Preventing HTML form tampering

http://advosys.ca/tips/form−tampering.html
Aug 07 2001

Introduction

All web applications rely on HTML forms to receive input from the user.

However, HTML forms have one large weakness: users can save the form to a file, edit it, then use the edited version to submit data back to the server.

This security problem is made worse by the "stateless" nature of web apps. HTTP transactions are connectionless, one−time transmissions. To lead a user through a series of input forms requires storing information about where they've been before.

Most developers choose to store this "state" information in the user's browser, and have it sent back with each transaction. State information can be stored in a browser three ways:

- Browser "cookies"
- Special tags in the URL
- HTML form "hidden" fields

There are advantages and disadvantages to each method of preserving state information. Arguably the most common method is to use HTML "hidden" fields... fields embedded in the HTML of an input form the browser doesn't display to the end user.

"Hidden" fields are the easiest to use and can hold a lot of information. However, because web browsers let users save HTML forms to their computer and re−submit them, the contents of "hidden" forms are easy to tamper with.

This paper describes the problem in detail and provides example code to secure HTML forms.

- In Part One we demonstrate how easy it is to tamper with hidden field values and show a way to detect tampering.
- In Part Two we expand on the technique to show a more secure way to save state information.

The example web applications are written in Perl using the popular CGI.pm library by Lincoln Stein (http://stein.cshl.org/WWW/software/CGI/cgi_docs.html). However, the concepts presented apply to any language used for web development... Java, C, PHP, Visual Basic etc.

Part One: Hidden field vulnerabilities

Hidden fields

"Hidden" fields are an easy choice for preserving state information in a web application. As the user proceeds through each input screen, you can use "hidden" fields to store the information already collected so it is sent back to the server as part of the next transaction.

Example app: log in and change address

Here's a simple web application that allows a user update their mailing address.

The app displays two pages:

1. A form for mailing address information
2. A page indicating the result of the transaction

```
#!/usr/bin/perl −T
#
# unsafe−form.pl
#
# CGI.pm Perl script demonstrating how easy it is to
# tamper with "hidden" form field values.
#
# Written by Advosys Consulting Inc., Ottawa
#
# Requires: Perl 5 with CGI.pm
#

# Include perl modules:
use CGI qw/:standard/;

# Print the MIME header before doing anything else:
print "Content−type: text/html\n\n";
print '<html><body>';
print '<h1>Unsafe input form demo</h1>';

# Assign some example values we don't want changed:
$userid = 'ktrout';
$credit_ok = 1;
$form_expires = '20001001:12:45:20';

# Display blank HTML form or check submission:
if ( ! param('chaddr') ) {
        print_form();
}
else {
        print "Thank you ", param('userid'), "<br>";
        print "Your address information has been updated.";
}

print "</body></html>";


### SUBROUTINES:
#

sub print_form {
```

```
# Prints example HTML form with signature in a hidden field:
print<<END_TEXT;
<p><form action="http://advosys.ca/cgi/unsafe-form.pl" method="post">
<table>
<tr>
<td><b>Address line 1:</b></td><td><input type="text" name="address1"></td>
<tr>
<td><b>Address line 2:</b></td><td><input type="text" name="address2"></td>
<tr>
<td><b>City:</b></td><td><input type="text" name="city"></td>
<tr>
<td><b>Prov:</b></td><td><input type="text" name="prov"></td>
<tr>
<td><b>Postal:</b></td><td><input type="text" name="postal"></td>
<tr><td colspan="2" align="center">
<input type="hidden" name="userid" value="$userid">
<input type="hidden" name="credit_ok" value="$credit_ok">
<input type="hidden" name="form_expires" value="$form_expires">
<input type="submit" name="chaddr" value="Change address">
</td></tr>
</table>
</form>
END_TEXT
}
```

(We're cheating a bit here and skipping a login page for the app and the script doesn't actually do anything with the form data, but you get the idea).

Run the above script in your browser and view the HTML source code. The form is using "hidden" fields to store information from the login process:

```
<input type="hidden" name="userid" value="ktrout">
<input type="hidden" name="credit_ok" value="1">
<input type="hidden" name="form_expires" value="20001001:12:45:20">
```

When you fill in some information and press the submit button, a confirmation screen uses the value of the userID hidden field, directly from the HTML form.

Tampering with the form

Use your browser's "Save as" feature to save the HTML of the change address form to your computer. The complete HTML, including values in the "hidden" fields are saved.

Use a text editor to change the userid field and save the file. You can alter the other "hidden" fields as well.

Open the file into your web browser and submit the form. Because the application trusts the contents of the "hidden" fields, it lets you change the address for and user ID you like.

What about HTTP_REFERER?

Experienced web programmers may be thinking this type of tampering can be prevented by checking the HTTP_REFERER variable.

Most browsers send an HTTP header named "HTTP_REFERER" (yes, it's really spelled that way). It contains the URL of the page the user viewed before the current one (in the CGI.pm module this header is

available from the referer() function).

For self−referring web applications such as our demo app, HTTP_REFER will contain the URL of the application itself. If the user saves the form to their computer and resubmits it, HTTP_REFERER is blank or contains a different URL.

This is *not* a safe method to validate a form against tampering. The contents of HTTP_REFERER are sent by the web browser. A user with only a little knowledge can write a script (in Perl or other language) to alter this header along with the contents of the form.

Checking HTTP_REFERER will catch trivial attempts to tamper with forms, but cannot be relied on for serious web applications.

A general rule when developing web applications is that *anything* sent back by a web browser: form fields, HTTP headers, and even cookies can all be tampered with and must be considered untrustworthy information.

Using digest algorithms

If all input from a web browser can be altered and is untrustworthy, how can a web app detect tampering?

One way is to use secure hash algorithms, called "Message Digest" algorithms. Digests are used in SSL browser connections, Virtual Private Networks (VPNs) and Public Key Infrastructure (PKI) systems to "sign" data. The same technique can be used in a web app to "sign" hidden fields.

Message digests are simular to checksums. They take a string of characters of any length and determine a "fingerprint" of the characters. Message digest algorithms are supposed to make it impossible to alter any part of the data without also altering the fingerprint.

The most common algorithm is RSA Data Security's (http://www.rsasecurity.com/) Message Digest 5 (MD5). Many other algoritms exist such as SHA1.

Digests and HTML forms

Applied to HTML forms, you can use MD5 or other algorithm to create a fingerprint of hidden field data. Simply concatenate the values of all the hidden fields into a string, compute a fingerprint and send it out in another hidden field. When the form is submitted by the user, the hidden fields returned can be fingerprinted again and compared to the original fingerprint.

But wait! If the user knows you're using MD5, it's possible for them to alter the hidden fields and generate a new fingerprint from the form data.

Fortunately, Digest algorithms are designed to make it impossible to determine the contents of a message from it's fingerprint. We can use this to also add a secret component to the fingerprint the user never sees. The user may be able to compute the fingerprint of the "hidden" form fields, but without also knowing the secret component, they cannot compute the correct fingerprint.

Example app: a tamper−proof form

Here's our simple "change address" application again but this time we add an MD5 "fingerprint" to the form.

```
#!/usr/bin/perl −T
#
# safe−form.pl
#
# CGI.pm Perl script demonstrating use of digest algorithms
# to prevent user tampering of "hidden" form field values.
#
# Written by Advosys Consulting Inc., Ottawa
#
# Requires: Perl 5 with CGI.pm and Digest::MD5 modules
#

# Include perl modules:
use Digest::MD5  qw(md5_base64);
use CGI qw/:standard/;

# Pick a secret password used to sign form variables:
$secretkey = 'tOpSeCret49jqr';

# Print the MIME header before doing anything else:
print "Content−type: text/html\n\n";
print '<html><body>';
print '<h1>Safe input form demo</h1>';

# Assign some example values we don't want changed:
$userid = 'ktrout';
$credit_ok = 1;
$form_expires = '20001001:12:45:20';


# Display blank HTML form or check submission:
if ( ! param('chaddr') ) {
        # Create an MD5 signature from the values:
        $signature = sigMD5( 'create', $secretkey, '$userid', '$credit_ok', '$form_expires');
        print_form();
}
else {
        # Validate signature
        if ( sigMD5( 'check', $secretkey, 'userid', 'credit_ok', 'form_expires' )  eq param('sig
        print "Thank you ", param('userid'), "<br>";
        print "Your address information has been updated.";
        } else {
                print "ERROR: 'Hidden' fields were tampered with!";
        }
}

print "</body></html>";


### SUBROUTINES:
#

sub print_form {
# Prints example HTML form with signature in a hidden field:
print<<END_TEXT;
<p><form action="http://advosys.ca/cgi/safe−form.pl" method="post">
<table>
<tr>
<td><b>Address line 1:</b></td><td><input type="text" name="address1"></td>
<tr>
<td><b>Address line 2:</b></td><td><input type="text" name="address2"></td>
<tr>
<td><b>City:</b></td><td><input type="text" name="city"></td>
```

```
<tr>
<td><b>Prov:</b></td><td><input type="text" name="prov"></td>
<tr>
<td><b>Postal:</b></td><td><input type="text" name="postal"></td>
<tr><td colspan="2" align="center">
<input type="hidden" name="userid" value="$userid">
<input type="hidden" name="credit_ok" value="$credit_ok">
<input type="hidden" name="form_expires" value="$form_expires">
<input type="hidden" name="signature" value="$signature">
<input type="submit" name="chaddr" value="Change address">
</td></tr>
</table>
</form>
END_TEXT
}

sub sigMD5 {
# Generates an MD5 hash from a secret key and either a list of perl variable
# names or a list of CGI.pm field names.
#
# Usage:
#   sigMD5( 'create', $secretkey, '$perlvar1' [, '$perlvar2' ] ... );
# or
#   sigMD5( 'check', $secretkey,  'ParamName1', [, 'ParamName2' ] ... );
#
        my $mode = shift;
        my $key = shift;
        my @names = @_;
        my $values = '';
        my $fieldname;

        # Join each variable name with it's value:
        foreach $fieldname (@names) {
                if ($mode eq 'create') {
                        $values .= $fieldname . eval $fieldname;
                } else {
                        $values .= '$' . $fieldname . param($fieldname);
                }
        }

        $values = $key . $values;
        return md5_base64($values);
}
```

Run the above script in your browser and view the HTML source code. The form has a new "hidden" field named "signature" to store an MD5 fingerprint::

```
<input type="hidden" name="userid" value="ktrout">
<input type="hidden" name="credit_ok" value="1">
<input type="hidden" name="form_expires" value="20001001:12:45:20">
<input type="hidden" name="signature" value="OZ+1iYhIPiDw5hJdtjywQA">
```

The fingerprint was generated using the names and values of the three hidden fields, plus a secret key stored only on the server.

When you submit the form, the contents of the three hidden fields are combined with the secret key and a MD5 fingerprint is generated. If it doesn't match the "signature" field sent with the original form, the fields have been tampered with.

Try saving the form to your computer and editing the "userid" field again. This time the app is able to detect the change.

The solution to tampering?

This method detects any change to fields you specify, but is not foolproof.

Secret keys

The biggest weakness is this method relies on a "secret key" stored on the web server. If someone breaks into your server or view the source code of the app, the method is no longer secure.

This is a weakness of all "secret key" encryption methods and difficult to solve. However, assuming your web server is reasonably secure from break−ins and you change the key regularly, it may be secure enough. Certainly this method is an huge improvement on assuming users won't tamper with "hidden" fields or that checking HTTP_REFERER is safe.

Better than hidden

Detecting "hidden" field tampering adds security to web applications, but for critical apps like e−commerce apps, hidden fields are a danger.

For example, storing a user's credit card number in a hidden field is not secure. hidden fields are transmitted across the Internet in plain text, making it possible for outsiders to "sniff" the number as it is transmitted from server to browser and back.

Use an encrypted SSL connection reduces the chance of sniffing, but it still allows anyone to use the browser's "View source" feature to see the contents of hidden fields.

Imagine an authorized user leaving their browser unguarded half−way though placing an order. Anyone with access to their computer could use "view source" to see the credit card information and other data stored in "hidden" fields.

To really add security, vital information shouldn't be stored in "hidden" fields at all. A better approach is to keep everything on the server and send only a session ID to the browser.

This technique will be described in Part Two of this paper.

Comments, suggestions, criticisms, additions to this document?
Please e−mail tips@advosys.ca

Latest version of this document available at http://advosys.ca/tips/form−tampering.html

Last modified Aug 07 2001

Copyright and terms of use

Use of this document

Permission to use this document from the Advosys Consulting web site is granted, provided that (1) This notice notice appears in all copies, (2) use is for informational and non−commercial or personal use only and will not be copied, reprinted, or posted on any network, computer or broadcast in any media, and (3) no modifications of the document are made.

Educational institutions (specifically K−12, universities and community colleges) may reproduce the Documents for distribution in the classroom, provided that (1) the below copyright notice appears on all copies, and (2) the original Uniform Resource Locator ("URL") of the document on the Advosys Consulting web site appears on all copies.

Use of this document for any other purpose requires written permission of Advosys Consulting Inc.

Limitation of liability

Advosys Consulting take no responsibility for the accuracy or validity of any claims or statements contained in the Documents and related graphics ("the content") on the Advosys web site. Further, Advosys Consulting Inc. makes no representations about the suitability of any of the information contained in the content for any purpose. All such documents, related graphics, products and services are provided "as is" and without warranties or conditions of any kind. In no event shall Advosys Consulting Inc. be liable for any damages whatsoever, including special, indirect or consequential damages, arising out of or in connection with the use or performance of information, products or services available on or through the Advosys Site.

Trademarks

Product, brand and company names and logos used on the Advosys web site site are the property of their respective owners.

About Advosys Consulting

Advosys Consulting is a privately held systems management corporation. Our global headquarters is in Ottawa, Ontario Canada. Formerly known as "Webber Technical Services", we have been providing systems management, computer engineering, security and consulting to private sector and government clients since 1991.

Areas of expertise

Advosys is a diversified consulting firm providing services in many areas of Information technology:

- Internet technologies
- Firewalls and information security
- Web applications
- Network architecture
- Unix and Linux systems management

Unbiased recommendations

Advosys Consulting is an *independent* consulting firm. We have broad experience with multiple vendors including Sun, Hewlett−Packard and Microsoft but are not a reseller of their hardware or software.

Unlike many consulting firms, Advosys receives no commissions, percentages, or other rewards from companies for promoting particular products or services.

This allows us the freedom to offer uncompromised objectivity. We have knowledge and experience with a broad range of products and technologies and can recommend solutions from any manufacturer, or open−source freeware if that is what best fits the requirements. Advosys works only for you, our client, not for a product manufacturer.

For more information, please visit us at http://advosys.ca