Special Topics in Security
ECE 5698

Engin Kirda
ek@ccs.neu.edu

Northeastern University

Admin Stuff and News

- Nearly done with Quiz 1 correction
  - Everything else is is online
- Today after class, Challenge 4 goes online
  - Should be an interesting challenge

Web Security III

Cache-Busting

- HTTP responses are subject to caching (e.g., by a proxy)
  - Suppose you have developed an application and are dropping a new cookie on request to URL:
  - http://sometesturl.com/welcome/register_for_info.cgi
  - A QueryString name/value pair is not used.
  - The request to the same URL might be considered a cache hit and the cached page (including the cookie!) might be returned to the user
- solution: cache-busting (i.e., prevent that certain data is cached)

Cache-Busting

- A popular technique when cookies are disabled:
  - If no session ID is presented, an HTTP Redirect (302) is issued and the client is redirected to session assignment
  - URL-encode a session ID through dynamic modification of the links in the page. e.g.:
  - http://www.helloworld.org/index.html ➔

    http//www.helloworld.org/index.html?session=123
  - This approach is vulnerable… if the page with hard-coded sessions is cached, many users may see each others sessions (!)
  - Furthermore, people may bookmark the links and this may cause more confusion

Cache-Busting

- Cookies can be used for flawless client tracking
  - If they are used in conjunction with SSL to prevent caching of *Set-Cookie* header
  - Avoid issuing Set-Cookie headers in response to HTTP requests that use the GET request method.
  - use *Cache-Control headers to forbid caching*
  - All sessions must expire after a short amount of inactivity (e.g., 20 minutes)

Cache-Busting

- The only guaranteed secure way to prevent cache-effects:
  - Use SSL as part of the process of meeting and greeting a new client
- The logic for initial SSL-secured HTTP request is simple:
  1.) If no valid session ID, redirect browser to https:// session generator: Cookie is set
  2.) Redirect browser to original http:// URL

  Because SSL is used, cookie cannot be cached.

  MS .NET Passport, Google, Facebook uses this approach

Garbage In - Garbage Out

- POST requests sent in connection with legitimate anonymous sessions carry some risk:
  - There is no way to know that data provided by the anonymous request is legitimate
  - Some people will use your form to send you “garbage”
  - Typical examples are feedback forms that generate e-mails and anonymous guest books.
  - Spammers may exploit such forms to advertise content or send unsolicited e-mails
  - note: typical for, but not restricted to POST requests (e.g., an application might also process GET requests in a similar fashion)

Prove You’re Human

- Every web site implicitly services requests from non-human clients (e.g., web crawlers)
- Requiring authentication does not provide proof that a human is responsible for server requests
  - Malicious code could launch brute-force attack to discover credentials
  - Malicious code could log in to existing account and perform actions such as sending mails (e.g., sending spam via free mail web interfaces)
- Hence, we need to implement a “prove you’re human” countermeasure (”CAPTCHA“: Completely Automated Public Turing Test to tell Computers and Humans Apart)

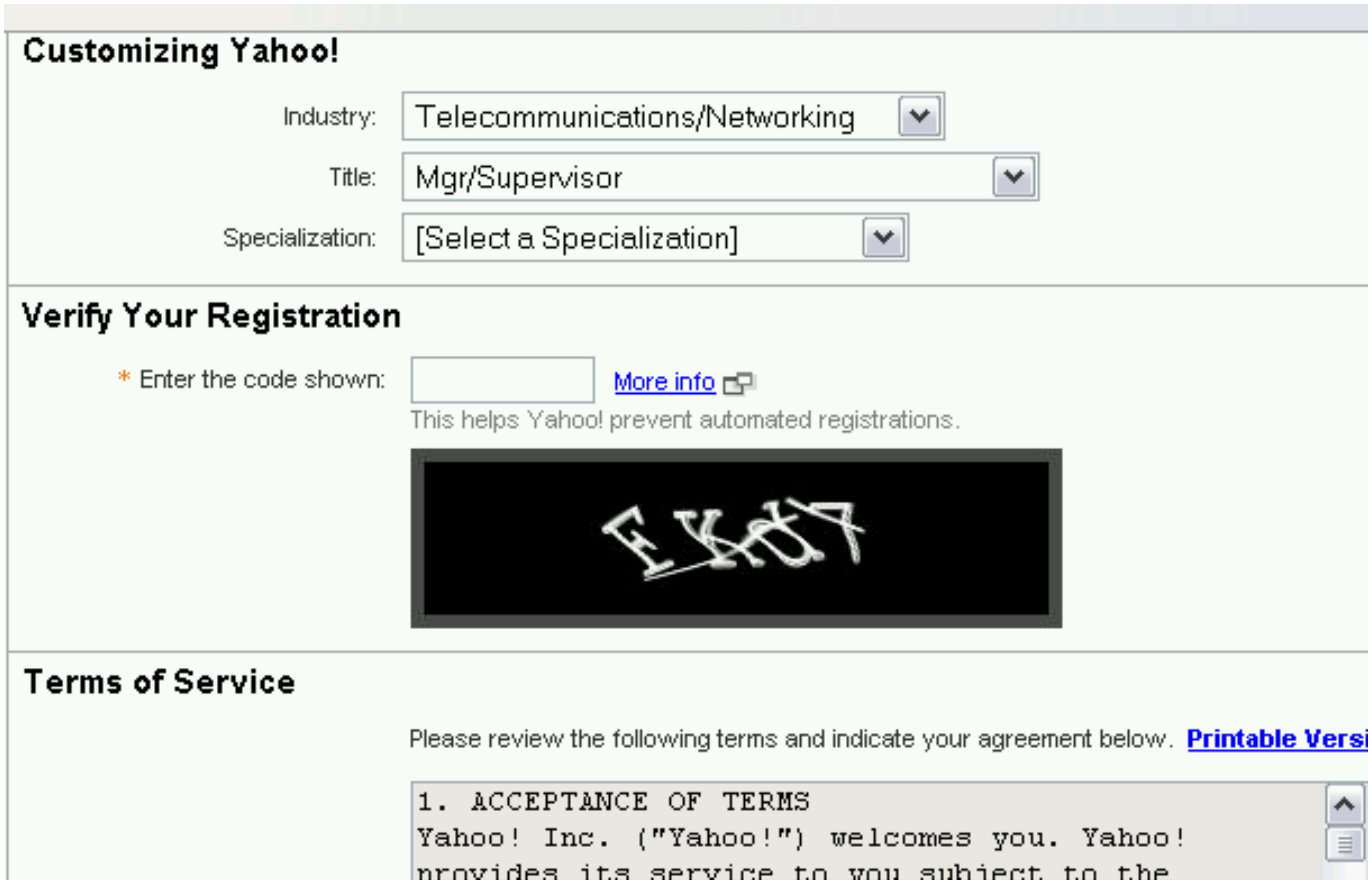
Prove You're Human

- The core features of any "prove you're human" countermeasure
  - Difficult for a computer program to respond with correct answer or action
  - Easy for a human to respond
- A popular method is to dynamically generate an image
  - Alphanumeric or numeric sequence
  - A form is displayed and the user is required to enter "challenge" displayed on the image

Prove You’re Human

- Are image-based prove you’re human mechanisms 100% safe?
  - Optical Character Recognition (OCR) algorithms can be used to analyze image and solve out challenge
  - This type of attack is sophisticated and not easy
  - It is important for the generated images to defeat OCR-based attacks.
  - Techniques used are font mixing, noise generation, and periodic changes to the visual characteristics of the images

Prove You're Human Example



Customizing Yahoo!

Industry:	Telecommunications/Networking
Title:	Mgr/Supervisor
Specialization:	[Select a Specialization]

Verify Your Registration

* Enter the code shown: More info

This helps Yahoo! prevent automated registrations.

Terms of Service

Please review the following terms and indicate your agreement below. **Printable Versi**

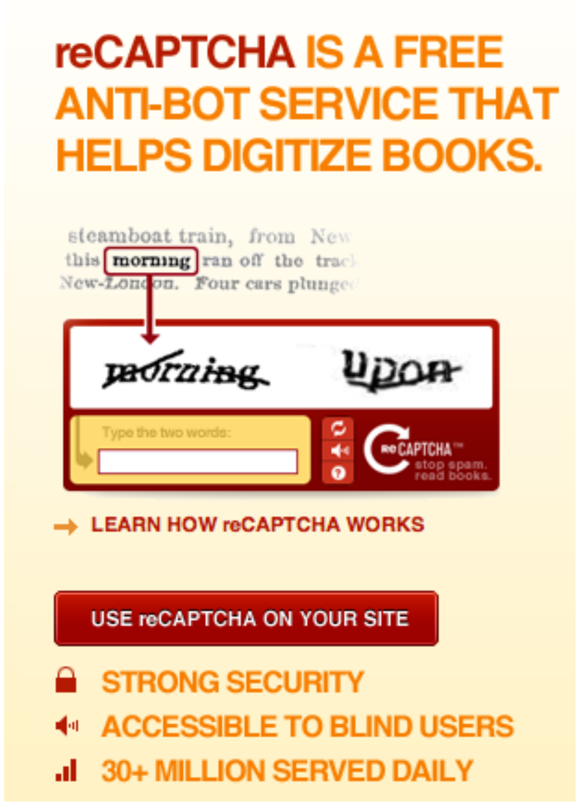
```
1. ACCEPTANCE OF TERMS
Yahoo! Inc. ("Yahoo!") welcomes you. Yahoo!
provides its service to you subject to the
```

Can you think of another way to bypass CAPTCHAs?

- OCR is a technical attack, but there are so-called “human” attacks
  - There are “CAPTCHA Farms” in certain countries where labor is cheap
  - Many people sit there and earn a couple of cents for every CAPTCHA solved
  - CAPTCHA farms are very effective in practice, and there is not much the provider can do against it
  - Other sites might trick users into solving a CAPTCHA in return for a service

A Good CAPTCHA Service to Use

- ReCAPTCHA
  - http://www.google.com/recaptcha
  - I am sure you have seen it
    - E.g., Facebook!
  - The idea is simple: If OCR cannot recognize it, then it is a good image
  - ReCAPTCHA takes words from books that could not be digitized. You are actually helping by solving CAPTCHAs



reCAPTCHA IS A FREE ANTI-BOT SERVICE THAT HELPS DIGITIZE BOOKS.

steamboat train, from New
this morning ran off the trac
New-London. Four cars plunge
morning upon
Type the two words:
reCAPTCHA
stop spam. read books.

LEARN HOW reCAPTCHA WORKS

USE reCAPTCHA ON YOUR SITE

STRONG SECURITY

ACCESSIBLE TO BLIND USERS

30+ MILLION SERVED DAILY

Broken Access Control

- Developers frequently underestimate the difficulty of implementing a reliable access control mechanism. Many of these schemes were not explicitly designed, but have simply evolved along with the web site.
  - Many of these flawed access control schemes are not difficult to discover and exploit
- One specific type of access control problem: administrative interfaces that allow site administrators to manage a site over the Internet.
  - Prime target for attacks

Broken Access Control

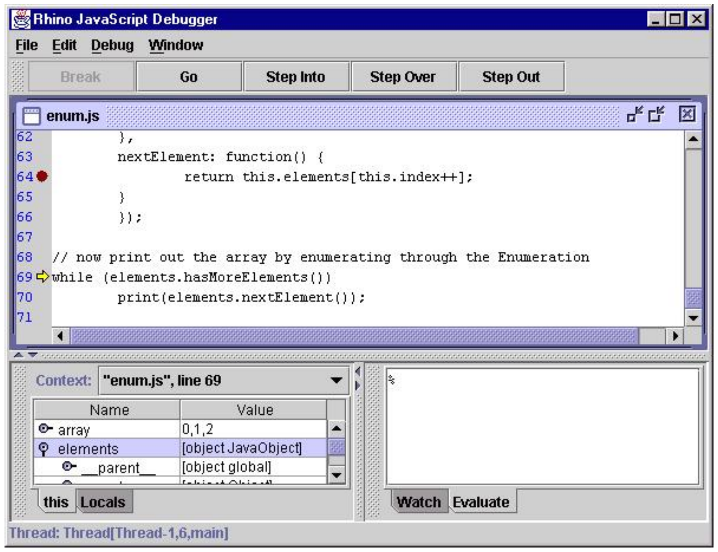
- Some specific access control issues:
  - Insecure ID’s (should not be guessable and no reliance on their secrecy)
  - Forced Browsing Past Access Control Checks (e.g., by using a different path through a different page)
  - Path Traversal e.g. “../../target_dir/target_file“
  - File Permissions (don‘t run server with root privileges!)
  - Client Side Caching (sensitive pages should not be cached, e.g., by using HTTP headers and meta-tags)

JavaScript

Unearthing the Excellence in JavaScript
JavaScript: The Good Parts
O'REILLY
YAHOO! PRESS
Douglas Crockford

Overview

- JavaScript language characterization
- Security policies
  - same-origin
  - code signing
- Browser vulnerabilities
  - implementation errors
  - design issues
- Interaction between client and server



```
        },
        nextElement: function() {
                return this.elements[this.index++];
        }
        });

// now print out the array by enumerating through the Enumeration
while (elements.hasMoreElements())
        print(elements.nextElement());
```

Context: "enum.js", line 69

Name	Value
array	0,1,2
elements	[object JavaObject]
__parent__	[object global]

Thread: Thread[Thread-1,6,main]

JavaScript

- Current version standardized as ECMA 357
- Most popular scripting language on the Internet
  - works with basically all browsers
- Designed to add interactivity to HTML pages
  - usually embedded directly into HTML pages `(<script>` tags)
  - dynamically add elements to page
  - can access elements of HTML page (DOM tree)
  - can react to events
- JavaScript is a scripting language
  - dynamic, weak typing
  - interpreted language
  - script executes on virtual machine in browser (with compilation)

JavaScript

- JavaScript is quite different from Java
  - Java is more complex and powerful
  - compiled language
  - static, strong typing
  - originally, JavaScript was named LiveScript
  - marketing department made developers change the name ;)

- Design decisions (Brendan Eich)
  - make it easy to copy and paste snippets of code
  - tolerate “minor” errors (missing semicolons)
  - simplified event handling
  - choose some powerful, often-needed primitives

JavaScript

- Syntax quite similar to Java
  - control statements, exception handling
- No classes, but object-based
  - uses objects with properties (name - value pairs)
- No input / output facilities per se
  - must be provided by embedding environment
- Scope of variables is either global or function-local
- Functions can be nested and are treated as objects
- Code can be generated at run-time and executed on-the-fly
  - `eval()` function

Security Policies

- Unknown code is downloaded to machine
  - always risky from security point of view
  - typically, user is asked for permission (not that it helps much :-))
  - impossible for JavaScript (too annoying, most pages use JS)
  - thus, special restrictions must apply

- JavaScript sandbox
  - no access to memory of other programs, file system, network
  - only current document accessible
  - might want to make exceptions for trusted code

- Basic policy for untrusted JavaScript code
  - *same-origin policy*

Same-Origin Policy

- *Access is only granted to documents downloaded from the same site as the script*
  - prevents hostile script from tampering other pages in browser
  - prevents script from snooping on input (passwords) to other windows
  - verify (compare) URLs of target document and script that access resource
- Domain comparison would not be trivial
  - use last two tokens of URL? [http://www.ccs.neu.edu]
  - use everything except the first token? [http://slashdot.org]
- Thus, checks are very restrictive
  - everything (including server name, port, and protocol) must match

Same-Origin Policy

Security problems

1. Browser vulnerability where policy is not enforced properly
   - bug in Mac OS X Safari browser (03/2006)
   - numerous ones for Internet Explorer and Mozilla
   - especially difficult when involving frames
   - quite common in early days

2. Problems with multiple parties on same site
   - one server can hold directories for different parties
     http://www.example.com/party1
     http://www.example.com/party2
   - no protection provided by same-origin policy in this case

Security Policies

- Browsers offer mechanisms to customize policies
  - Firefox and Internet Explorer allow general security policies
  - grant different capabilities to different origins
  - can be very fine-grained
  - often difficult and cumbersome to configure
  - IE security zones “low”, “medium”, “high”

Signed Scripts

- Introduce a mechanism so that trusted scripts can be run with elevated rights
  - allows access to file system and full control over browser
  - uses classic asymmetric cryptography
  - code provider obtains public / private key pair
  - publishes signed public key (certificate)
  - signs code using private key

A. Lincoln

- Signing does not imply that code is not malicious!
  - signatures can sometimes be easy to obtain
  - Did you hear of the COMODO case this year?
  - thus, most useful in restricted (corporate) Intranets

Browser Vulnerabilities

- Long and troubling history of bugs
  - Main way computers are compromised today
- Typically involve user privacy
  - access to file system
  - access to browser cache (previous surfed pages, URL strings)
  - access to browser preferences (email address, network settings)
  - frame information leak
  - session monitoring
  - forced sending of emails
- Also more severe bugs detected
  - upload and execution of arbitrary files
- Of course, known implementation holes are quickly closed
  - situation was worst between 1995 – 2000
  - Now, plugins are often attacked

Browser Vulnerabilities

- Design problems with JavaScript - malicious scripts
- Scripts can consume CPU resources
  - infinite loops
    - some browsers have heuristics that can stop such behaviour
    - unfortunately, detection of infinite loops undecidable in general case
- Scripts can consume memory
  - stack (space) overflow
    - can be easy caused by recursive functions

```
function f() {
  var x = 1;
  f();
}
```

  - allocating objects in infinite loops

Browser Vulnerabilities

- Scripts can keep browser busy
  - self-referencing `<frameset>` elements
    can cause infinite recursion of document fetches
  - pop up annoying number of alert messages
  - create windows that will `blur()` when receiving focus
  - create windows that can re-spawn on `unload()` events
    - Malicious websites may make use of these techniques

- Scripts that can be used to deceive user
  - pop up windows that mimic operating system messages
    trick user into downloading or installing malicious software
  - useful for phishing
    - map in padlock to pretend SSL connection is active
    - spawn window that overlays browser location bar to spoof actual URL

Developing JavaScript

- Protection of JavaScript source code
  - not possible, code is sent in plain text
  - futile attempts of JavaScript code protection
    - solution - disable view source, e.g., right mouse button menu
    - problem - this is not possible for certain browsers, e.g., Safari
    - solution - only send script to certain browsers
    - problem - browser information (or referrer) can be faked easily
    - problem - JavaScript can be disabled
  - code can be obfuscated (remember that eval is available)
    but de-obfuscation function must be included in code as well

→ don’t store secrets in the code

Developing JavaScript

- Password protection of pages

```
if (document.forms[0].elements[0].value == 'mypassword')
  location.href = 'protectedpage.html';
```

- Better is to use password as name of page

```
location.href = this.elements[0].value + '.html';
```

  - page can be accessed when name is known
  - content is transmitted plain text

- Better is to send encrypted page and use password locally to decrypt

- Preferable is server-side authentication with SSL

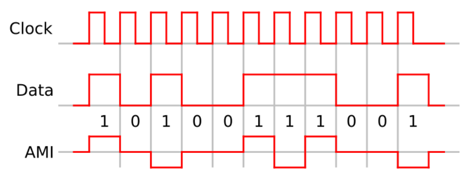
Developing JavaScript

- Perform encryption before sending data to server
  - impossible because key must be shared between user and server
  - thus, key must be contained in script
  - can be snooped by attacker
- Performing client-side checks
  - can be useful to improve performance
  - but server cannot rely on *any* client-side security validation
- Nice mechanisms to hide email addresses
  - assemble email address on load
  - crawler parses only text
- Cross-site scripting problematic (we will look at this next time)

AJAX

- In the last five years, the web has become increasingly interactive
  - Asynchronous Javascript and XML
  - Allows the development of highly interactive websites (e.g., Gmail, Facebook)
  - New problems are emerging: Now, there is a lot of code running on the client
  - How to secure these applications that split functionality between server and client?

URL Encoding



- Values in URLs can be URL-encoded
  - must be decoded properly
- Hex encoding (RFC compliant)
  - %XX, where XX is hexadecimal ASCII value of character
    A = %41
- Double hex encoding (Microsoft IIS)
  - %25XX, where XX is hexadecimal ASCII value of character (%25 = %)
    A = %25XX
- Double nibble hex encoding (Microsoft IIS)
  - each hexadecimal nibble is separately encoded
    A = %25%34%31

HTML Filtering

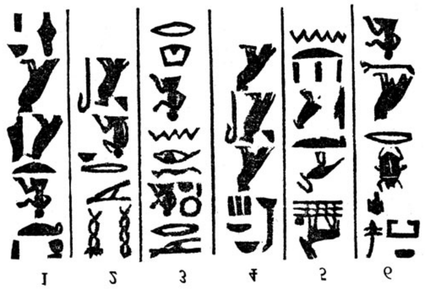
- When HTML data must be accepted
  - use validation of HTML data
  - list of “safe” HTML tags
  - nesting must be balanced
  - check attributes (some may contain scripts)

- Validating links (URIs/URLs)

```
URI = scheme://authority[path][?query][#fragment]
authority = [username[:password]@]host[:portnumber]
```

  - most other options should be immediately removed (user / passwd)

Character Encoding



- Strings are represented as characters
- Traditionally, 8-bit ASCII characters were used
  - only 256 characters possible
  - unsuitable for many languages except English
- ISO 10646 Universal Multiple-Octet Coded Character Set (UCS)
  - unique 31 bit values for each character
  - first 65536 characters termed16-bit BSM (basic multi-lingual plane)
  - merged with Unicode forum efforts
- Problem with existing programs that expect a character to be a byte

➢ UTF-8 encoding

Character Encoding

- UTF-8 encoding
  - variable length encoding (character is 1 to 7 bytes long)
  - classical US ASCII characters (0 to 0x7f) encode as themselves
  - Characters beyond 0x7f are encoded as a multi-byte sequence consisting only of bytes in the range 0x80 to 0xfd
    - especially, no null character permitted

- Problem
  - same value can be encoded in different ways
  - standard now requires "smallest possible form”
  - thus some sequences are not permitted
  - opens problems for misinterpretation
    00 could be encoded (illegally) as C0 80

Cross-site scripting (XSS)

- Simple attack, but difficult to prevent and can cause much damage
- An attacker can use cross site scripting to send malicious script to an unsuspecting victim
  - The end user's browser has no way to know that the script should not be trusted, and will execute the script.
  - Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by your browser and used with that site.
- These scripts can even completely rewrite the content of an HTML page!

Cross-site scripting (XSS)

- XSS attacks can generally be categorized into two classes: stored and reflected
  - Stored attacks are those where the injected code is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, etc.
  - Reflected attacks are those where the injected code is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request.

XSS Delivery Mechanisms

- Stored attacks require the victim to browse a Web site
  - Reading an entry in a forum is enough…
  - Examples of stored XSS attacks: Yahoo (last year), e-Bay (this year)
- Reflected attacks are delivered to victims via another route, such as in an e-mail message, or on some other web server
  - When a user is tricked into clicking on a malicious link or submitting a specially crafted form, the injected code travels to the vulnerable web server, which reflects the attack back to the user’s browser. Example: Squirrelmail

Cross-site scripting (XSS)

- The likelihood that a site contains potential XSS vulnerabilities is extremely high
  - There are a wide variety of ways to trick web applications into relaying malicious scripts
  - Developers that attempt to filter out the malicious parts of these requests are very likely to overlook possible attacks or encodings
- How to protect yourself?
  - Ensure that your application performs validation of all headers, cookies, query strings, form fields, and hidden fields (i.e., all parameters) against a rigorous specification of what should be allowed.
- OWASP Filters project

XSS Delivery Mechanisms

- Stored attacks require the victim to browse a Web site
  - Reading an entry in a forum is enough…
  - Examples of stored XSS attacks: Yahoo (last year), e-Bay (this year)
- Reflected attacks are delivered to victims via another route, such as in an e-mail message, or on some other web server
  - When a user is tricked into clicking on a malicious link or submitting a specially crafted form, the injected code travels to the vulnerable web server, which reflects the attack back to the user’s browser. Example: Squirrelmail

Cross-site scripting (XSS)

- The likelihood that a site contains potential XSS vulnerabilities is extremely high
  - There are a wide variety of ways to trick web applications into relaying malicious scripts
  - Developers that attempt to filter out the malicious parts of these requests are very likely to overlook possible attacks or encodings
- How to protect yourself?
  - Ensure that your application performs validation of all headers, cookies, query strings, form fields, and hidden fields (i.e., all parameters) against a rigorous specification of what should be allowed.
- OWASP Filters project

Simple XSS Example

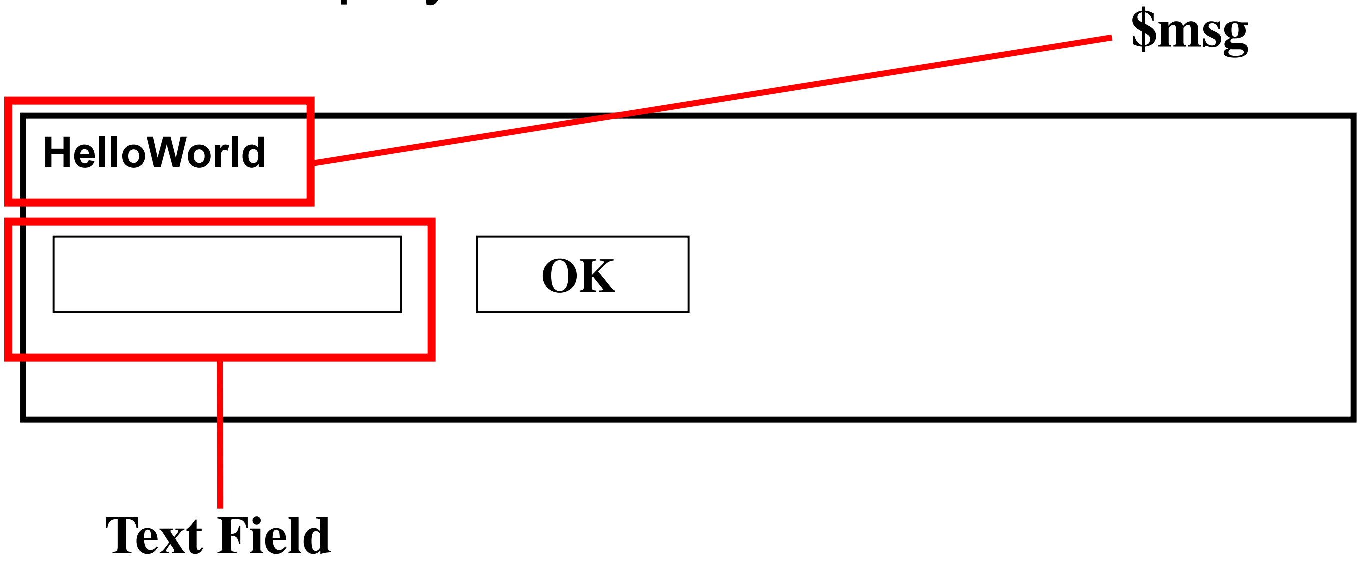
- Suppose a Web application (*text.pl*) accepts a parameter *msg* and displays its contents in a form:

```
$query = new CGI;
$directory = $query->param(“msg”);
print “
<html><body>
<form action=“displaytext.pl” method=“get”>
$msg <br>
<input type=“text” name=“txt”>
<input type=“submit” value=“OK”>
</form></body></html>“;
```

Unvalidated input!

Simple XSS Example

- If the script *text.pl* is invoked, as
  - *text.pl?msg=HelloWorld*
- This is displayed in the browser:


$msg
HelloWorld
OK
Text Field