Part 11: Handling Authentication and Session Tracking

User Experience, Not Metrics

by:

R. Scott Barber

"The moment you install a Web server at your site, you've opened a window into your local network that the entire Internet can peer through," …

points out one article on Web site security, "The World Wide Web Security FAQ" by Lincoln D. Stein and John N. Stewart. "Most visitors are content to window shop, but a few will try to peek at things you don't intend for public consumption. Others, not content with looking without touching, will attempt to force the window open and crawl in," warns the article. Because security is an issue for every Web site, any performance testing effort needs to take security matters into consideration. This article addresses how (and why) to simulate various forms of secure (and insecure) authentication and session identification using Rational tools.

This is the eleventh article in the "User Experience, Not Metrics" series, which focuses on correlating customer satisfaction with your Web site application's performance as experienced by users. This article begins the final trilogy, which will discuss some advanced topics related to using Rational TestStudio to conduct performance testing. This article is intended for intermediate to advanced Rational TestStudio users. A general knowledge of Web security methods and procedures will help you apply the concepts discussed. Some C programming and an ability to use and manipulate regular expressions is a must.

About Authentication, Cookies, and Session Tracking

Before we can discuss how to handle authentication, cookies, and session tracking in performance test scripts, we first need to understand the basics of what they are, why they're used, and how they work. The sections below are intended to give you enough background so you can understand why it's important to ensure that your performance testing scripts handle these aspects of Web applications correctly.

Authentication

Authentication is most commonly thought of as the process of logging in to a computer, Web site, or application. There are many types of authentication, but the basic idea is that the user selects or is assigned credentials (normally a username and password) that are associated with certain privileges. When the user tries to access the application or site, it

asks for those credentials (except where the authentication is cookie based, as discussed below) and either allows or prohibits access, based on the credentials provided. Authentication is often done through a pop-up window, such as the one shown in Figure 1, but may also be done without a pop-up, directly on the Web page.

Figure 1: Common authentication pop-up window

Authentication is particularly important to performance testing. If your scripts don't do authentication properly, they'll be prohibited from performing the action that you're trying to test and will therefore provide misleading results. Remember, if a site requires you to log in – ever – you need to think about authentication when developing your scripts.

Cookies

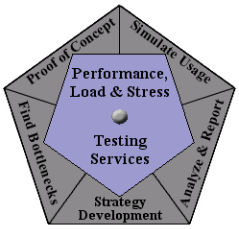
Did you ever notice that when you log on to certain Web sites after your initial visit, you don't have to present any credentials to gain access? That's because the site stored your username and password on your machine in a little chunk of code known as a cookie. Not all Web sites leave cookies on your computer, but many do, especially the large well-known sites. They use cookies to track what you're viewing and to recognize your computer when you come back to visit again.

It's important to consider this during performance testing. If the site you're testing uses cookie-based authentication, you must take care to accurately script new versus return users. Authentication from user submission will likely be more performance intensive than authentication from a previously stored cookie. See "Persistent Client State HTTP Cookies" for a short summary of how cookies work.

Session Tracking

Quite often, Web sites that require authentication also use sessions. These sessions are generally assigned at the time authentication is processed and are tracked in one of a number of ways including hidden fields, client or server side cookies and/or URL parameters. If a site whose performance you're testing tracks sessions, it's imperative that your script handle that session tracking correctly.

For our purposes, a session is the time and activity of a single user accessing a single Web site from a single browser without closing the browser or switching to another Web site. For instance, you go to a



site that sells books and log in if you're a return visitor or create an account if you're a first-time visitor. This begins your session. You navigate the site, find the book you want, add it to your shopping cart, enter your billing information to purchase the book, and finally close your browser or move on to another Web site. This ends your session.

The important point for us is that from the time you logged in until you logged out (or more likely timed out), the bookstore application was keeping track of your activity. Session tracking is critical to the majority of B2B and B2C e-commerce websites. If session tracking isn't handled properly in your performance tests, they'll likely not be performing the activities they were designed to but will simply be forcing unexpected error messages, or worse, associating activities such as orders with the wrong customer.

Handling Authentication in Your Scripts

Since most sessions are tracked based on the results of authentication, we must first ensure that authentication is handled properly in our performance test scripts. The VUc recording and scripting mechanism typically does a very good job of handling authentication automatically. We'll discuss here the proper Rational Robot configurations to ensure that three common types of authentication – basic, secure, and automatic (with cookies) – are handled correctly, and what additional considerations to evaluate.

Basic Authentication

Basic authentication is handled very cleanly in performance (VUc) scripts. Whether you record using the API, network, or proxy method, basic authentication will be detected (assuming, of course, that you're recording against a supported protocol that you have a license for). The option to change recording methods is found in Robot under Tools > Session Record Options on the Method tab, as shown in Figure 3. If you're not familiar with the three recording methods, please refer to Rational documentation.

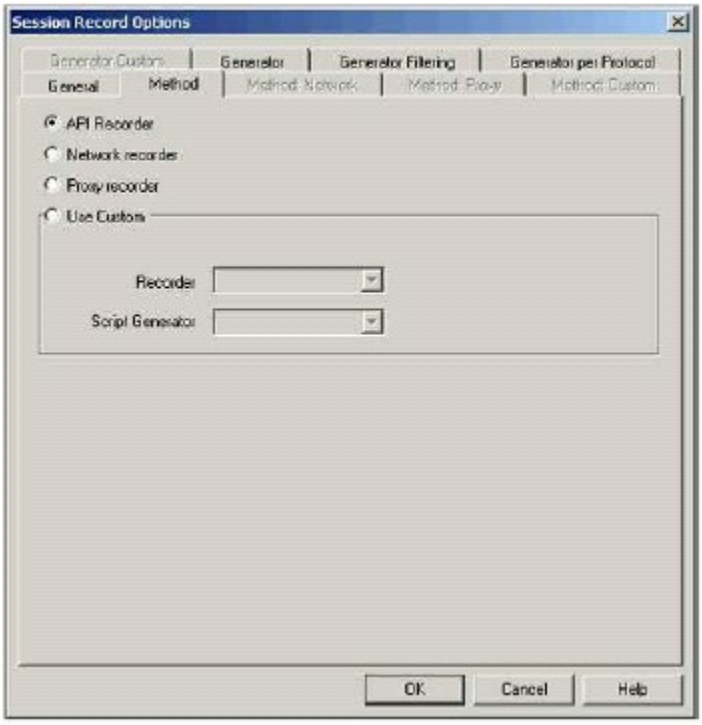


Figure 2: Session Record Options, Method tab

To make sure basic authentication is simulated accurately by your performance test, be sure to do the following:

- Create datapools with enough valid usernames and passwords to keep the same user from logging in multiple times concurrently during your tests – unless, of course, you're intentionally doing security testing or certain types of functional testing and want to see how your application handles this anomaly.
- Ensure that all of the usernames and passwords in the datapool are valid before executing the performance test. I recommend recording a simple script that logs into the home page and then ends. Then configure your suite to execute that script the same number of times as there are values in your datapool and let it run. Theoretically, there should be no errors. If there are errors, check the logs and see which datapool values caused the errors, and then validate them by hand. Then continue recording your scripts and linking them to the validated datapool.

Secure Authentication

Secure (https://) authentication is just a little trickier. Secure authentication is also handled well automatically by VUc scripts, but only when you use the API recording method. The reason for this is that the API recording method captures traffic before the encryption algorithm is initiated to convert the clear text into encrypted form. If you use the network or proxy method to generate scripts against a site that uses secure authentication, the credentials appearing in your script will be encrypted and thus you won't be able to use values from a datapool to simulate different users.

To see the difference, compare the scripts shown in Listings 1 and 2 below. Listing 1 was captured

using the network method. (Note that HTTPS traffic can't be captured via network or proxy methods using the HTTP protocol. If you need to record using network or proxy methods against HTTPS sites, you'll have to select the Socket protocol.) As you can see, all of the information is in hexidecimal format and would be more than slightly difficult to decode. Listing 2 is the exact same script with the same credentials (username and password), recorded using the API method instead.

```
sock_send
"`829eea05ecc336ec80`_4Z:`91eb1087111d`9G`fe5d1be25db9819e`VBK`1f79cf04e2"
"bba731190f6aebffca95c15df979180ee5aa598c1860759b51fa99c826cbc7be65c55a04"
"09ece62183a61ee61617ffffc4b2691f21e0867c1267d855b1c670d9df0aa2e155bf`5/"
"p6`a3f2cf99`ra`8ff2`zg`ef10`Q1`eee84eb90452e67bc6bd39a96660d9995408fbed`9"
"^T{C(`b66fb7eeaf00`S7:\b`a1f7e3e79d9359b9bc0da90f1aa3b6`Se`9deb5c44ac`h"
"9`d9996f94`-)`93`^>7`19877f5ef0b8ce6ebf`,/1\f]`c8`zRs3`f90a`jyD`fb1ac04b"
"d27f23c2b60215`-F`cb6dc9d98eee6d158a499e28f188`U=`af`3G4`d1e1980bc2d3c7fa"
"`</`8d1d0189991b28ce53bc`Va`0ed4e573c76ef9a27cf81ee60475a533bdfbf4`FIX^"
"`1ee5e9`x@`e4280621d147e8`uMid`fdde9ea735e888e9ec10a5d0e24812165c32ac10d4"
"91179ea611f0`@G`1b67f38dc3e1f1830ceb32b217`VGv`02029590e582fa50ada79db823"
"0116ad`l9+`85b38e91` =!6;`fdf4db66d4469d`sS`da6fa7ced5b7`:I`f376faa7e5`p"
"[{`fac63898da2f86bee7d9`0W`60a6b88e`prv`a1` 7`b4b4ed95`N4\t`1574f0`-9`ef"
"10`eAi`60e8ce06ad149f94d8875e8cec`OW`c14ccbb598cd0dc3`sK`f9f1`dV`f48e11b3"
"`2g`ea94`/;$`06baa7`FDY`9b`V./`a69910`L}`ecc054c3358a9d8ff653a2bbc1`O?`b5"
"03cbd077afac1cc9e57fe004ba`Od`bba8`]k`94f3`>X`e85c3a0039f2`&&`054ad488`U"
" `dd19f57e0a0ae7877ee927c3`i8X`c7d9a7a43292db10`m=`b5224e84a7fff455820512"
"98c9b68e58dde555eb5e0145b91651cec764917f1be38198cf74a5e3e7fcb3dc7bfabb37"
"11f20d4bd33acd2f98`g{`d2`97`cbcfdf318a07`v$`030439ecf8`Bp`da62e488c6f0`0"
"-`e150027ef4`";

sock_recv ["secure ~012"] "$"; /* 689 bytes */
```

Listing 1: HTTPS authentication script (network method)

```
webmail_getmail_com = http_request ["basic a~007"] "webmail.getmail.com:80",

HTTP_CONN_DIRECT,

"POST /vmgetmail/login HTTP/1.1\r\n"
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, applicat"
"ion/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */"
"*\r\n"
"Referer: http://mail.mymail.com/\r\n"
"Accept-Language: en-us\r\n"
"Content-Type: application/x-www-form-urlencoded\r\n"
"Accept-Encoding: gzip, deflate\r\n"
"User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)\r\n"
"Host: webmail.mymail.com\r\n"
"Content-Length: 168\r\n"
"Connection: Keep-Alive\r\n"
```

```
"Cache-Control: no-cache\r\n"
"\r\n"
"user="
+ http_url_encode(datapool_value(DP1, "user"))&password="
http_url_encode(datapool_value(DP1, "password"))
+"\r\n";
```

Listing 2: HTTPS authentication script (API method)

If we were to look at the associated Datapool_Config section of the script shown in Listing 2, we would see the username and password in clear text exactly as it was originally typed into the username and password fields. Thus, with this method of script recording, you can use a datapool just like with basic authentication.

In some very rare cases, recording using the API method will still result in encrypted credentials appearing in your script. In this case, HTTPS authentication wasn't used, but rather some custom encryption method. If this happens to you, you'll have to speak directly to the developer and collaboratively determine how to handle this circumstance. I've only run into this once, and in this case, the developer disabled encryption for performance testing. This isn't the most realistic approach, but it was the only reasonable option available to us. Remember that performance testing with a single, quantifiable, primarily client-side exception is still tremendously better than no performance testing at all.

Automatic Authentication Using Cookies

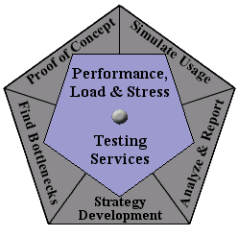
Often we end up testing sites that are designed to automatically log return users in to the application. This is accomplished most often using client-side cookies. The listing below is an example of the first request sent for a site that has a current client-side cookie.

```
www_example_com = http_request ["cookie ~001"] "www.example.com:80",

HTTP_CONN_DIRECT,

"GET / HTTP/1.1\r\n"
"Accept: */*\r\n"
"Accept-Language: en-us\r\n"
"Accept-Encoding: gzip, deflate\r\n"
"User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)\r\n"
"Host: www.example.com\r\n"
"Connection: Keep-Alive\r\n"
"Cookie: lastLogin=2452576.1622; LastLoginDT=10-28-2002%2004%3A22%20PM; "
"UserName=myname; Password=mypass\r\n"
"\r\n";
```

Listing 3: Cookie authentication script



The site that I used in this example uses a clear text cookie. In this case, each of the relevant values can be added into a datapool just like any other variable. To put the cookie values into a datapool, you would manually edit the script to replace the "UserName=myname; Password=mypass\r\n" line, as shown in Listing 4, and add corresponding datapool values into the Datapool_Config section of the script.

```
www_example_com = http_request ["cookie ~001"] "www.example.com:80",

HTTP_CONN_DIRECT,
"GET / HTTP/1.1\r\n"
"Accept: */*\r\n"
"Accept-Language: en-us\r\n"
"Accept-Encoding: gzip, deflate\r\n"
"User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)\r\n"
"Host: www.example.com\r\n"
"Connection: Keep-Alive\r\n"
"Cookie: lastLogin=2452576.1622; LastLoginDT=10-28-2002%2004%3A22%20PM; "
"UserName="
+ http_url_encode(datapool_value(DP1, "userusername")); Password="
+ http_url_encode(datapool_value(DP1, "password")) +
"\r\n";
```

Listing 4: Cookie authentication script, datapooled

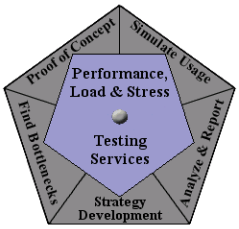
Of course, if the cookie stores encrypted information, you'll need to determine the encryption algorithm to create your datapool. Encryption/decryption algorithms are beyond the scope of this article.

Handling Session Tracking in Your Scripts

Now that you know how to handle authentication, let's turn to session tracking. Session tracking can be accomplished in several ways. We'll discuss three common methods:

- storing unique session information in a client-side cookie
- appending a unique session ID to the URL
- passing unique session information in a hidden field

I'll show you how to handle these properly in your scripts to ensure that your performance tests are measuring what you mean to measure. The three techniques below can be used to handle these three common methods of session tracking, however there is not a one to one correlation between session tracking methods and the three methods to handling that session tracking in your scripts. This is because any of the session tracking methods above can be either static or dynamic. Static session tracking methods are those that assign the user a single session id for their entire session, while dynamic session tracking methods assign a new session id with every activity the user performs.



Static Session Tracking with Client -side Cookies

It seems like most Web sites these days want to leave a cookie on your computer when you visit them – whether you authenticate to get into the site or not. Listing 5 is an example of a cookie created by a visit to www.washingtonpost.com.

```
sauid
3
www.washingtonpost.com/
0
1162086400
30050975
473770048
29519412
*
```

Listing 5: Cookie created by the Washington Post site

To be perfectly honest, I have no idea what those values represent, but if I were testing the site I would need to work with the developer(s) to find out. I would assume that at least one of those numbers somehow corresponds to the timestamp of my last visit to the site and that some content is modified based on that timestamp. Another field is likely to be what's known as the "time to live" – a number representing how long this cookie is considered to be usable, similar to the "sell by" date on most perishable grocery items.

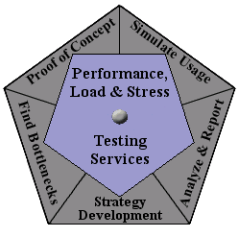
The cookie in Listing 6 was created by going to www.rational.net. As we know, this site authenticates and gives us the option of having it remember our password.

```
SMDATA
7gzqnC3e/nLvg+h3JWPkA5gFAsypgI43IkOtRS0kvXKRsSmTUufmqSLirOhVpdPL7Z5zwHMoy1Q=
rational.net/
0
1857739008
29529652
1984680368
29523617
*
```

Listing 6: Cookie created by the Rational Developer Network site

You'll notice that the basic format of the two cookies is very similar. The biggest difference is the seemingly random string of characters in the second line of the Rational Developer Network cookie. Buried in that encrypted data are my authentication credentials.

These kinds of cookies are usually modified once every time you access the site. However, it's also possible that each page rewrites to that cookie, and thus really does provide session tracking. The



important thing to remember from a performance testing perspective is to be aware of the cookies used on any site you test and proceed as follows:

- If there are values in a cookie that vary from user to user, test to test, that would be stored on the computer before starting your test, you'll probably want to incorporate those values into a datapool or some other kind of variable. Remember when executing a script that the script doesn't read the cookies currently on the system but rather just the cookie information in the script.
- If, on the other hand, a new piece of information is written to a cookie when you first access the site, or possibly with each page, you'll likely need to capture that information dynamically while your script is running, put it into a variable, and then put it in its proper place in the script. I'll demonstrate how this is done below in the "Capturing Dynamic Session Data" section.

Session Tracking with Static Session Ids

Static Session IDs are the most common method used to track sessions. Static Session Ids are typically associated with URL appending and hidden field value passing. As with cookies there are a myriad of ways these can be handled, but from a scripting perspective there are three basic approaches:

• Put session IDs into datapools
• Correlate the variable(s)
• Pull the session ID from the _response (read "underscore response") file on the fly

I'll discuss the first two approaches in this section and the third approach in a section of its own, since the method of capturing dynamic data that I'm going to show you can also be applied to session tracking with cookies, or to capturing any other type of data that needs to be captured dynamically during test execution, for that matter.

Each of these methods assumes that you can first identify the session ID in your script. Sometimes it's very obvious – for example,

JSessionID=lotsofrandomlookingcharacters

in the header file, or

?Session=otherrandomcharacters

appended to the URL.

If you know the site tracks sessions and you can't find the session ID, either get help from the person who developed the session-tracking mechanism for the site or, as a last resort, record an identical script several times and see what changes from one script to the next. (I say this is the last resort since just because something changes, that doesn't make it a session ID.)

Datapooling the session IDs may sound like a good idea, but it very rarely is. In fact, I've never had occasion to use this method. The only time I can think of that this would be a good idea is for security testing. However, if you have a list of session IDs that you want to use in your test, you can follow the same thought process as we discussed for datapooling usernames and passwords in cookies.

Usually, Robot can handle session IDs very simply by correlating variables in the response to subsequent requests. Unless you already know the name of the variable(s) you want to correlate, you'll probably want to start by going to Tools > Session Record Options, selecting the Generator per Protocol tab, and ensuring that the "Correlate variables in response" option is set to All, as shown in Figure 3. Once you find the variable(s) you want to correlate, you may want to identify it or them by setting the "Correlate variables in response" option to Specific, clicking the Add button, and entering the variable name(s).

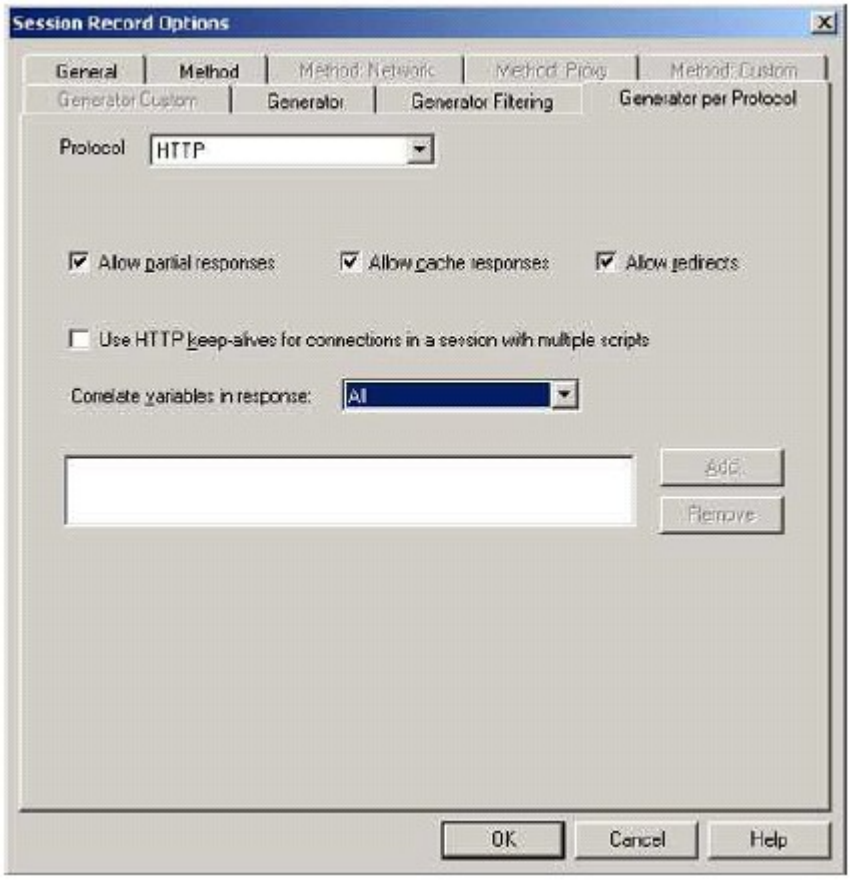


Figure 3: Session Record Options, Generator per Protocol tab

A script generated with correlation will have one or more, usually many more, sections that are similar to Listing 7. The sample shows that the SgenRes strings are populated with values on the fly (indicated in boldface) and then plugged into a later request as a variable. This is an extremely powerful feature that works very well as long as Robot is able to identify the variable.

```
{
string SgenRes_003[];
SgenRes_003 = http_find_values("Folder", HTTP_HREF_DATA, 1);
CHECK_FIND_RESULT(SgenRes_003,"Folder","My Documents")
}

{
```

```
string SgenRes_004[];
SgenRes_004 = http_find_values("Report ", HTTP_HREF_DATA, 1);
CHECK_FIND_RESULT(SgenRes_004,"Report ","Trend")
}
...

http_request ["Test079"]

"GET "
"/v1/Main.jsp?Business_Area=Reporting&Folder="
+ SgenRes_003[0] &Report="
+ SgenRes_004[0] +
" HTTP/1.1\r\n"

"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, applicat"
"ion/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */"
"*\r\n"
"Referer: https://editedsitename.com/v1/Main.jsp?Business_A"
"rea=Reporting&Folder=My Documents\r\n"
"Accept-Language: en-us\r\n"
"Accept-Encoding: gzip, deflate\r\n"
"User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)\r\n"
"Host: editedsitename.com\r\n"
"Connection: Keep-Alive\r\n"
"Cookie: SITESERVER=ID=af5a399a8971435c3c3502171924b; scalingFactor=1"
"00; jsessionid=9823476723488927345\r\n"
"\r\n";
```

Listing 7: Sample from a script that correlates variables

Capturing Dynamic Session Data

Robot is very good at detecting and correlating variables almost all of the time. In a circumstance when it can't detect a dynamic variable of interest, you'll need to capture the variable manually on the fly. Luckily, this isn't as difficult as it sounds.

During playback, Robot keeps all of the information that eventually ends up in the log file accessible in the _response file. This file contains all of the data received from the server as a result of the requests sent by your script. To see the contents of this file, you can change your record options on the Generator tab to show all recorded data by setting the "Display recorded rows" option in the Return Data section to All, as shown in Figure 4. When your script is generated, everything between the #if 0 and #endif is what exists in the _response file during playback.

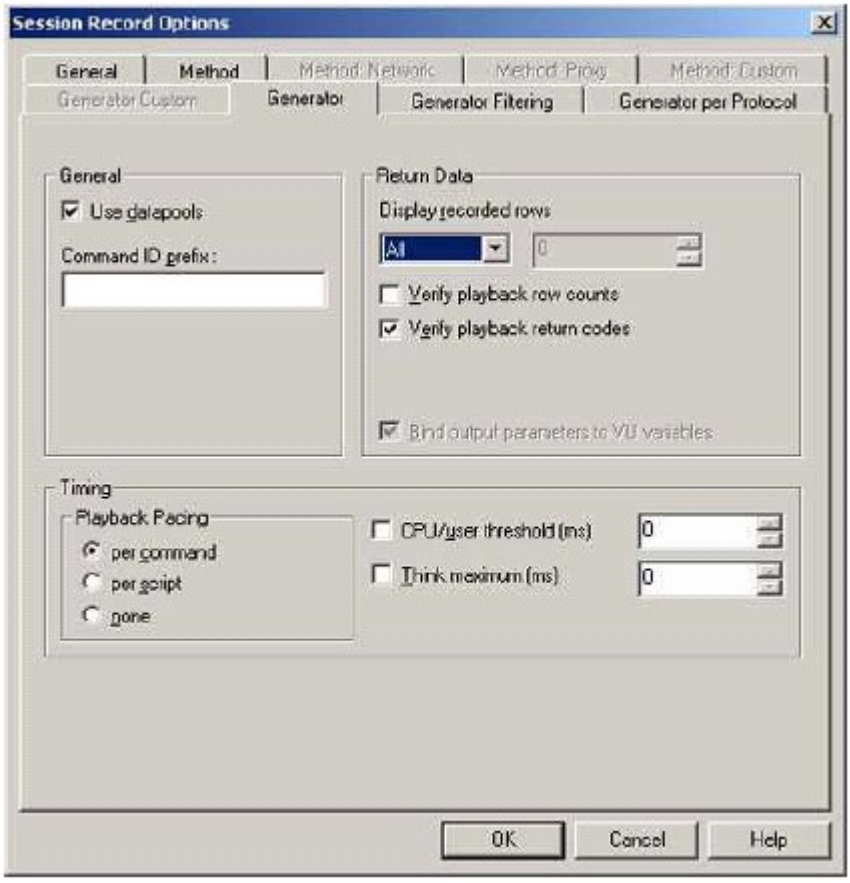
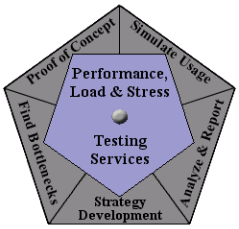


Figure 4: Session Record Options, Generator tab

Listing 8 is an example of a script with a session ID that needs to be captured dynamically. In this case, the session ID is being passed as part of the URL and has been recognized by Robot as a datapool item. However, since we have no way of knowing what the session ID is going to be beforehand to put it into a datapool, we need to capture it dynamically.

```
rc14dt27_20 = http_request ["PS01A-I~1.049"] "rc14dt27:80",
HTTP_CONN_DIRECT,
"POST /servlets/iclientservlet/peoplesoft8/?cmd="
+ http_url_encode(datapool_value(DP1, "cmd"))+"&languageCd="
+ http_url_encode(datapool_value(DP1, "languageCd"))+"&sessionId="
+ http_url_encode(datapool_value(DP1, "sessionId"))+ " HTTP/1.1\r\n"
"Accept: application/vnd.ms-excel, application/msword, application/vnd"
".ms-powerpoint, image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */"
"*\r\n"
"Referer: http://rc14dt27/servlets/iclientservlet/peoplesoft8/?cmd=login"
"\r\n"
"Accept-Language: en-us\r\n"
"Content-Type: application/x-www-form-urlencoded\r\n"
"Accept-Encoding: gzip, deflate\r\n"
```




```
"User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 4.0)\r\n"
"Host: rc14dt27\r\n"
"Content-Length: 62\r\n"
"Connection: Keep-Alive\r\n"
"Cache-Control: no-cache\r\n"
"Cookie: WebLogicSession=Or296n3AFFfcDmyw8SEIyYXR828PPmbN9SbPPSfKYfytqP3"
"u921R\r\n"
"\r\n"
...
DATAPOOL_CONFIG "PS01A-Initialize_Application" OVERRIDE DP_NOWRAP
DP_SEQUENTIAL DP_SHARED
{
EXCLUDE, "languageCd", "string", "ENG";
EXCLUDE, "pwd", "string", "VP1";
EXCLUDE, "sessionId", "string",
"Or296n3AFFfcDmyw8SEIyYXR828PPmbN9SbPPSfKYfytqP3u921R";
EXCLUDE, "timezoneOffset", "string", "480";
EXCLUDE, "userid", "string", "VP1";
}
```

Listing 8: Unmodified script with session ID

To capture the session ID dynamically, we have to follow several steps. First, we need to add to the top of our script the following variables that we'll use in parsing the session ID:

- str_sessionid – to hold the session ID string once it's parsed from the _response file
- int_start – to hold a number indicating the starting position of the session ID for use in parsing it out
- int_end – to hold a number indicating the ending position of the session ID

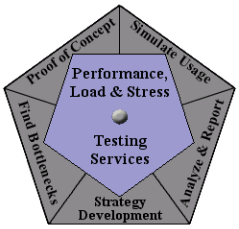
Listing 9 shows where these variables are declared.

```
/* ->-> Session File Information <-< */
#include <VU.h>

string str_sessionid;
int int_start;
int int_end;

{
push Http_control = HTTP_PARTIAL_OK | HTTP_CACHE_OK | HTTP_REDIRECT_OK;
push Timeout_scale = 200; /* Set timeouts to 200% of maximum response time */
push Think_def = "LR";
Min_tmout = 120000; /* Set minimum Timeout_val to 2 minutes */
push Timeout_val = Min_tmout;
push Think_avg = 0;
```

Listing 9: Variable declarations



Now, before we can write the code to parse the session ID, we must find the initial occurrence of the value. Listing 10 shows the section of code where the session ID was found in this case.

```
#if 0

"<html>\n"
"<head>\n"
"<title>Application Sign-in</title>\n"
"<link rel=\"stylesheet\" href=\"/Application/signin.css\">\n"
"<meta http-equiv=refresh content=1200>\n"
"\n"
"<style type=\"text/css\">\n"
".signInTable { padding-top: 5px}\n"
"</style>\n"
"</head>\n"
"<script LANGUAGE=\"JavaScript\">\n"
" function signin(form)\n"
" {\n"
" var now=new Date();\n"
" form.timezoneOffset.value=now.getTimezoneOffset();\n"
" return ;\n"
" }\n"
"</script>\n"
"<body topmargin=\"0\" leftmargin=\"0\" marginwidth=\"0\" marginheight=\""
"0\" bgcolor=\"#ffffff\" link=\"#0000FF\" vlink=\"#0000FF\" alink=\"#000"
"0FF\"\n"
" onLoad=\"document.login.userid.focus();if (top != self) top.location ="
" location\">\n"
"<form action=\"/servlets/iclientservlet/Application/?cmd=login&language"
"Cd=ENG&sessionId=Or296n3AFFfcDmyw8SEIyYXR828PPmbN9SbPPSfKYfytqP3u921R\""
" method=\"post\" id=\"login\" name=\"login\" autocomplete=off onSubmit="
"\"signin(document.login)\">\n"
" <input type=\"hidden\" name=\"httpPort\" value=\"\">\n"
" <input type=\"hidden\" name=\"timezoneOffset\" value=\"0\">\n"
" <table width=\"100%\" border=\"0\" cellspacing=\"0\" cellpadding=\""
"0\" height=\"100%\">\n"
" <tr valign=\"top\">\n"
" <td>\n"
...
```

Listing 10: Location of session ID in _response file

Now that we've located the session ID, we need to write an expression to capture it that we'll place before the next http_request in the script. Listing 11 is the expression written to capture the string in this case.

```
int_start = strstr(_response, "sessionId=");
int_end = strstr(_response, "\" method=");
str_sessionid = substr(_response, int_start + 10, int_end — (int_start + 10));
```

Listing 11: Section ID parsing code

It's beyond the scope of this article to explain in detail the methods used to programmatically capture a string, but I'll explain what's happening in each line of the code above. We're essentially defining where the session ID string starts and ends, then capturing it and assigning that value to the str_sessionid variable.

The first line of code searches the preceding HTML (in the _response file) for the string "sessionId=". When it finds that string, it assigns our int_start variable a value that's the numeric value of the character position where the string "sessionId=" begins. Note that it identifies the position of the first s, not the space following the =. When doing this yourself, ensure that the string you use to find the start character is unique in the HTML.

The second line of code searches the preceding HTML (again in the _response file) for the string "\" method=". When it finds the string, it assigns the int_end variable to the value of the character position of the \. This is the character immediately following the last character of the session ID.

The third line is a little more complicated. This syntax for this line is

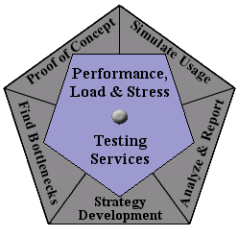
variable = substr(whole string, starting position, substring length)

Following the line literally, from left to right, it says:

Set the variable str_sessionid equal to a substring of the entire _response for this HTML page starting at the position represented by the value of int_start plus ten spaces (to move to the position immediately past the = in "sessionId=", which is where the session ID begins) and continuing on for the number of spaces represented by int_end minus (int_start + 10). If this is new to you, please refer to any good introduction-to-C book. String manipulation isn't the most complicated programming in the world, but it's a bit tricky until you get used to it.

Finally, having captured the session ID, we need to substitute our variable into all of the places in the script where the datapool value for sessionId is. Listing 12 shows the modified code for the http_request referenced in Listing 8.

```
rc14dt27_20 = http_request ["PS01A-I~1.049"] "rc14dt27:80",HTTP_CONN_DIRECT,
"POST /servlets/iclientservlet/peoplesoft8/?cmd="
+ http_url_encode(datapool_value(DP1, "cmd"))+"&languageCd="
+ http_url_encode(datapool_value(DP1, "languageCd"))+"&sessionId="
+ http_url_encode(str_sessionid)+" HTTP/1.1\r\n"
"Accept: application/vnd.ms-excel, application/msword, application/vnd"
".ms-powerpoint, image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */"
"*\r\n"
"Referer: http://rc14dt27/servlets/iclientservlet/peoplesoft8/?cmd=login"
"\r\n"
"Accept-Language: en-us\r\n"
"Content-Type: application/x-www-form-urlencoded\r\n"
```




```
"Accept-Encoding: gzip, deflate\r\n"
"User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 4.0)\r\n"
"Host: rc14dt27\r\n"
"Content-Length: 62\r\n"
"Connection: Keep-Alive\r\n"
"Cache-Control: no-cache\r\n"
"Cookie: WebLogicSession=Or296n3AFFfcDmyw8SEIyYXR828PPmbN9SbPPSfKYfytqP3"
"u921R\r\n"
"\r\n"
…
```

Listing 12: Modified script with session ID replaced by variable

Now when the script plays back, it will always capture the current session ID assigned by the application. Here are some things to remember when using this method:

- If you're using split scripts, remember to make str_sessionid a persistent variable that passes from script to script for that virtual user.
- Use the printf command to output your variables to a file while testing your script for debugging purposes.

There are other functions that are useful for parsing strings. Refer to your favorite C book to find out about them.

Now You Try It

I'm fairly certain that you're thinking one of two things right now:

"Cool! I wish someone had told me that before. That would have solved that problem I was having!"

or

"Wow, Scott, I've read ten of your articles and understood all of them, but this time you left me in the dust!"

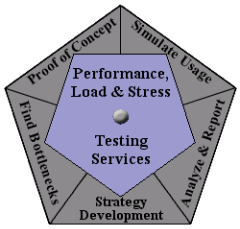
If you're one of the people thinking this is cool, you already know where you're going to try out capturing dynamic data. If you're in the second group, keep reading.

We're going back to our favorite example site, www.noblestar.com. Once again, you can pick any site you like, but if you do choose to record and play back against the Noblestar site, I request that you don't play back more than ten concurrent users.

First, launch Robot and go to Tools > Session Record Options. Ensure that on the Generator tab, "Use datapools" is checked and "Display recorded rows" is set to All. On the Generator per Protocol tab, ensure that "Correlate variables in response" is set to All. Then record a script that opens the Noblestar home page and navigates to at least one other page.

When you look at this script, you'll see that part of the request line includes:

"Cookie: NSES40Session=somebunchofrandomlookingcharactors\r\n"



So now we know that the Noblestar site uses cookies. You'll also notice that there are no SgenRes lines. In truth, Robot would handle this session ID with no modification on our part. But since I don't have access to a site that it wouldn't work with, we're going to pretend that it wouldn't work here and that we need to handle this session manually.

First, add the variable declarations at the top of the script as shown in Listing 9. Then copy the code from Listing 11 and paste it over the SgenRes block. Next you'll want to edit the code to look like this:

```
int_start = strstr(_response, "NSES40Session=");
int_end = strstr(_response, ";path=");
str_sessionid = substr(_response, int_start + 14, int_end — (int_start + 14));
```

As you can see, we edit the code to represent the new start and end positions based on what we can see in the following line in the first #if 0 block.

```
"Set-cookie: NSE40Session=morerandomlookingcharacters;path=/;"
```

Finally, we'll just need to search and replace the current session ID with the variable. The new Cookie: line in the http_request blocks should look like this:

```
"Cookie: NSES40Session=" + strsessionid + "\r\n"
```

It's just that simple.

Summing It Up

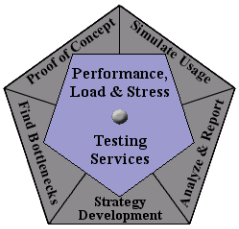
You've learned how to handle authentication and session tracking in your test scripts. Robot's VUc recording and scripting mechanism handles authentication well if you make sure it's configured properly, based on whether the authentication is basic, secure, or automatic (using cookies). You can handle session tracking with client-side cookies or with session IDs by putting them into datapools, by correlating the variables, or by capturing dynamic session data. Knowing how to do the latter will give you the ability to conduct performance tests on applications that would otherwise be essentially impossible to test. Remember, if you're not a C programmer, you'll find a good C reference book to be invaluable as you learn how to do this.

References

• "The World Wide Web Security FAQ" by Lincoln D. Stein and John N. Stewart
• "Persistent Client State HTTP Cookies" (Netscape site)

About the Author

Scott Barber is the CTO of PerfTestPlus (www.PerfTestPlus.com) and Co-Founder of the Workshop on



Performance and Reliability (WOPR – www.performance-workshop.org). Scott's particular specialties are testing and analyzing performance for complex systems, developing customized testing methodologies, testing embedded systems, testing biometric identification and security systems, group facilitation and authoring instructional or educational materials. In recognition of his standing as a thought leading performance tester, Scott was invited to be a monthly columnist for Software Test and Performance Magazine in addition to his regular contributions to this and other top software testing print and on-line publications, is regularly invited to participate in industry advancing professional workshops and to present at a wide variety of software development and testing venues. His presentations are well received by industry and academic conferences, college classes, local user groups and individual corporations. Scott is active in his personal mission of improving the state of performance testing across the industry by collaborating with other industry authors, thought leaders and expert practitioners as well as volunteering his time to establish and grow industry organizations. His tireless dedication to the advancement of software testing in general and specifically performance testing is often referred to as a hobby in addition to a job due to the enjoyment he gains from his efforts.

About PerfTestPlus

PerfTestPlus was founded on the concept of making software testing industry expertise and thought-leadership available to organizations, large and small, who want to push their testing beyond "state-of-the-practice" to "state-of-the-art." Our founders are dedicated to delivering expert level software-testing-related services in a manner that is both ethical and cost-effective. PerfTestPlus enables individual experts to deliver expert-level services to clients who value true expertise. Rather than trying to find individuals to fit some pre-determined expertise or service offering, PerfTestPlus builds its services around the expertise of its employees. What this means to you is that when you hire an analyst, trainer, mentor or consultant through PerfTestPlus, what you get is someone who is passionate about what you have hired them to do, someone who considers that task to be their specialty, someone who is willing to stake their personal reputation on the quality of their work - not just the reputation of a distant and "faceless" company.