Software Security
Web vulnerabilities, injection attacks

Alessandra Gorla

institute IMdea software

Goals for today

- Injection vulnerabilities
  - SQL injections
  - Cross-site scripting attacks
  - ...

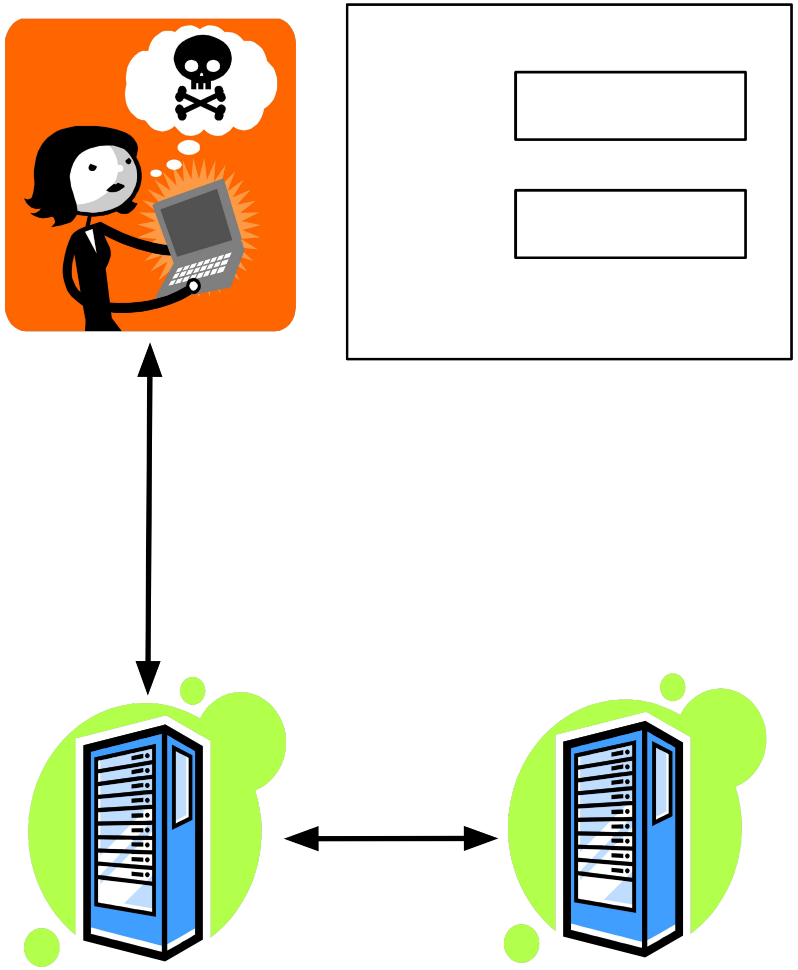
Injections

- Injection attacks trick an application into including unintended commands in the data sent to an interpreter.
- Interpreters
  - Interpret strings as commands.
  - Ex: SQL, shell (cmd.exe, bash), LDAP, XPath
- Key Idea
  - Input data from the application is executed as code by the interpreter.

SQL injections

SQL injection

1. App sends form to user.
2. Attacker submits form with SQL exploit data.
3. Application builds string with exploit data.
4. Application sends SQL query to DB.
5. DB executes query, including exploit, sends data back to application.
6. Application returns data to user.



SQL injection

```
$link = mysql_connect($DB_HOST, $DB_USERNAME, $DB_PASSWORD) or die ("Couldn't connect: " . mysql_error());

mysql_select_db($DB_DATABASE);

$query = "select count(*) from users where username = '$username' and password = '$password‘ ";

$result = mysql_query($query);
```

SQL injection attack #1

Unauthorized Access Attempt:

```
password = ’ or 1=1 --
```

SQL statement becomes:

```
select count(*) from users where username = ‘user’ and password = ‘’ or 1=1 --
```

Checks if password is empty OR 1=1, which is always true, permitting access.

SQL injection attack #2

Database Modification Attack:

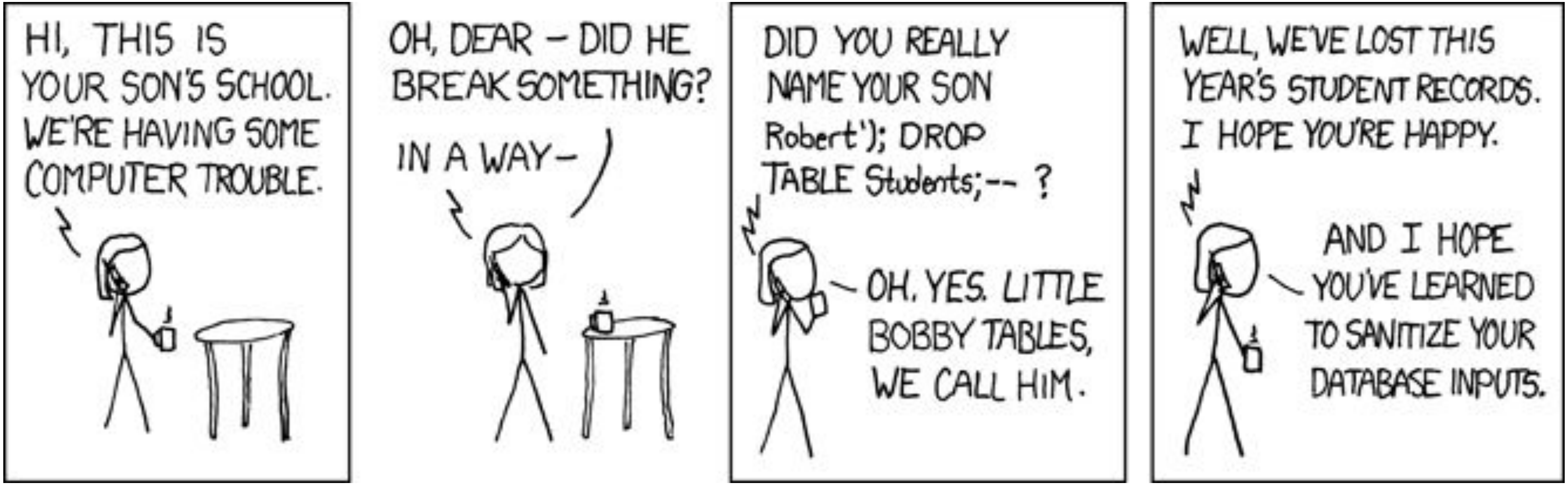
```
password =  foo’; delete from table users where username like ‘%
```

DB executes two SQL statements:

```
select count(*) from users where username = ‘user’ and password = ‘foo’

delete from table users where username like ‘%’
```

Exploits of a mum


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Finding SQL injection bugs

- Submit a single quote as input.
  - If an error results, app is vulnerable.
  - If no error, check for any output changes.
- Submit two single quotes.
  - Databases use ’’ to represent literal ’
  - If error disappears, app is vulnerable.

Inserting into SELECT

- Most common SQL entry point.

```
SELECT columns
        FROM table
        WHERE expression
        ORDER BY expression
```

1. Places where user input is inserted:

   WHERE expression

   ORDER BY expression

   Table or column names

Inserting into INSERT

Creates a new data row in a table.

INSERT INTO table (col1, col2, ...)

VALUES (val1, val2, ...)

Requirements

- Number of values must match # columns.
- Types of values must match column types.

Technique: add values until no error.

```
foo’)--
foo’, 1)--
foo’, 1, 1)--
```

Inserting into UPDATE

Modifies one or more rows of data.

```
UPDATE table
    SET col1=val1, col2=val2, ...
    WHERE expression
```

Places where input is inserted

- SET clause
- WHERE clause

Be careful with WHERE clause

- ’ OR 1=1 will change all rows

Inference attacks

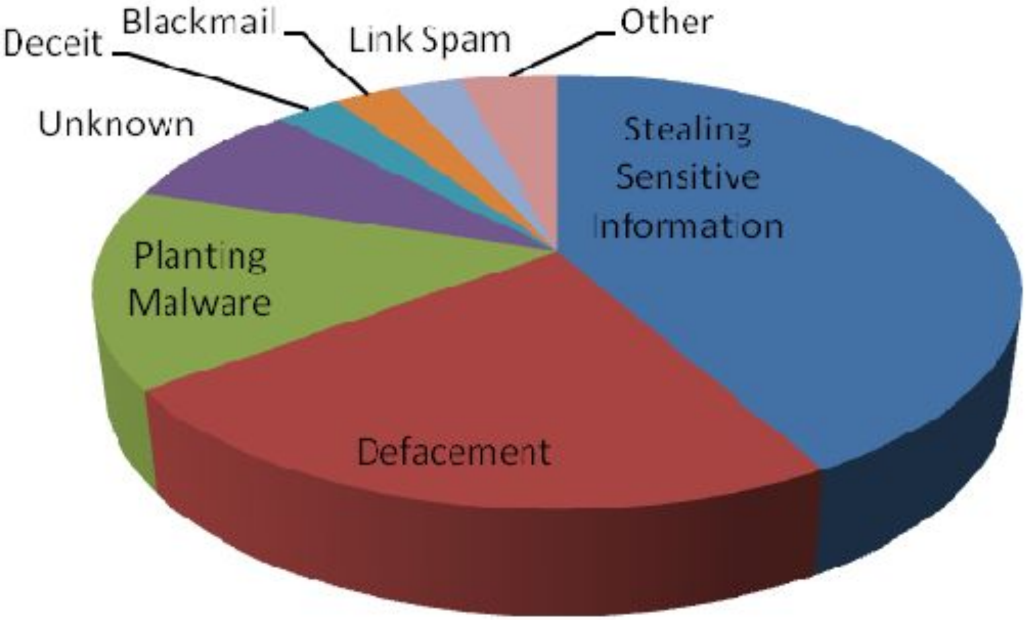
Problem: What if app doesn’t print data?

Injection can produce detectable behavior

- Noticeable time delay or absence of delay.

Impact of SQLI

1. Leakage of sensitive information.
2. Reputation decline.
3. Modification of sensitive information.
4. Loss of control of db server.
5. Data loss.
6. Denial of service.



Root cause of the issue

Building a SQL command string with user input in any language is **dangerous**.

Mitigating SQL Injections

- **Ineffective Mitigations**
    - Blacklists
- **Partially Effective Mitigations**
    - Whitelists
    - Prepared Queries

Blacklists

- Filter out or Sanitize known bad SQL meta-characters, such as single quotes.
- But...
  - URL escaped metacharacters.
  - Unicode encoded metacharacters.
  - Did you miss any metacharacters?
- **Though it's easy to point out some dangerous characters, it's harder to point to all of them.**

Bypassing filters

- Different case
  - `SeLecT instead of SELECT or select`
- Bypass keyword removal filters
  - `SELSELECTECT`
- URL-encoding
  - `%53%45%4C%45%43%54`
- SQL comments
  - `SELECT/*foo*/num/*foo*/FROM/**/cc`
  - `SEL/*foo*/ECT`
- String Building
  - `‘us’||’er’`
  - `chr(117)||chr(115)||chr(101)||chr(114)`

Whitelists

- Reject input that doesn’t match your list of safe characters to accept.

- Identify what is good, not what is bad.
- Reject input instead of attempting to repair.

Prepared queries

- An SQL statement string is created with placeholders and it's compiled into an internal form.
- Later, this prepared query is "executed" with a list of parameters.

Example in perl

```
$sth = $dbh->prepare("SELECT email, userid FROM members WHERE
email = ?;");

$sth->execute($email);
```

Prepared queries

- Bound parameters in Java

Insecure version

```
Statement s = connection.createStatement();

ResultSet rs = s.executeQuery("SELECT email FROM member WHERE name = " +
formField);
```

Secure version

```
PreparedStatement ps = connection.prepareStatement( "SELECT email FROM member
WHERE name = ?");

ps.setString(1, formField);

ResultSet rs = ps.executeQuery();
```

Other injection types

- Shell injection.
- Scripting language injection.
- File inclusion.
- XML injection.
- XPath injection.
- LDAP injection.
- SMTP injection.

XSS injections

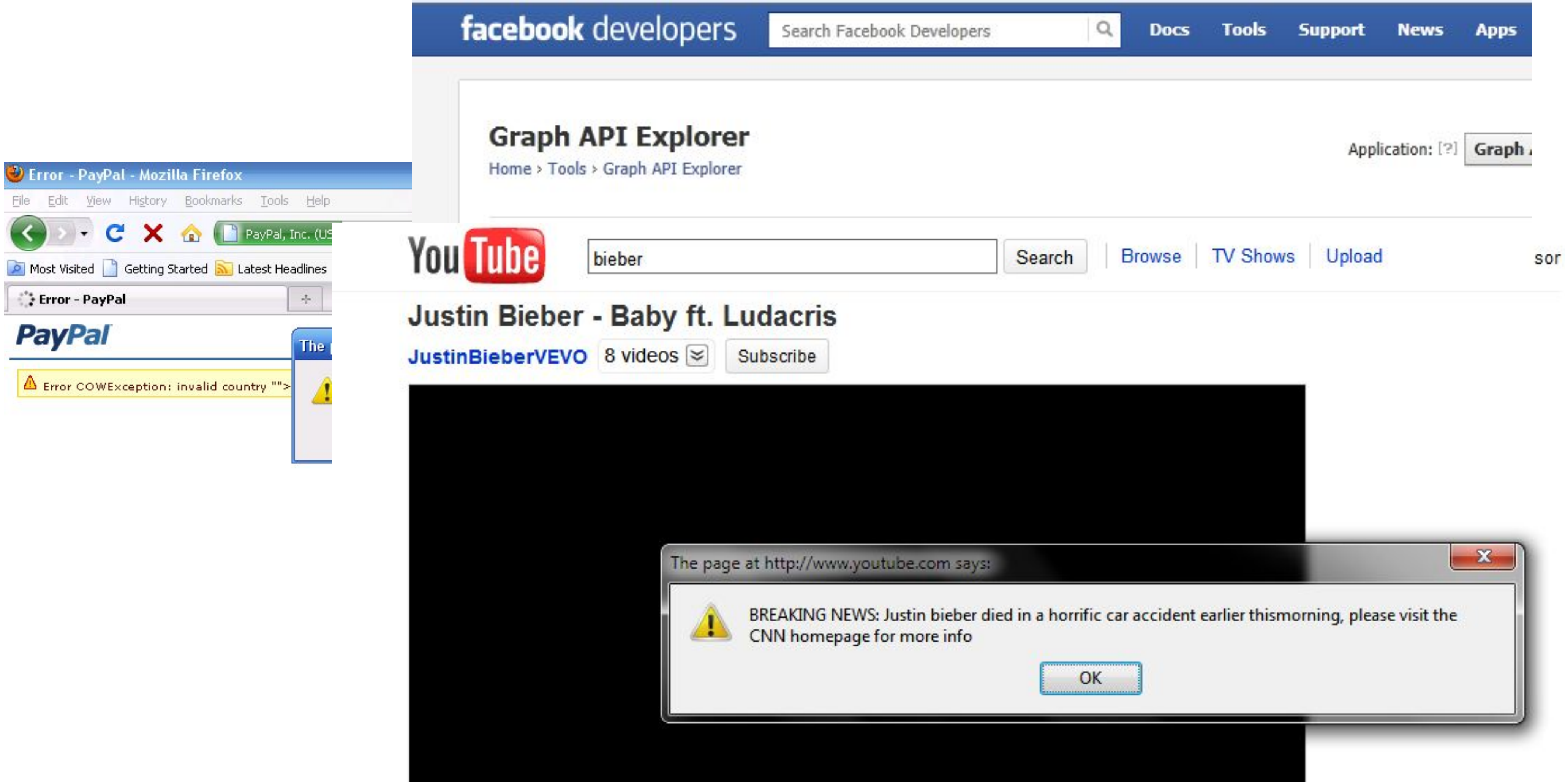
Cross-site scripting

- Malicious client-side script is injected into the application output and subsequently executed by the user's browser
    - *i.e. Not filtering out HTML and JavaScript in user input = bad*
- It can be used to take over a user's browser in a variety of ways

Should we really worry about XSS?

- XSS used to be considered a low-risk type of security issue,
- Then.. XSS attacks have increased in prominence and sophistication

familiar names...



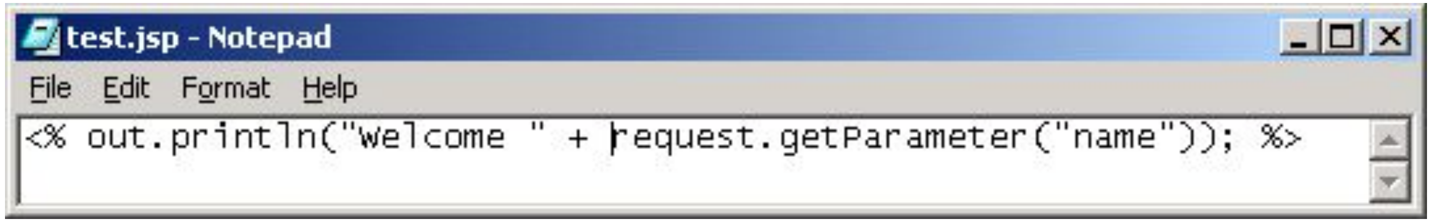
Error COWException: invalid country "">

Justin Bieber - Baby ft. Ludacris

The page at http://www.youtube.com says:

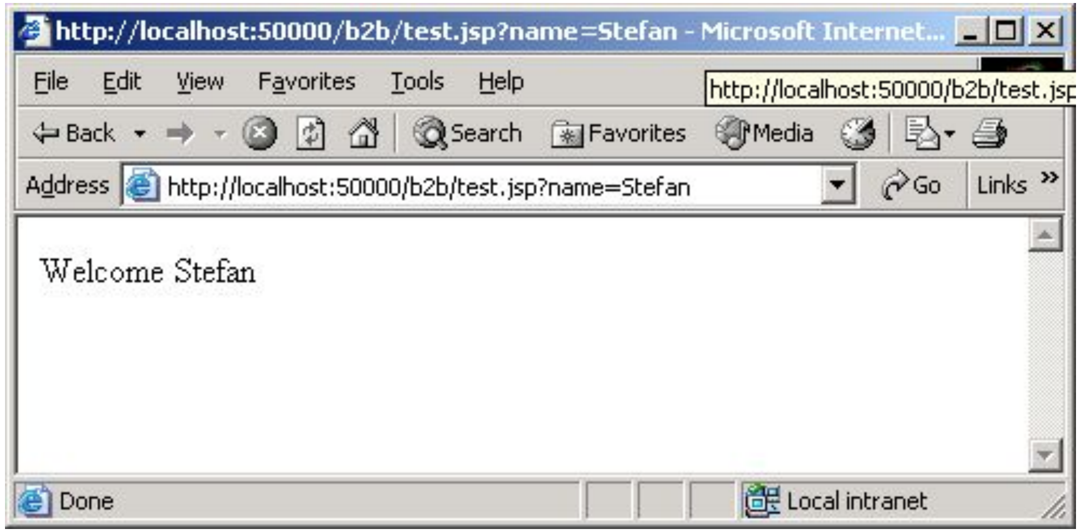
BREAKING NEWS: Justin bieber died in a horrific car accident earlier thismorning, please visit the CNN homepage for more info

A simple example



```
<% out.println("welcome " + request.getParameter("name")); %>
```

http://myserver.com/test.jsp?name=Stefan



http://localhost:50000/b2b/test.jsp?name=Stefan - Microsoft Internet...
File
Edit
View
Favorites
Tools
Help
http://localhost:50000/b2b/test.jsp
Back
Search
Favorites
Media
Go
Links
Done
Local intranet

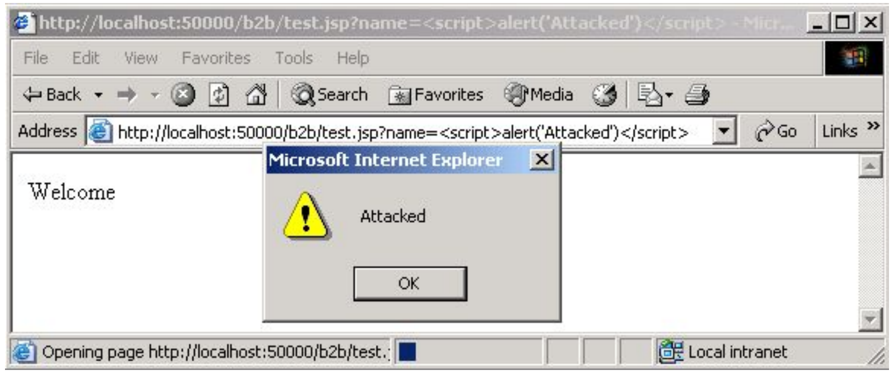
Address: http://localhost:50000/b2b/test.jsp?name=Stefan

Welcome Stefan

```
<HTML>
<Body>
Welcome Stefan
</Body>
</HTML>
```

A simple example

http://myserver.com/welcome.jsp?name=<script>alert("Attacked")</script>



http://localhost:50000/b2b/test.jsp?name=<script>alert('Attacked')</script> - Micr...

Address: http://localhost:50000/b2b/test.jsp?name=<script>alert('Attacked')</script>

Welcome

Attacked

Opening page http://localhost:50000/b2b/test.:

```
<HTML>
<Body>
Welcome <script>alert("Attacked")</script>
</Body>
</HTML>
```

Are XSS dangerous?

Just think, any JavaScript you want will be run in the victim's browser in the context of the vulnerable web page

what can you do with JavaScript?

- Steal cookies
- Pop-up alerts and prompts
- Access session tokens
- Detect installed programs
- Detect browser history
- Capture keystrokes
- Redirect to a different web site

- Detect if the browser is being run in a virtual machine
- Rewrite the status bar
- Exploit browser vulnerabilities
- Get user-agent string
- Determine if they are logged on to a particular site
- Capture clipboard content
- See enabled plugins (e.g. Chrome PDF viewer, Java, etc.)

How can we mitigate XSS?

Never trust the user!

How can we mitigate XSS?

- Almost all client-side script injection comes down to the following characters:

```
< > ( ) { } [ ] " ' ; / \
```

- Make sure you never display a user-entered string without properly encoding it

How can we mitigate XSS?

for instance in PHP

`$int = intval($_GET['a']);` // This will never return anything other than an integer

`$str = htmlentities($_GET['b']);` // This will encode any character for which there is an HTML entity equivalent (e.g. &gt; &lt; &quot;)