

Hidden Risks in Web Code

by Rich Brauchle

A LOOK AT THE HTML SOURCE CODE behind Web sites can often reveal security issues that would never be uncovered by those blissfully ignorant of the code. This bug report will examine two common methods of maintaining state and passing data in Web-based systems—hidden form fields and the HTTP GET method—and demonstrate some of the associated security risks through an examination of HTML code.

Security Risks in Hidden Form Fields

An HTML form allows a Web site user to provide input to the Web server/system via the ubiquitous text field, list box, radio list, check box, and other elements. Many e-commerce systems require the user to fill out multiple forms, one after another, to complete a purchase. Typically, this is done for usability and page-load time concerns. Placing all of the input fields on one page makes the form daunting. Additionally, the HTML code for that single form is then quite large, resulting in long page-load times for those impatient Web users.

Since HTTP connections are stateless, something must be done to maintain state—keep track of user information or order information, for example—among the multiple forms. Cookies are one option. Another option is hidden form fields—hidden text fields that contain state-related data. These hidden text fields are *not* displayed in the browser's GUI, but the information they contain is sent to the Web server when the form is submitted. (The HTTP POST method is what accomplishes this.)

Let's consider a real-world example to clarify the hidden form field concept. I

QUICK LOOK

- Using white box testing to find security flaws
- Examining hidden form fields and the HTTP GET method

recently tested a Web-based system that sold individual songs you could purchase for a small fee and download to your PC. (The exact nature of the system was somewhat different, but I need to disguise it for client confidentiality purposes.) A sequence of three forms had to be completed in order to make a purchase.

The first form contained a Java search applet that allowed you to browse the available songs by category—jazz, rock, classical, etc. Upon finding the song you wanted, you would select it and click the Purchase button to proceed to the second form. The second form displayed the details of the song you chose and required you to enter your credit card and contact information. The third and final form allowed you to review your selection and the credit card/contact information for correctness before committing to the purchase.

While testing the system, I selected the song "In the Mood" on the first form

selectedSong, kept track of the song selection I made on the previous form. The second hidden field, **sessionID**, was a unique ID corresponding to my shopping session on the site. The third hidden field, **itemCode**, appeared to be the numeric code of the "In the Mood" tune. The fourth hidden field, **itemCost**, appeared to be the purchase price of this song.

Hmmm. It's okay to send the cost to the browser as HTML text, to display the item price to the user. However, that wasn't happening here. Since the price was being sent in a hidden form field as part of the second form, that price would be sent back to the server when I submitted the form. So, what happens if I edit this hidden price field? Will the site actually use my altered price (hopefully not), or will it use the **itemCode** value to look up the price of the song in the server-side database and ignore my altered price (hopefully)?

```
<input type="hidden" name="selectedSong" value="In the Mood">
<input type="hidden" name="sessionID" value="2222222222222265VX">
<input type="hidden" name="itemCode" value="850574">
<input type="hidden" name="itemCost" value="2.99">
```

Figure 1: Hidden form fields maintain state

and proceeded to the second form. I had my browser's cookie option set to prompt me, but was not prompted to accept or reject a cookie. Since this signaled the site wasn't using cookies, I guessed that hidden form fields were being used to "remember" the song I had selected. To test this hypothesis, I brought up the HTML source code for the second form using the View Source in my browser. I was using IE, so the code came up in the Windows Notepad. I used the Find function in Notepad to search for the word "hidden" and found the block of code in Figure 1.

The first hidden form field, named

To find out, I saved the second form to my PC's hard drive using IE's File | Save As function and loaded this page into Notepad. I changed the price from \$2.99 to \$0.01, loaded the edited page into my browser, and attempted to complete the purchase. Lo and behold, the server took my edited price, and I was able to purchase the song for one penny. The next thing to try would be making the price negative, perhaps -\$10, and seeing if my credit card was refunded \$10. I could also try changing the other hidden form fields and changing combinations of hidden form fields.

I made several "reduced" purchases

in this fashion and then had the site developers check the backend database and credit card charge log. They confirmed that the purchases with edited prices were going through without any problem. To resolve this issue, the developers reworked the form code to avoid sending pricing information in hidden fields. Instead, all pricing information was read directly from the server-side database that was unavailable to the general public.

Security Risks in the HTTP GET Method

I was testing the system for a technology council that my company belongs to. This council had an online events calendar that displayed all upcoming council events in date order. Clicking on an event in the calendar brought up a detail page for that event with a verbose event description, intended audience, location, directions, and price information. This detail page also had a Register Now! button for each event that took you to a registration and online payment form. The form looked like this:



Naturally, I wondered how the event details were being passed from the event detail page to the registration form. I moved my cursor over the Register Now! button and noticed that its link code (displayed in the bottom status bar of my browser window) was quite long. I right-clicked the link, chose the Copy Shortcut menu item to copy the link code to the clipboard, and pasted it into Notepad for a closer look. The link code was:

```
https://somedomain.com/events/signup.cgi?ename=How+Secure+Are+You?&membcost=20&noncost=40&stucost=0&evtdate=06/19/2001&evtime=8:30+am&day=19&month=06&year=2001&evtdbid=304
```

Clicking this link took me to the following page:



FIELD NAME	VALUE	MEANING
Ename	How+Secure+Are+You?	Event name
Membcost	20	Cost to attend for council members
Noncost	40	Cost to attend for non-members
Stucost	0	Cost to attend for students
Evtdate	6/19/2001	Date of event
Evtime	8:30+am	Time event starts
Day	19	Day of month of event
Month	06	Month of event
Year	2001	Year of event
Evtdbid	304	Event id in server-side database

Figure 2: Information encoded in a URL

The HTTP GET method was being used here to pass all kinds of data to a server-side script, `signup.cgi`, that processes the registration. The GET method encodes data in the URL and separates the individual data fields being passed with the ampersand character, “&”. Contrast this with the HTTP POST method used to transmit the hidden form field data discussed previously. GET transmits the data in the URL, but POST sends it separately.

Let’s dissect the link above and see if we can decipher what the data fields are (see Figure 2).

(If you read the previous section on hidden form fields, you’ll know what’s coming next.)

I took the original link code, edited some of the data it contained, and then used the edited link to access the registration form. Specifically, I changed the event name, member cost, and non-member cost. The edited link (with edited values in boldface) and resulting registration form are shown below.

```
https://somedomain.com/events/signup.cgi?ename=Not+Too+Secure!&membcost=1&noncost=2&stucost=0&evtdate=06/19/2001&evtime=8:30+am&day=19&month=06&year=2001&evtdbid=304
```



This registration form, accessed with the edited link, allowed me to register for

the event at a \$1 price, instead of the proper \$20.

A question similar to that asked in the hidden form field situation can be asked here: Why did the Web site encode vital and sensitive pricing information in the link? A better design would have the link transmit *only* the event ID, like `https://somedomain.com/events/signup.cgi?evtdbid=304`.

Then, the server-side script could access the server-side database to pull the event name, date, time, and cost information. Why send all of this information as part of the link and then have it sent right back to the server for processing? Why expose the price and other information to user-hacking attempts?

Another interesting test would include deleting fields from the link and replacing numeric data (like the event ID) with alphabetic and/or symbol data. The Web site should trap this incomplete or invalid input and present a plain-English error message to the user. It should not crash, malfunction, or corrupt data in the database.

Delving into HTML source code and searching for hidden form fields and the use of the HTTP GET method can reveal security issues that would not be found otherwise. Editing the hidden form field used for the song price demonstrated that an end user could make the price anything he wished. Without rolling up my sleeves and searching the source code to investigate how state was being maintained between forms, I would have missed the “change song prices” bug.

Stepping back from the GUI-centric

Bug Report

approach and researching how state was being maintained between forms uncovered the use of the HTTP GET method on the online events calendar. As in the song e-commerce site, a quick edit of the code (link code, in this case) demonstrated that an end user could also arrange for a “discount.” *STQE*

Rich Brauchle is Vice President and Co-Founder of Testware Associates, a New Jersey-based software testing consulting services firm. Unless you're sending spam, he can be reached at richb@testwareinc.com.

***STQE* magazine is produced by
STQE Publishing, a division of
Software Quality Engineering.**