



# Website Cookie Testing

Richard Brauchle  
Vice President  
Testware Associates, Inc.

**Testware Associates, Inc.**

242 Old New Brunswick Rd., Suite 240, Piscataway, NJ 08854  
tel: 732-562-0155 email: [info@testwareinc.com](mailto:info@testwareinc.com) Web: [www.testwareinc.com](http://www.testwareinc.com)

**Contents**

<i>Abstract</i>	<i>1</i>
<i>Stateless, Stateful Systems</i>	<i>2</i>
<i>The Stateless HTTP</i>	<i>2</i>
<i>To State or Not To State on the Web</i>	<i>2</i>
<i>Maintaining State with Cookies</i>	<i>3</i>
<i>Per-Session Cookies and Cookie Expiration</i>	<i>3</i>
<i>Cookie Detective Work</i>	<i>3</i>
<i>Cookie Usage by Amazon.com</i>	<i>6</i>
<i>What's Inside a Cookie?</i>	<i>7</i>
<i>Amazon.com Cookie Analysis</i>	<i>7</i>
<i>Cookie Testing</i>	<i>9</i>
<i>Disabling Cookies</i>	<i>9</i>
<i>Selectively Rejecting Cookies</i>	<i>10</i>
<i>Corrupting Cookies</i>	<i>11</i>
<i>Cookie Encryption</i>	<i>12</i>
<i>Conclusion</i>	<i>12</i>
<i>About Testware Associates</i>	<i>12</i>
<i>References</i>	<i>13</i>

**Abstract**

The protocol used for exchanging files on the Web is stateless, but maintaining state is essential for most websites. To maintain state, one option that Web developers have is to use cookies. This paper provides a technical background and real-world examples to help you understand how cookies work and how to test systems that employ cookies. The amazon.com website is used as a real-world example to demonstrate cookie testing techniques.

## Stateless, Stateful Systems

According to whatis.com, a *stateless* system has “no record of previous interactions and each interaction request has to be handled based entirely on information that comes with it.” On the flip side, a *stateful* system does keep record of previous interactions.

To elaborate on stateless systems, we will consider the Hypertext Transfer Protocol (HTTP), which is the protocol used to exchange files on the World Wide Web.

### The Stateless HTTP

If you enter `http://www.testwareinc.com` into your Web browser’s address bar and press Enter, the conversation between your browser and Testware’s Web server over HTTP goes like this:

Your browser: “Hey Testware Web server! Can I please have the page `http://www.testwareinc.com/index.html?`”

Testware Web server: “Yes. The document you requested does exist.”

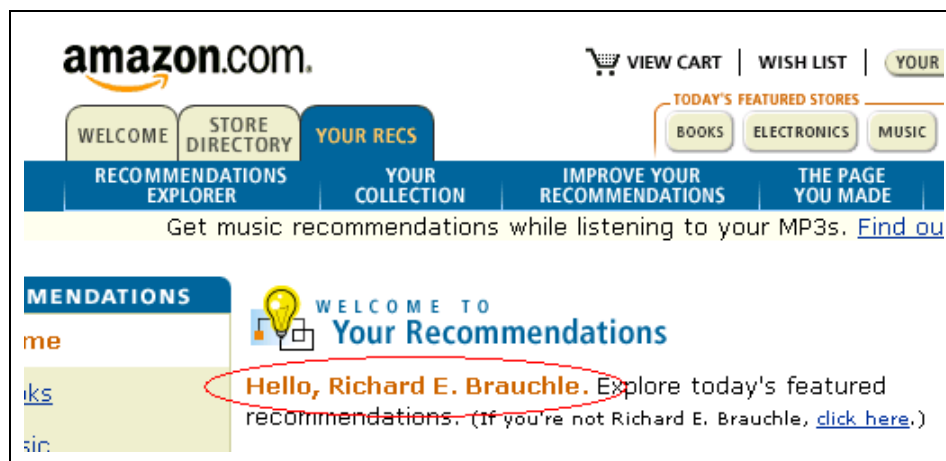
Testware Web server: “Here is the text of the document: <document text follows>.”

Once your browser receives the last byte of the `index.html` page using HTTP, the Testware Web server essentially “forgets” about what you did. If you now go elsewhere on the Testware website, the Testware Web server responds to your new request as above, without memory of your earlier request. This isn’t a bad thing for the Testware website; no harm, no foul. It does not need to know anything about your earlier request to respond to your new request. But are there cases in which state does matter for a Web-based system?

### To State or Not To State on the Web

As we’ll see, state certainly does matter on the Web! Take everyone’s favorite online purveyor of books and music, `amazon.com`. If there weren’t ways to overcome the stateless nature of HTTP and maintain state on the Web, the following would not be possible:

- The nice “Hello, <your name>” message that greets returning shoppers on the Amazon home page. Without state, how could the site have any knowledge of your name?



- The virtual shopping cart. In the absence of state, how could Amazon keep track of the individual items I add to my cart and the quantity of each item?

**Shopping Cart** for Richard E. Brauchle (if you're not Richard E. Brauchle, [click here](#)) See more items like those in your cart

[Continue shopping](#) [in Help](#)

[Proceed to checkout](#) [Sign in](#) to turn on 1-Click ordering.

Shopping Cart Items--To Buy Now	Qty.		
<a href="#">Symphony 2/4</a> L.V. Beethoven; <b>Audio CD</b> Usually ships in 2-3 days <input type="checkbox"/> <b>Add gift-wrap (\$2.99)</b>	<input type="text" value="1"/>	<b>Our Price:</b> <b>\$5.97</b>	<a href="#">save for later</a> <a href="#">delete</a>
<a href="#">Sonata Piano 14/17/23</a> L.V. Beethoven; <b>Audio CD</b> Usually ships in 24 hours <input type="checkbox"/> <b>Add gift-wrap (\$2.99)</b>	<input type="text" value="1"/>	<b>Our Price:</b> <b>\$7.97</b>	<a href="#">save for later</a> <a href="#">delete</a>

If you changed any quantities, please press this [update](#) button to

**Subtotal: \$13.94**

So, how does Amazon work around the stateless HTTP protocol to accomplish the “magic” above? Part of the answer is *cookies*.

### Maintaining State with Cookies

Whatis.com notes that a cookie is “information a Web site puts on your hard disk so that it can remember something about you at a later time.” Why? “Each request for a Web page is independent of all other requests. For this reason, the Web page server has no memory of what pages it has sent to a user previously or anything about your previous visits.”

How and where cookies are stored depends on the browser and operating system you use. Internet Explorer (IE) stores each cookie in a separate file, usually underneath the Windows operating system folder. Netscape (NS) stores all cookies in a single file named cookies.txt, usually in a folder underneath the browser’s installation folder.

### Per-Session Cookies and Cookie Expiration

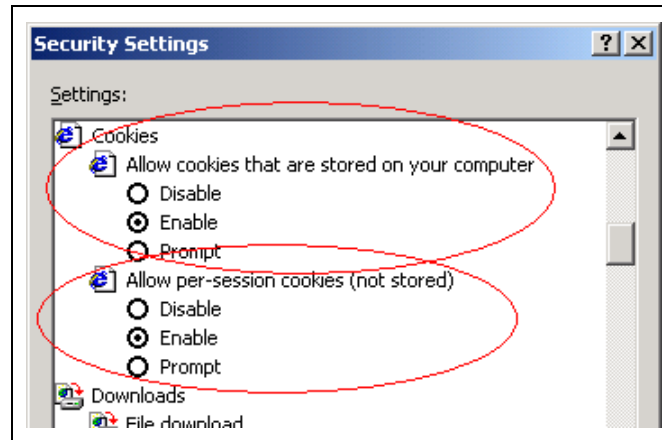
When a Web server sets a cookie on your system, it can optionally give that cookie an expiration date. As time marches on, any cookies with expiration dates in the “past” are deleted.

If the Web server does not give a cookie an expiration date, that cookie is a per-session cookie. Per-session cookies are deleted when you close your Web browser; they only exist for the single Web surfing session beginning when you start the browser and ending when you close the browser.

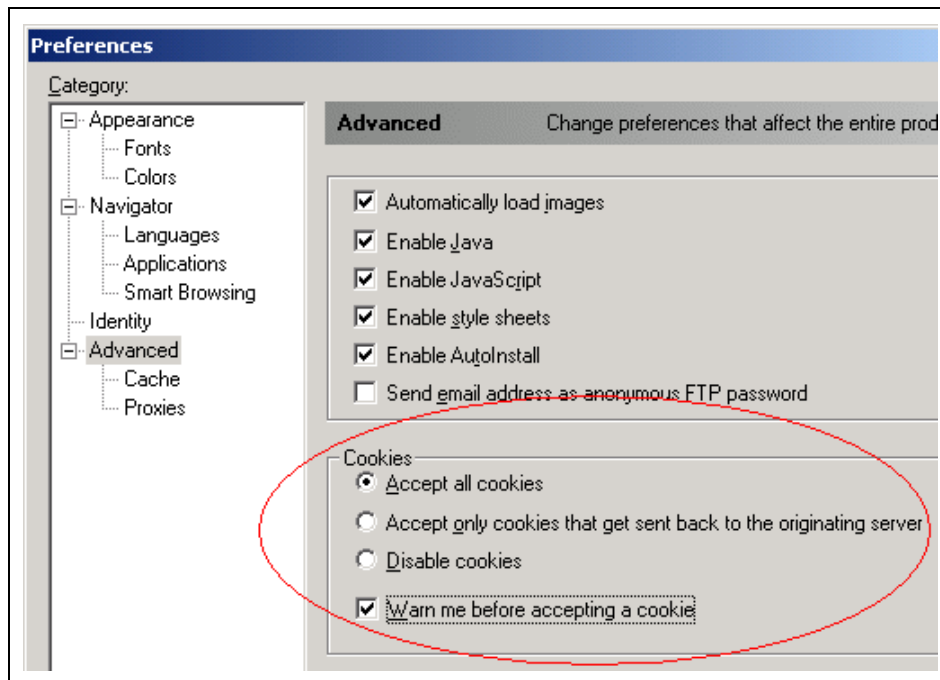
### Cookie Detective Work

How can you tell if the Web system you are testing uses cookies? Simply read the website design documents, functional specs, etc. – if such documents are available. A more direct approach, especially useful in the likely absence of such documentation, is:

- Find the folder on your PC where cookies are stored.
- Delete all of the existing cookies. In Internet Explorer, the cache files are stored in the same folder as the cookies. Clearing the browser cache in IE can make finding the cookies easier, but isn't strictly necessary.
- Set your browser's cookie options to "prompt me" In Internet Explorer, choose Tools | Internet Options, navigate to the Security tab, click Custom Level and select the "Prompt" radio button under "Allow cookies that are stored on your computer". Also do the same under "Allow per-session cookies (not stored)."

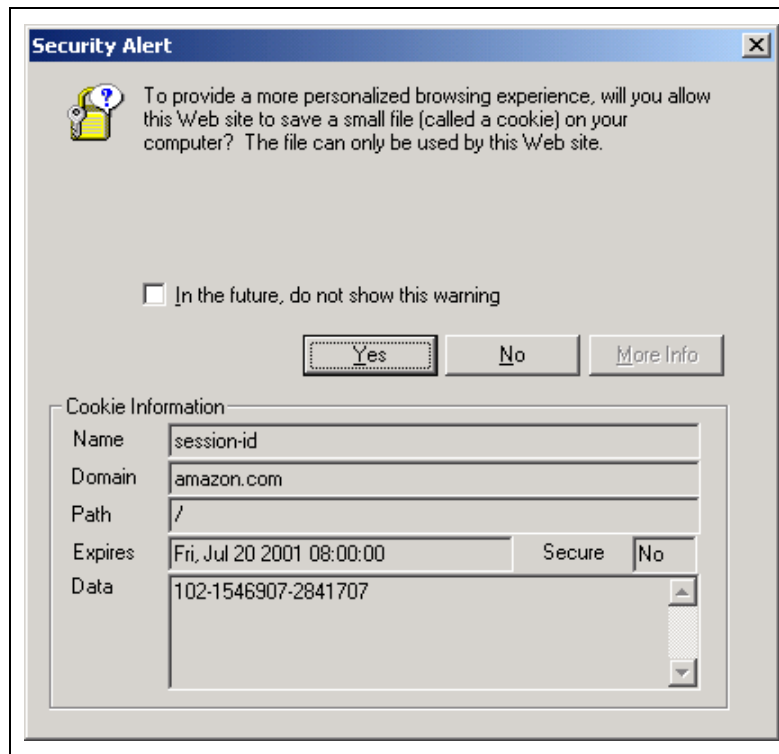


In Netscape, choose Edit | Preferences, select Advanced and check the "Warn me before accepting a cookie" box.



- Navigate through all of the major features and functions on the site to see where cookies are employed.

- How do you know where cookies are used? Whenever the site attempts to record state information in a cookie on your PC, you will be prompted with a message. Internet Explorer's prompt looks like this:



Netscape uses the following prompt:



- Every time this dialog appears, record the cookie details and what action(s) cause the cookie to be created or modified. Then, click Yes to accept the cookie. Personally, I find it easier to accept the cookie, open the cookie file and copy/paste the cookie details into a "cookie log" with my observations for later analysis. Save this data, including the cookie names and contents, creating a log of cookie activity correlated to your activities on the website. A word of warning: some sites are highly active with cookies, setting or modifying them on every page you visit. Creating the cookie log on these types of sites will be time consuming and drive you to a certain level of insanity. Getting as much info as possible in advance about cookie activity from the developers is usually your best bet.

## Cookie Usage by Amazon.com

Let's make the cookie concepts we've discussed so far more concrete by examining how amazon.com uses cookies. In doing so, we will also encounter a common problem in "cookie testing" – figuring out what the hieroglyphic-like information in the cookie means!

To start, I deleted all Netscape cookies from my PC and set the cookie option to prompt me. Next, I navigated to www.amazon.com.

The first cookie activity by the Amazon Web server was to create a cookie (in the cookies.txt file) with the following data.

```
.amazon.com TRUE / FALSE 994320128 session-id 102-7224116-8052958
```

The prompt that Netscape presented me with indicated the cookie will expire on Thursday July 5, 2001, one week from today's date as I write this. (We'll explore the details in the cookie in the next two sections.)

The second cookie set by Amazon contained the following data and also expires on 7/5/2001.

```
.amazon.com TRUE / FALSE 994320181 session-id-time 994320000
```

Amazon's third cookie contained the following and expires on 1/1/2036. My laptop will be reduced to either paperweight or landfill status by then, so this is pretty much a "permanent cookie" relative to the useful life of my laptop.

```
.amazon.com TRUE / FALSE 2082787330 ubid-main 077-4356846-2652328
```

The fourth cookie is a per-session cookie, since the Netscape prompt did not include an expiration date. Since per-session cookies aren't written to the hard drive, examining the cookie content can be done only through the actual Netscape prompt.



The fifth cookie Amazon set expires on 1/1/2036 and contained the following data.

```
.amazon.com TRUE / FALSE 2082787787 x-main hQFiIxHUFj8mCscT@Yb5Z7xsVsOFQjBf
```

After accepting this fifth cookie, the amazon.com home page (finally!) displayed. The URL of the home page was <http://www.amazon.com/exec/obidos/subst/home/home.html/102-7224116-8052958>.

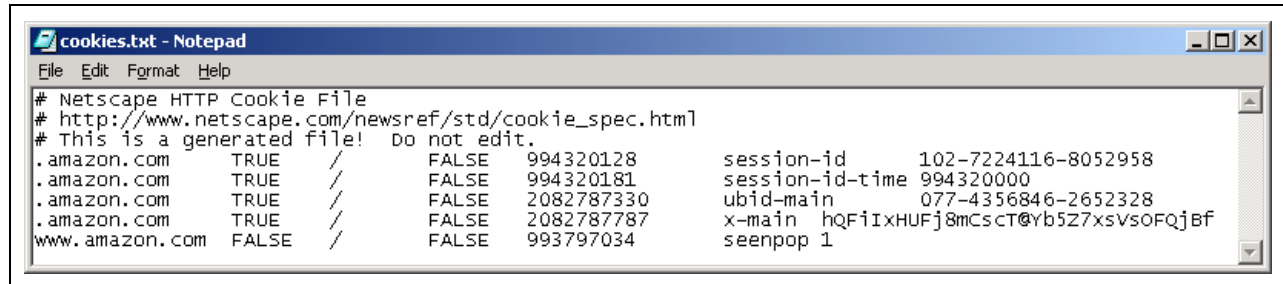
Have we seen that number sequence at the end of the URL before? Yes, it's the session ID stored in the first cookie.

A sixth cookie containing the following data and expiring on 6/29/2001 was then set.

```
www.amazon.com FALSE / FALSE 993797034 seenpop 1
```

Upon accepting this cookie, a secondary browser window popped up with a free shipping promotion notice. A logical guess at this cookie's purpose, then, would be that it tracks whether or not you've seen the promotion popup ad.

After all of these cookies were set, my Netscape cookies.txt file looked like this:



```
cookies.txt - Notepad
File Edit Format Help
# Netscape HTTP Cookie File
# http://www.netscape.com/newsref/std/cookie_spec.html
# This is a generated file! Do not edit.
.amazon.com TRUE / FALSE 994320128 session-id 102-7224116-8052958
.amazon.com TRUE / FALSE 994320181 session-id-time 994320000
.amazon.com TRUE / FALSE 2082787330 ubid-main 077-4356846-2652328
.amazon.com TRUE / FALSE 2082787787 x-main hQFIxHUFj8mCscT@Yb5Z7xsvsoFQjBF
www.amazon.com FALSE / FALSE 993797034 seenpop 1
```

Why are there only five cookies in the file? The per-session cookie is kept in memory only; it is *not* written to the cookies.txt file.

### What's Inside a Cookie?

Before we attempt to analyze all of the cookies set by amazon.com, let's take a quick look at cookie structure and the meaning of cookie data.

The first cookie set by Amazon was:

```
.amazon.com TRUE / FALSE 994320128 session-id 102-7224116-8052958
```

Using the information at [www.cookiecentral.com](http://www.cookiecentral.com), I'll break the cookie down into its individual fields from left to right and describe what each field is used for.

- **.amazon.com** is the **domain** this cookie is valid for. Only cookies set by machines in the amazon.com domain can read this cookie. (However, bugs in Web browser cookie implementation have allowed unauthorized sites to access cookies in the past.)
- **TRUE** is a **flag** indicating whether or not all machines in the domain can access the cookie.
- **/** is the **path** the cookie is valid for.
- **FALSE** is a **secure** flag indicating whether or not a secure (encrypted) connection is needed to access the cookie.
- **994320128** is the UNIX **expiration** time of the cookie. UNIX time is the number of seconds since January 1, 1970 00:00:00 GMT.
- **session-id** is the **name** of the variable stored by this cookie.
- **102-7224116-8052958** is the **value** of this variable.

### Amazon.com Cookie Analysis

Our cookie experiment for amazon.com showed us that simply loading the Amazon home page creates six cookies – one per-session (non-persistent) cookie and five persistent cookies. Since the site design documents and developers are unavailable to us, let's put the cookie data into a table and try to

decipher what the cookies are used for and the meaning of the cookie data. We will consider only the first five cookies, since we determined the purpose of the sixth cookie above.

cookie #	domain	accessible by all machines	path	secure connection needed	expiration	name	value
1	.amazon.com	TRUE	/	FALSE	994320128	session-id	102-7224116-8052958
2	.amazon.com	TRUE	/	FALSE	994320181	session-id-time	994320000
3	.amazon.com	TRUE	/	FALSE	2082787330	ubid-main	077-4356846-2652328
4	.amazon.com	TRUE	/	FALSE	(per-session cookie)	obidos_path	(see above)
5	.amazon.com	TRUE	/	FALSE	2082787787	x-main	hQFiIxHUFj8mCscT@Yb5Z7xsVsOFQjBf

The first cookie is a session ID assigned to my shopping session by the Amazon server. The primary giveaway here is the variable name “session-id”. Another clue is that the data in its value field, 102-7224116-8052958, can be found at the end of the home page URL visible after the 5<sup>th</sup> cookie was set, [www.amazon.com/.../home.html/102-7224116-8052958](http://www.amazon.com/.../home.html/102-7224116-8052958). Cookie 1 expires on 7/5/2001, based on the warning dialog I saw in Netscape before the cookie was set. So, the UNIX expiration time 994320128 in this cookie must correspond to 7/5/2001.

The second cookie’s purpose isn’t obvious. Based on its name and value, session-id-time and 994320000, respectively, I would guess it is the maximum possible “end” time in UNIX time of my amazon.com session. I know from the Netscape warning above that this cookie expires on 7/5/2001, so I can infer that the expiration value of 994320181 in this cookie corresponds to 7/5/2001. Why? These two UNIX times are only  $994320181 - 994320000 = 181 \approx 3$  minutes apart.

The purpose of cookies 3 and 5 is yet even harder to decipher. The names of cookies 3 and 5, `ubid-main` and `x-main`, don’t lend us any immediate understanding. Both of these cookies expire in 2036, so whatever Amazon is tracking here, it must be of long-term use.

The fourth cookie, the only per-session / non-persistent cookie, contains a *long* value with the substring “continue-shopping-url”. I would guess this cookie value tells the Amazon Web server where to send me if I click the “Continue Shopping” button on the shopping cart page. As before, I’m having a hard time figuring out with certainty what this cookie is used for without doing further investigation.

I’d have to talk to the Amazon developers or get access to some design documents or specifications to get any further here.

## Cookie Testing

Now that we're in the know about what cookies are, how they're used to provide state in Web systems and what cookie contents look like, let's address how to test sites that use cookies.

### Disabling Cookies

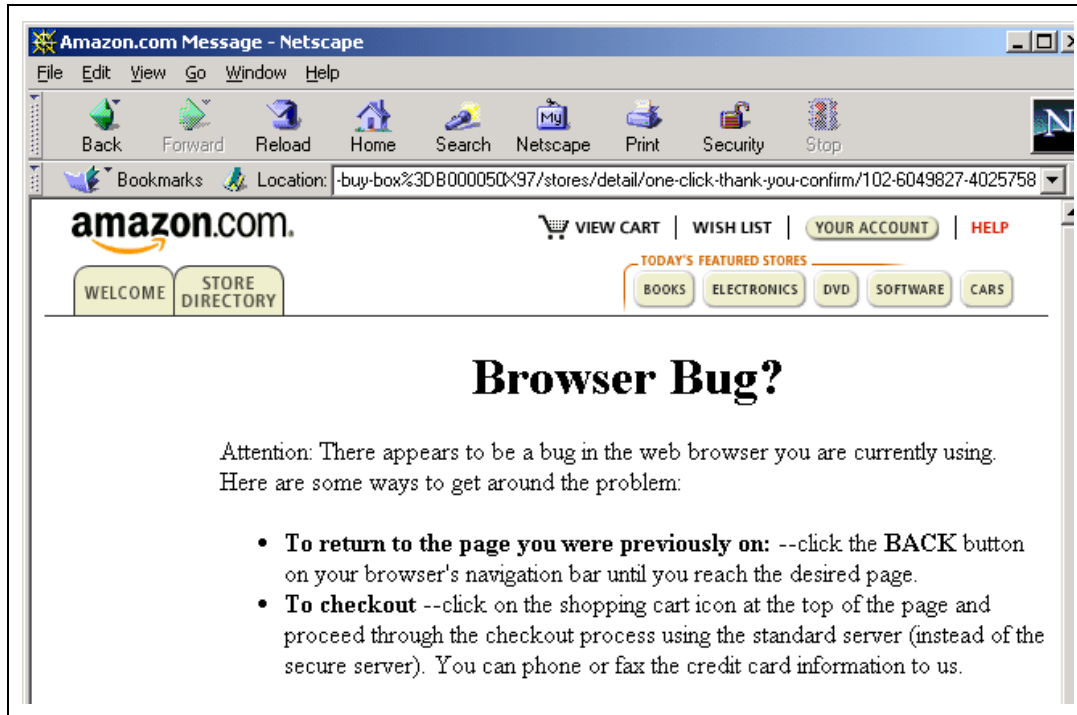
This is probably the easiest area of cookie testing. What happens to the web site if all cookies are disabled? Start by closing all instances of your browser and deleting all cookies from your PC set by the site under test. The cookie file is kept open by the browser while it's running, so you must close the browser to delete the cookies. Closing the browser also removes any per-session cookies in memory.

Disable all cookies and attempt to use the site's major features and functions. Most of the time, you will find that these don't work, since cookies are disabled. This isn't a bug, but rather a fact of life: disabling cookies on a site that requires cookies (of course!) disables the site's functionality.

With cookies disabled, your testing job is somewhat reduced. Is it obvious to the website user that he must have cookies enabled to use the site? Is the Web server recognizing that its attempts to set cookies are failing? If so, does it send a page to the user stating, in plain language, that cookies must be enabled for the site to work? Or, can the user frustratingly attempt the same operation many times in a row without a clue as to why the site isn't working?

Amazon.com passes this test with flying colors. I was able to use all major aspects of the site – searching, shopping cart, checkout functions – even though cookies were completely disabled. I'd bet that state maintenance was being taken care of server-side, based on the session ID at the end of the home page URL. Let's test this hypothesis. The home page URL was `www.amazon.com/.../home.html/104-0274809-0482344`. If I change the rightmost digit from 4 to 5 and repost the URL, Amazon discards my edited URL and "recovers" from the corruption by creating a URL with a new session ID, `www.amazon.com/.../home.html/107-0357560-1728507`. So, it appears that the hypothesis is correct.

Let's probe a little further. I chose the Yamaha CD-ROM kit on the Amazon home page and added it to my shopping cart. The shopping cart page URL was `www.amazon.com/.../one-click-thank-you-confirm/107-0357560-1728507`. Changing the rightmost digit from 7 to 8 and posting this edited URL lost my shopping cart and brought up the following error page, lending further support to the hypothesis of server-side state maintenance with a session ID in the URL.



This server-side state maintenance allows someone to shop at amazon.com even if they have totally disabled cookies – an intelligent design. If cookies are enabled, though, we saw previously that Amazon sets a session ID cookie to “remember” your session ID. Why? If you leave the site with a non-empty shopping cart and then return, the session ID cookie is used to resume your previous shopping session and shopping cart state.

### Selectively Rejecting Cookies

What happens to the site if some cookies are accepted and others are rejected? Start by deleting all cookies from your PC set by the site under test and set your browser's cookie option to prompt you whenever a web site attempts to set a cookie. Exercise the site's major functions. You will be prompted for each and every cookie the site attempts to set. Accept some and reject others. (Analyze site cookie usage in advance and draw up a test plan detailing what cookies to reject/accept for each function.) How does the site hold up under this selective cookie rejection? As above, does the Web server detect that certain cookies are being rejected and respond with an appropriate message? Or, does the site malfunction, crash, corrupt data, or misbehave in other ways?

Let's strategize a selective cookie rejection test for the amazon.com home page. Each test case will require either accepting or rejecting each of the six cookies, so there are  $2^6 = 64$  possible test cases. A subset of the test cases is enumerated in the following table.

test case #	cookie 1 (persistent)	cookie 2 (persistent)	cookie 3 (persistent)	cookie 4 (per session)	cookie 5 (persistent)	cookie 6 (persistent)
1	reject	reject	reject	reject	reject	reject
2	reject	reject	reject	reject	reject	accept
3	reject	reject	reject	reject	accept	reject
4	reject	reject	reject	reject	accept	accept
5	reject	reject	reject	Accept	reject	accept
...						
64	accept	accept	accept	Accept	accept	accept

If I were to run the fifth test case, for example, I would reject the first three cookies when amazon.com tries to set them, but allow Amazon to set the fourth, fifth and sixth cookies.

The first test case is equivalent to the disabling cookies test performed previously, but I'll leave it in the table for completeness. If you think in binary and consider reject to be 0 and accept to be 1, the table has binary representations of the decimal numbers 0 through 63 inclusive.

Based on Amazon's performance in the disabling cookies test, I would guess that the site would pass most or nearly all of the selective cookie rejection test cases. I executed test cases 2 and 5, closing the browser and deleting the cookies before starting each test case. Both passed; I was able to use the site's major functions, as above, without problem. Looks like the site designers ensured that "problems" with cookies would have little or no effect on a customer's ability to shop at amazon.com.

Note that the test cases above only deal with the cookies being rejected or accepted when amazon.com *first* tries to create them. We also should test rejecting and accepting *cookie modifications*. Allow a cookie to initially be set. If/when the Web server attempts to subsequently modify that cookie, what happens if you disallow the change, retaining the "old" value?

### Corrupting Cookies

Now's our chance to really abuse the site under test! You will first need to do the "Cookie Detective Work" mentioned above to determine how and where your site uses cookies and the meaning of the cookie data. Now, exercise the site's major features. Along the way, as cookies are created and modified, try things like:

- Altering the data in the persistent cookies. (Since the per-session cookies are stored only in memory, they aren't readily accessible for editing. There might be tools out there for corrupting per-session cookies, but I'm not aware of any.) Example: in the first cookie written by amazon.com, change the variable name `session-id` to something different, perhaps `ses-id` or `sexqion-id`. Remember, you will have to close the browser to edit the cookies. Before this edit is made, if I visit the Amazon site, close the browser, restart the browser and go back to amazon.com, my "previous" session is maintained based on the session ID in the cookie. However, if I corrupt the session ID variable name, Amazon detects the corruption and recovers by discarding all six of the cookies and recreating them with new values. After editing the cookie, restart the browser and

reload/continue using the site. Did the corrupted cookie cause the site to malfunction? Is any data lost or corrupted in the database?

Second example: change the session-id value in data field by adding 1 to the rightmost digit; 102-7224116-8052958 becomes 102-7224116-8052959. Are you now looking at someone else's shopping session? Anything lost or corrupted in the database?

- Selectively deleting cookies. Allow the cookie to be written (or modified), perform several more actions on the site, and then delete that cookie. Continue using the site. What happens? Is it easy to recover? Any data loss or corruption?

### Cookie Encryption

The last cookie test I'll mention is a simple one. While investigating cookie usage on the site you're testing, pay particular attention to the meaning of the cookie data. Sensitive information like usernames and passwords should NOT be stored in plain text for all the world to read; this data should be encrypted before it is sent to your computer. I've tested many sites where this seemingly obvious rule has been violated. A case can certainly be made that certain types of sensitive data – credit card numbers, for example – should never be stored in cookies, even encrypted.

Based on the amazon.com cookie analysis we performed above, I'd say Amazon easily passes the cookie encryption test. No sensitive user or credit card information is stored in plain text. I did find a way to 'hack' into an account using the cookie data. On User A's machine, I navigated to the Amazon home page and then added a book on John Adams to the shopping cart. I copied all five of the persistent cookies out of the cookies.txt file and pasted them into the cookies.txt file on User B's machine. Navigating to the amazon.com home page on User B's machine gave me access to User A's shopping cart with the John Adams book.

A little further experimentation showed that editing just User B's session ID cookie to match User A's does not allow User B to access A's shopping cart. This leads me to believe that one or more of the other cookies are used as part of a unique key to lessen the probability that this type of hacking would be successful.

### **Conclusion**

State information can be maintained in Web systems by the use of cookies. (Other methods for maintaining state include hidden form fields and embedding state data in HTML links; I recommend that web testers explore these methods as well.) Our job as testers is to find out, by talking to developers, reading system documentation or experimenting with the web site, which of these technologies are being used and to design tests accordingly.

### **About Testware Associates**

Testware Associates, Inc. is a testing services firm that helps clients address their "testing pain" and reduce the risks and costs associated with poor quality software. Testware serves clients by performing the entire end-to-end testing function or by handling selected parts of the overall test effort. Testing work is performed at the client's site, at Testware's in-house testing lab or a combination of the two. It's never too early or too late for Testware to become involved – whether you are building a brand-new, leading edge system or periodically updating a live system, Testware can help! Please visit [www.testwareinc.com](http://www.testwareinc.com) for more information.

**References**

[www.whatis.com](http://www.whatis.com): excellent Web-based technical encyclopedia

[www.cookiecentral.com](http://www.cookiecentral.com): tons o' info about cookies

[www.netscape.com/newsref/std/cookie\\_spec.html](http://www.netscape.com/newsref/std/cookie_spec.html): Persistent Client State HTTP Cookies Preliminary Specification

Stickyminds.com article "Behind Closed Doors – What every tester should know about Web privacy" by Russ Smith, posted 5/22/2001: discussion of HTTP GET and POST methods and cookies

Stickyminds.com article "E-Business Testing: Test Techniques and Tools – Risk-Based E-Business Testing Part 2" by Paul Gerrard, posted 2/7/2001: discussion of HTTP GET and POST methods, cookies, hidden form fields and security testing