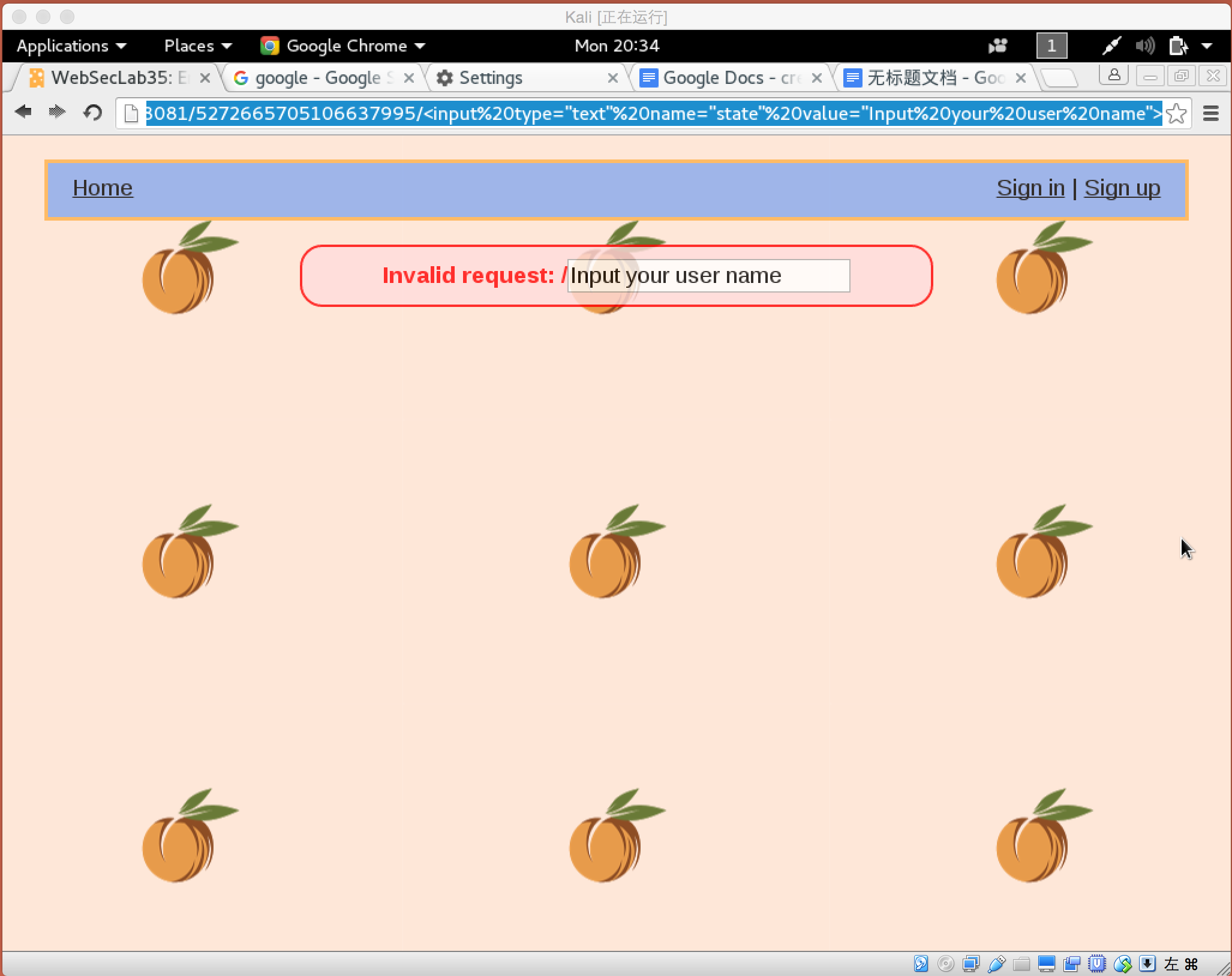


So we could see that when we try to visit the http://ec2-52-36-128-0.us-west-2.compute.amazonaws.com:8081/5272665705106637995/%3Cscript%3Ealert(%22Shujun%20is%20invading!%22)%3C/script%3E, the Burpsuite automatically remove the malicious part, and everything works well.

1. What is the limit or side effect of your solution? Provide snapshots of examples to support your arguments. (2 points)

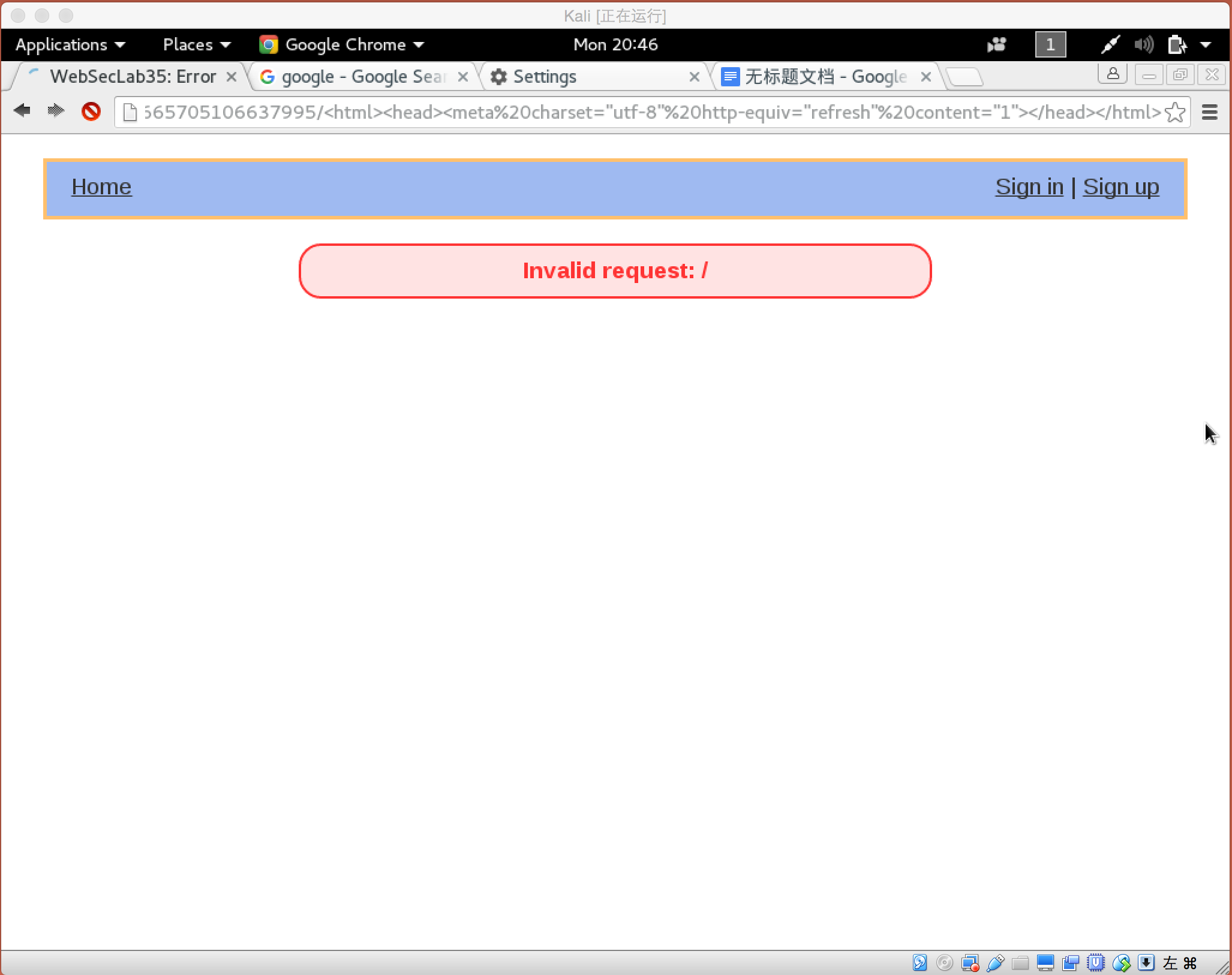
Because I only detect one mode of malicious code which starts with “<script>” and ends with “</script>”, some reflected XSS could pass the interception. We could try to write some different kinds of Javascript without the script tag.

For example, we could use “<input type="text" name="state" value="Input your user name">” instead of “<script>…</script>”.



As we can see, this <input> tag is showing on the website, which means it passed our interception.

Another example for different pattern of Javascript could be “<html><head><meta charset="utf-8" http-equiv="refresh" content="1"></head></html>”, so that the website would refresh automatically every 1 second. This kind of reflected XSS also doesn’t contain any script tag.



**References**

<https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)>

<https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)>

<https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting>

<https://google-gruyere.appspot.com/part2>

<http://excess-xss.com/>

<https://portswigger.net/burp/help/suite_gettingstarted.html>