

# CS4235 / CS6035 Intro to InfoSec

## Solution template for Lab 2 Web Security

Group members (at most 3): Shen Yang, Benjamin Southern

gtid(s): 902912328, 902979774

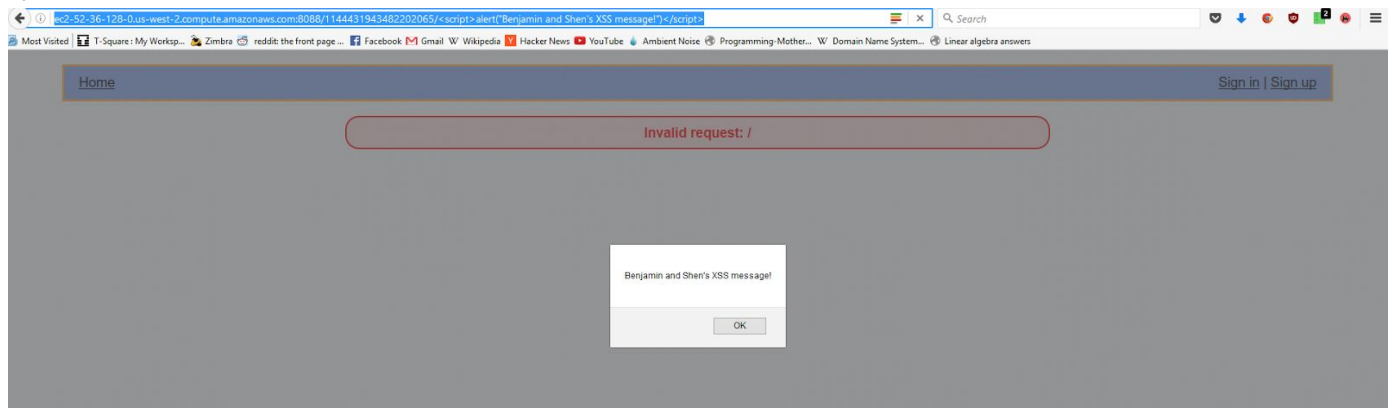
Note: Please submit PDF file.

### Problem One: Exploits (15 Points Total)

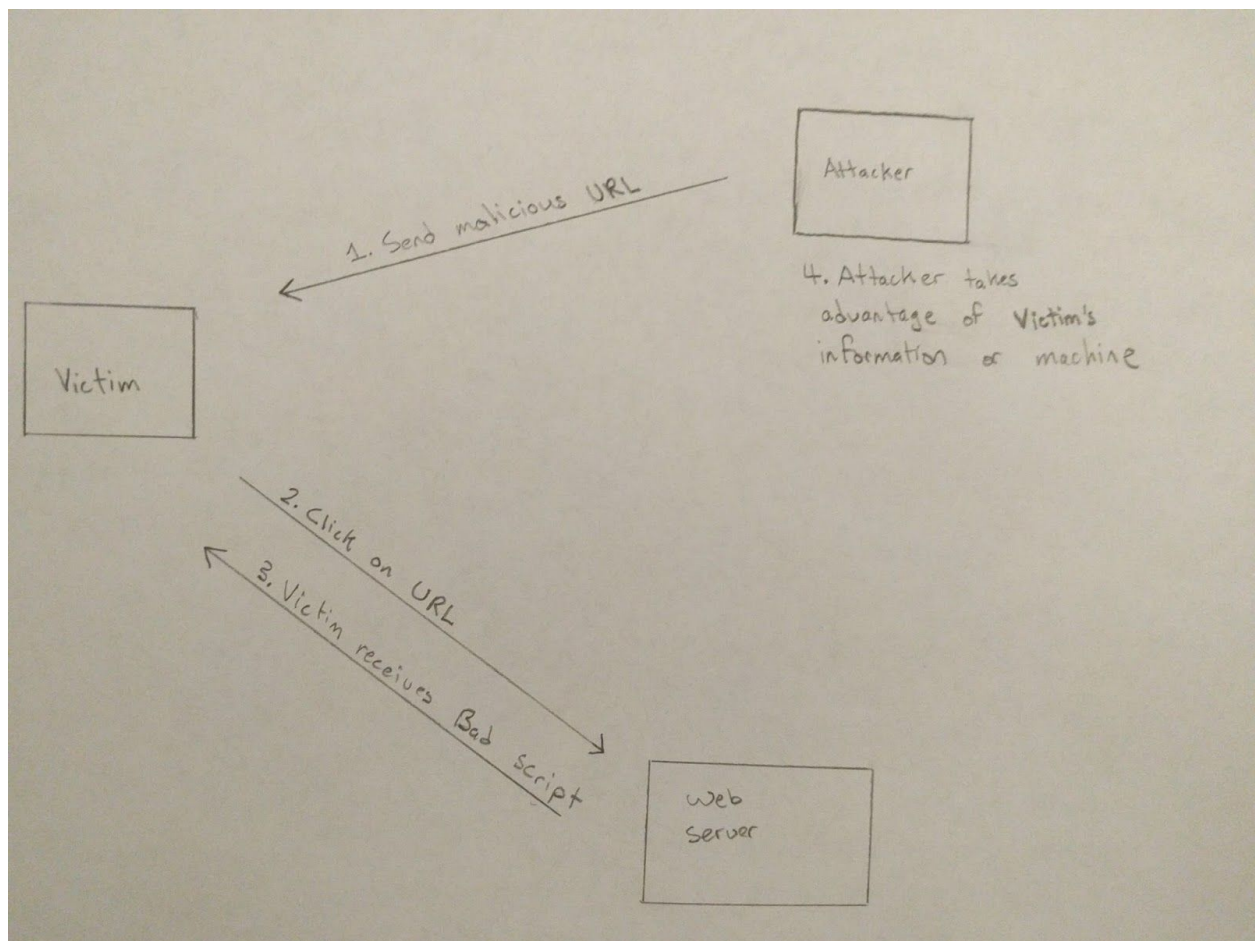
#### Subproblem 1: Reflected XSS (5 Points)

- Try the exploit yourself and provide snapshots. The snapshot should show your signature (something about you uniquely). (1.5 points)
- Draw a diagram of the exploit and explain the procedure. (1 point)
- 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)
- What is the cause of this exploit? Any quick fix? (1.5 points)

a)



b)



With the url as

<http://ec2-52-36-128-0.us-west-2.compute.amazonaws.com:8080/5272665705106637995/This%20site%20does%20not%20exist>, and that the request is invalid, the request is being added to a message div that tells the user the certain request is invalid.

```
140
141
142 <div class='message'>Invalid request: /This site does not exist</div>
143
144
```

However, if the request is a javascript, it gets executed when printed, thus popping a dialog on the browser.

```
140
141
142 <div class='message'>Invalid request: /<script>alert("You are exploited!")</script></div>
143
144
```

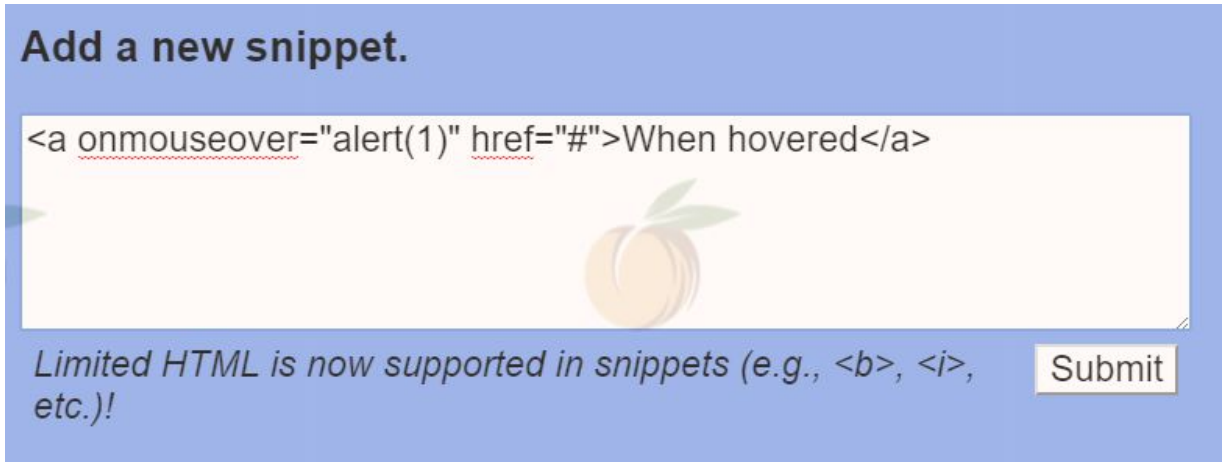
c)

http://ec2-52-36-128-0.us-west-2.compute.amazonaws.com:8081/5272665705106637995/%3Csvg/onload=alert(%22Hello!%22)%3E

http://ec2-52-36-128-0.us-west-2.compute.amazonaws.com:8081/5272665705106637995/%3CBODY%20ONLOAD=alert('XSSmessage')%3E

d) This exploit is caused due to a naive trust of user input. In this XSS attack data is sent to the web server without being validated. There are many ways to fix it, including parsing the request and escaping characters. The web application server should analyze requests by users.

### Subproblem 2: Stored XSS (5 Points)

Bug report
Name: Stored XSS
<p>Description, payload (2 points):</p> <p>Payload:</p> <pre>&lt;a onmouseover="alert(1)" href="#"&gt;When hovered&lt;/a&gt;</pre> <p>The above payload is inserted into the snippet text. When the user hover across the text "When hovered", the javascript will execute resulting in a dialog box appearing.</p>
<p>Snapshots (Try to be explicit, highlight the key parts) (2 points):</p> <div></div>

## Most recent snippets:

yolo

When hovered

All snippets Homepage

ec2-52-36-128-0.us-west-2.compute.amazonaws.com:8082 says: ×

1

☐ Prevent this page from creating additional dialogs.

OK

Pseudo Code to fix the bug from server side (1 point):

```
chars = {'"': '&quot;', '\\': '&#39;', '&': '&amp;', '<': '&lt;', '>': '&gt;'}
```

```
for each in input:
```

```
  if each in chars:
```

```
    replace(each, chars[each])
```

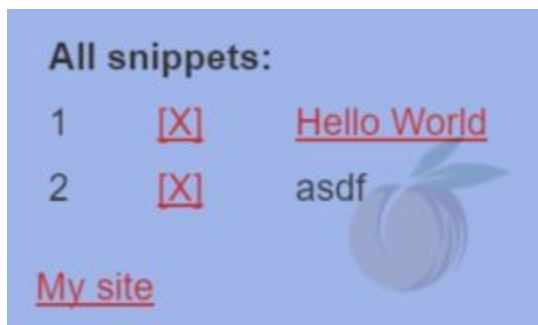
```
return each
```

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

Bug report
Name: CSRF
Description, payload (2 point):  Payload: <a href="http://ec2-52-36-128-0.us-west-2.compute.amazonaws.com:8082/52726657051066395/deletesnippet?index=1">Hello World</a>

When clicked on the text “Hello World”, it actually goes to the link which deletes the snippet index 1.

Snapshots (Try to be explicit, highlight the key parts) (2 points):



Detail the common practice to thwart CSRF (1 points):

Having a CAPTCHA or some other challenge-response defense for the user to complete to confirm the action.

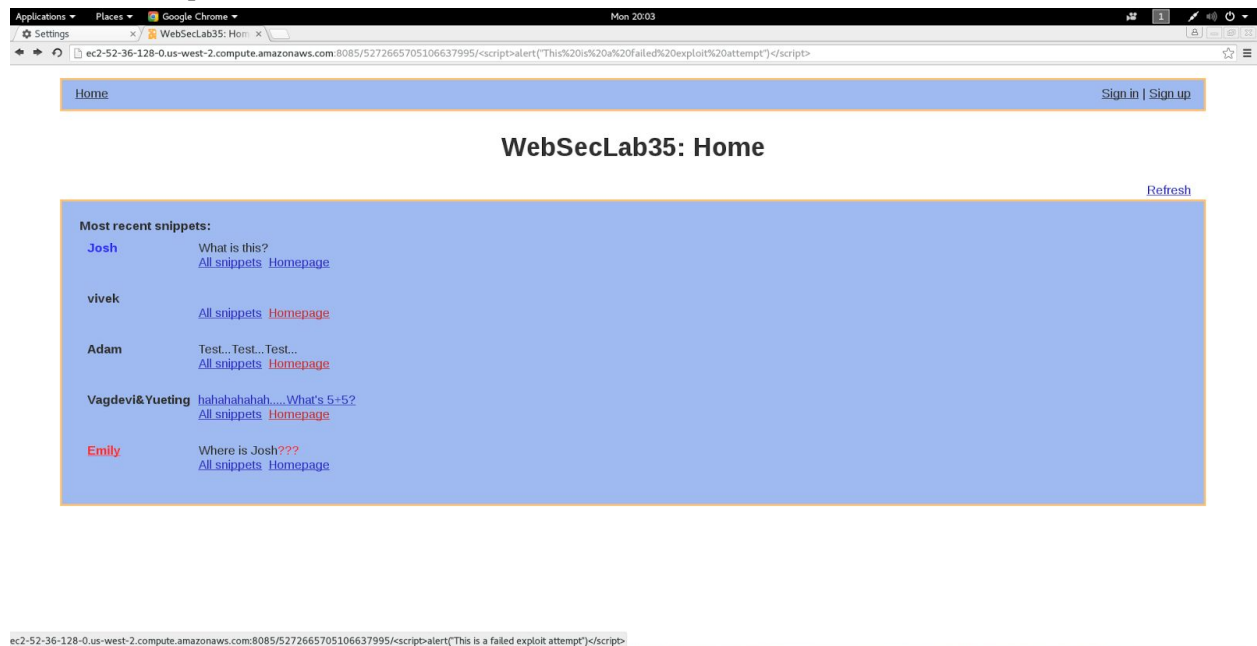
## Problem Two: Sanitizer (5 Points Total)

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

First, I had to configure Kali to use the Burp proxy. This was done in the “Network” subsection of settings. Chrome uses the system proxy settings, so by changing the host computer’s settings I was able to also effectively change the chrome proxy configuration. Next, we had to add a “match and replace” rule in Burp. This is done under the Options subtab within Proxy. If you scroll down to the Match and Replace section, we can add a regex rule to remove any script from a request header. The regex rule I used was match: “(%3C)script.\*” and replace: “”.

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

We can observe in the following screenshot that even though the user has tried to access a URL with a reflected XSS attack, Burp has removed the injected script from the request header. The page has loaded without the script.



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

This method of sanitization only eliminates XSS attacks that make use of the `<script>` tag. Attacks that do not use `<script>` will still work. For example, the methods detailed in part 1.c of this lab works even if the rule is in place.

