User Experience, Not Metrics

by:
R. Scott Barber

Part 13: Working with Unrecognized Protocols

So you're pretty confident in your ability to test a web-based application. You've got some successful testing engagements behind you and you're feeling good. Now your boss comes to you and says…

"Great job on those performance tests! I've got another one for you. Bob has built a custom application that needs to be performance tested. I'm sure you can handle it. Oh, by the way, it doesn't use HTTP. Good luck!"

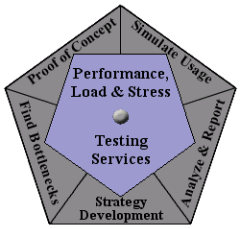
This is the final installment of the "User Experience, Not Metrics" series, which focuses on correlating customer satisfaction with your application's performance as experienced by users. This article is intended for intermediate to advanced Rational TestStudio users. A general knowledge of TCP/IP communications protocols will aid in the application of the concepts discussed.

Communications Protocols

Thus far in the series all of our discussions have focused on scripts that used the Hyper Text Transfer Protocol (HTTP). The HTTP protocol is the predominant way that web servers communicate with web browsers. HTTP is not a language, but rather just a standard by which data requested and received. Let's take a few minutes to discuss what you probably know instinctively about HTTP, but may not have ever thought about consciously. This will give us some common ground around which discuss other protocols.

HTTP is essentially broken into a format to request files, a format for responses to those requests and a standard format for all header files. HTTP requests (see listing 1) typically request a file (HTML, GIF, JPG) to be sent over the Internet back to the program requesting it, usually a web browser. For each request the server may send back one or more responses. The response includes text header (see listing 2) and the requested data (see listing 3).

```
"GET /solutions/papers/whitepapers.jsp 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://www.noblestar.com/"
"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: www.wmata.com\r\n"
"Connection: Keep-Alive\r\n"
"\r\n";
```

Listing 1: Sample HTTP Request

```
"HTTP/1.1 200 OK\r\n"
"Server: Netscape-Enterprise/4.1\r\n"
"Date: Sat, 18 Jan 2003 21:03:51 GMT\r\n"
"Content-type: text/html; charset=Cp1252\r\n"
"Set-cookie: NSES40Session=e0f%253A3e29bba2%253A224c54f45191a4ac;path=/;"
"expires=Sat, 18-Jan-2003 21:33:51 GMT\r\n"
"Transfer-Encoding: chunked\r\n"
"\r\n"
```

Listing 2: Sample HTTP Receive Header

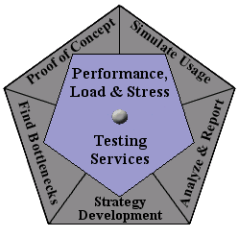
```
"<HTML>\r\n"
"<HEAD>\r\n"
"<TITLE>Solutions - White Papers</TITLE>\r\n"
"<LINK REL=STYLESHEET TYPE=\"text/css\" HREF=\"/css/main.css\">\r\n"
"<SCRIPT LANGUAGE=\"JavaScript\" SRC=\"/js/menu.js\"></SCRIPT>\r\n"
"<SCRIPT LANGUAGE=\"JavaScript\" SRC=\"/js/topmenu.js\"></SCRIPT>\r\n"
"</HEAD>\r\n"
"<BODY BACKGROUND=\"/images/back_pat.gif\" TOPMARGIN=0 LEFTMARGIN=0 RIGH"
"TMARGIN=0 MARGINHEIGHT=0 MARGINWIDTH=0>\r\n"
"<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0>\r\n"
" <TR BGCOLOR=\"#000066\" VALIGN=TOP>\r\n"
"   <TD COLSPAN=7><A HREF=\"/index.jsp\"\r\n"
"    onMouseOver=\"clearMain()\" onMouseOut=\"reset()\" onClick=\"set(""
",0)\"><IMG\r\n"
…
```

Listing 3: Sample HTTP Receive

HTTP is a high-level communication protocol, it does not concern itself about such things as data packets and retries, these are handled by the TCP/IP communication layers. TCP/IP is a low-level communication protocol. (For more information on TCP/IP see this overview published by Yale University. http://www.yale.edu/pclt/COMM/TCPIP.HTM) HTTP could be thought of as a sub-protocol to TCP/IP.

If you were to draw a comparison to written mail, the letter would be the file being requested or received, HTTP would be the rules dictating how the envelope is addressed, and TCP/IP is the postal service responsible for picking up and delivering the letters.

Conceptually, all other communications protocols work the same way – it's just the rules and formats that change. Some protocols transmit data in formats that are relatively easy for human eyes to read (like HTTP), while others transmit data in a compressed or encrypted format that is essentially



unreadable to human eyes (like HTTPS).

Rational TestStudio Vu Scripts captures traffic at the communication protocol level, as do all of the top load generation tools. So you may be wondering, if the traffic is captured, what makes a particular protocol either "supported" or "unsupported"? The "support" that is being referred to is script generation support for playback. Just capturing the traffic isn't enough to be able to playback that traffic in a way that simulates real users. It is possible to capture and playback many unsupported protocols successfully, but there are additional considerations.

Supported Protocols

Rational TestStudio officially supports scripts generated from the capture of traffic from the following communications protocols:

- TCP/IP Socket
- HTTP / HTTPS
- DBLIB
- DCOM
- IIOP
- Jolt
- ODBC
- Oracle
- SQL Server
- Sybase
- Tuxedo

What this means is that if the application you are testing uses one of the protocols above, TestStudio will generate scripts that can more or less be played back as-is. As we have seen through previous articles in this series, recorded scripts do often require some editing prior to multi-user playback. This editing is just part of performance testing regardless of whether or not the protocol is supported by the tool and not part of the discussion of this article.

TCP/IP Socket can be an exception that is discussed in more detail below.

Unsupported Protocols

Every communication protocol that is not on the list in the previous section is considered to be unsupported by TestStudio. Some common examples of unsupported protocols are:

- FTP (File Transfer Protocol)
- SNMP (Simple Network Messaging Protocol)
- SMTP (Simple Mail Transfer Protocol)
- POP3 (Post Office Protocol - 3)
- WAP (Wireless Application Protocol)

- VoIP (Voice over Internet Protocol)

Just because these protocols are unsupported does not mean that TestStudio Vu Scripts cannot be used to performance test applications using these protocols. Any TCP/IP type communication can be captured using the correct Session Record Options. All of the protocols listed above are TCP/IP protocols as HTTP is. You will see below that scripts recorded using the TCP/IP Socket recording options will look different than HTTP scripts, but remember that all of these protocols are just different ways to format requests and receives based on the applications and types of data being sent.

To configure these options, in Rational Robot select Tools -> Session Record Options from the menu bar. Select Network recorder on the Method tab (figure 1) and manual filtering on the Generator Filtering tab (figure 2). While on the Generator Filtering tab, click the Advanced button to ensure that the both Well-known protocols and unrecognized protocols options are checked (figure 3).

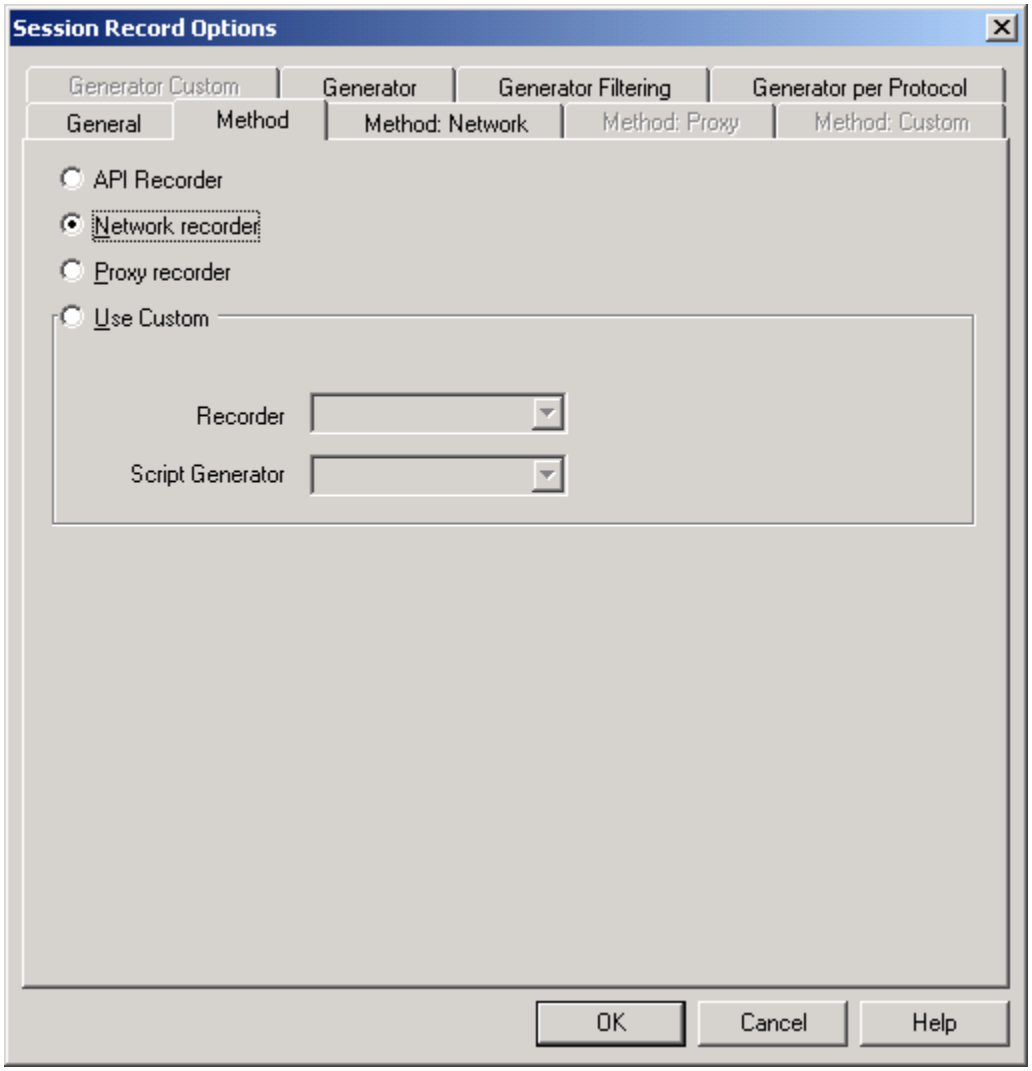


Figure 1: Network recorder Option

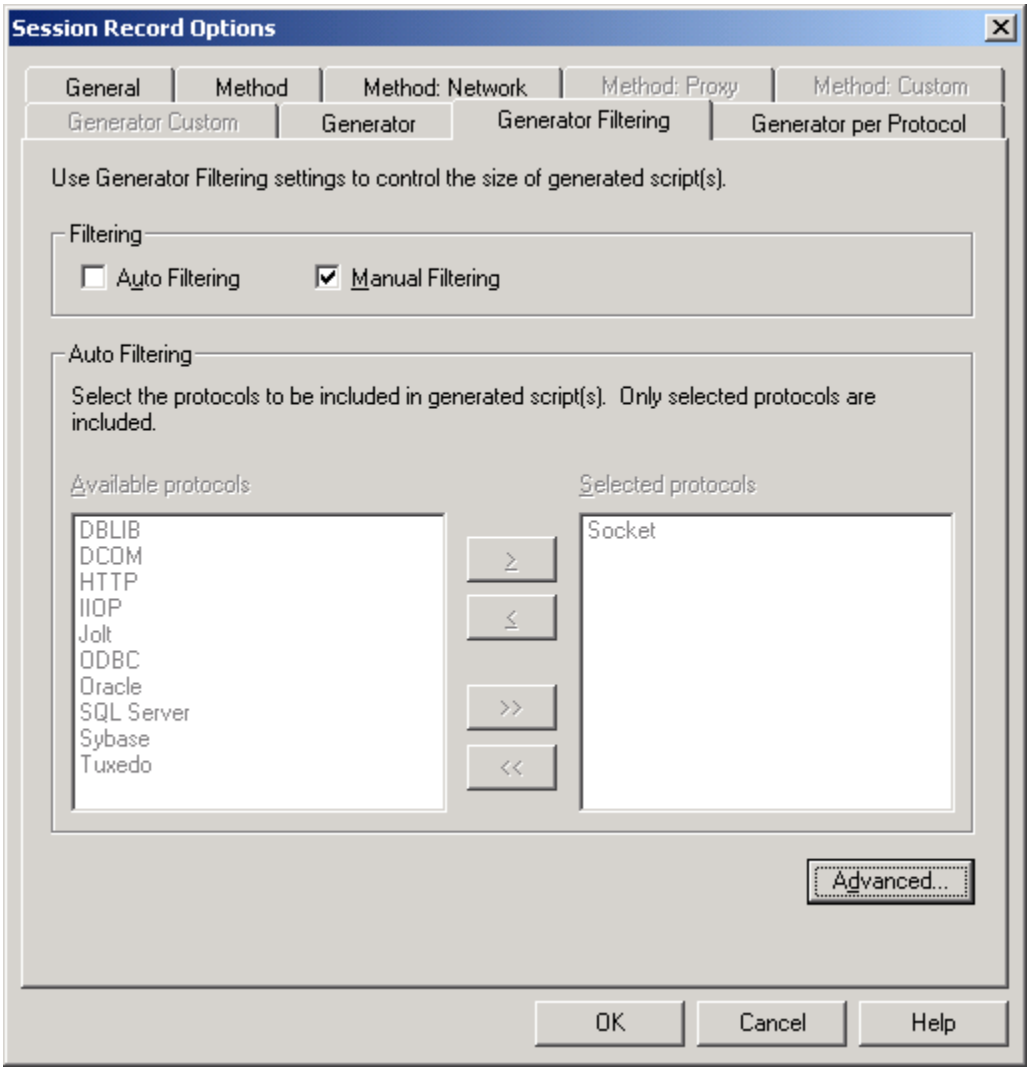


Figure 2: Generator Filtering

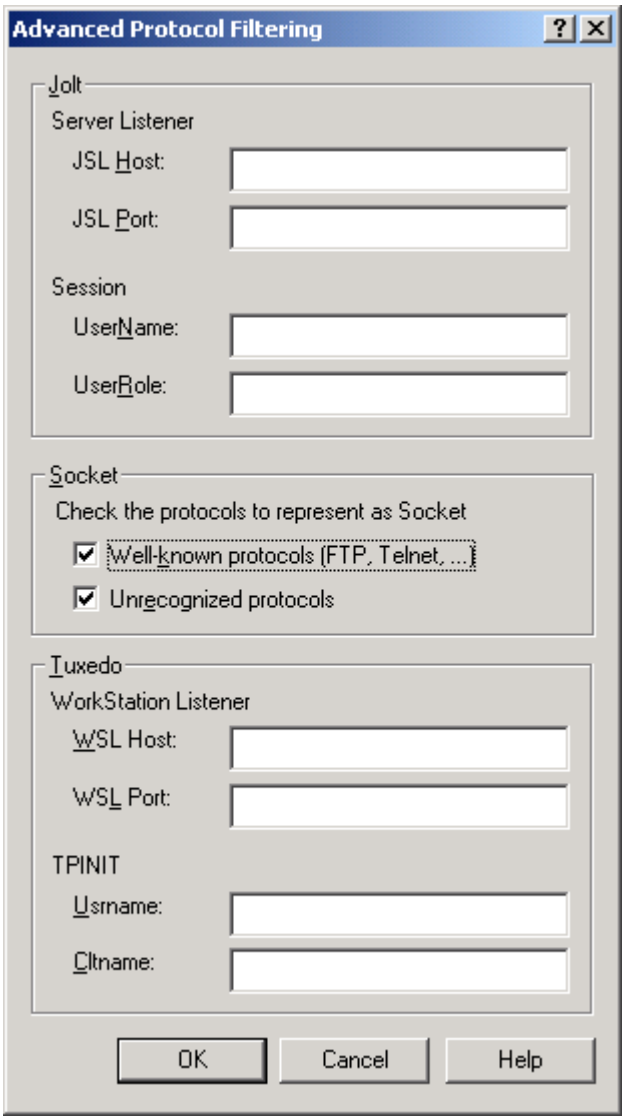


Figure 3: Generator Filtering, Advanced

If you know the name or IP of the server that you will be recording against, you can specify that server by going to the Method: Network tab (figure 4) adding the server information by pressing the Manage Computers button and filling in the appropriate information. Finally, select the server that you have added. This is done to ensure that "extra" network traffic isn't unintentionally captured. This should only be done if you are certain that all of the traffic that you are interested will be from that server.

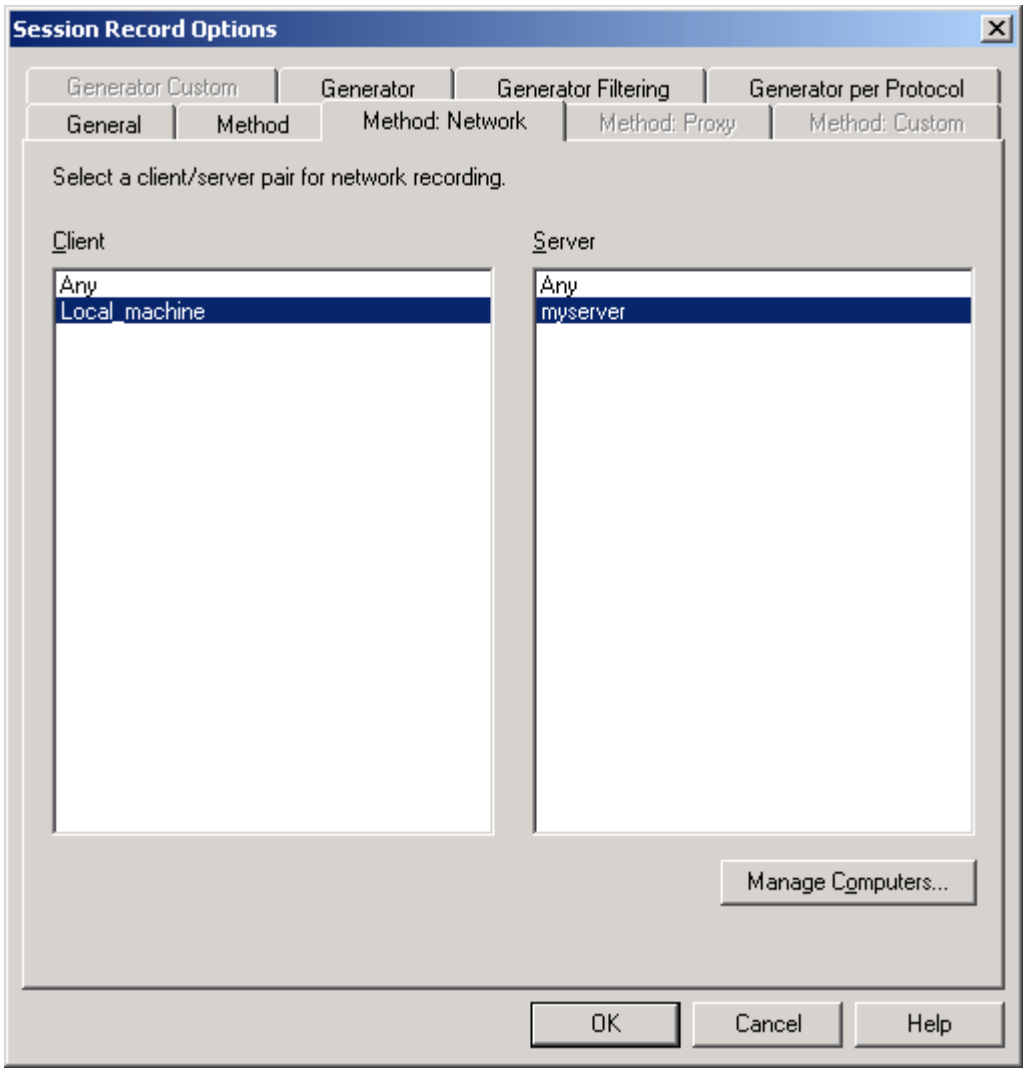


Figure 4: Method: Network tab

Once you have configured these settings and recorded your first script you will see that communication you have captured falls into one of two categories. I refer to these "Clear Text" protocols and "Encoded" protocols. I put those terms in quotes because they are not always technically correct, but they are descriptively accurate.

In a script of a "Clear Text" protocol you can essentially read what you recorded. For instance, listing 4 shows a script segment of SMTP traffic.

```
#include <VU.h>
int n;
{
Dmail_yahoo_com = sock_connect("SMTP001", "mail.yahoo.com:25");
```

```
set Server_connection = Dmail_yahoo_com;
n = sock_isinput();
sock_nrecv ["SMTP002"] n;

sock_send "helo john.doe.com\r\n";
n = sock_isinput();
sock_nrecv ["SMTP003"] n;
sock_send "mail from: sbarber@noblestar.com\r\n";
n = sock_isinput();
sock_nrecv ["SMTP006"] n;
sock_send "rcpt to: cwalters@noblestar.com\r\n";
n = sock_isinput();
sock_nrecv ["SMTP008"] n;
…
```

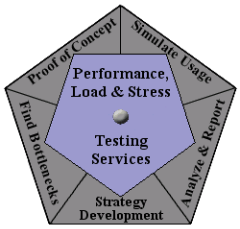
Listing 4: Sample "Clear Text" Protocol Script

In this script segment, you can see that the mail message that was recorded was sent from sbarber (me) to cwalters (my officemate who delivered this topic in a presentation for the Rational User's Conference, 2002 – that presentation is available here {link}). While you may not know all of the details of the SMTP protocol, you will be able to recognize much of the text.

Listing 5 shows a sample of an "Encoded" protocol script. You will notice that this section of script is completely un-readable to human eyes. This is a section of script recorded against a client/server application that was transmitting a short statement in a comment field.

```
sock_send
"`14030000010116030000`8~Pv\b`d3`pB`80aa31c3865e8ea9bf`(7`c4f0d0`&MCl`8787"
"fb56f0a9ae36f632d9de548ae7dd29ed72ff55e5`;@\tDu`952bacc8170300017de03dde"
"173a9620fb`;C`d941e07011`i1`a2dbb05db693dcb9f3df35fbe82f141c9ca65c90cc7e"
"08a3a7509024f775842089b3`Ul`c2`%lO`0e152d886de3d028d609`~wx`d7222e04e689"
"bfa64ce897`+.`eb5d17`&c`e379db44da`!G\fvc]`d0240cc5f2c520fdef`F4`0753b0aa"
"a81a96dac3d49764fbe47d7f4202`_#`19cd7683146af4ec`c{`00a2fa`AJ`5c31b61d18"
"34ebbe0f`_b`c49a7a10f3600297a59a7eebbe389bbad9a88fb137c9e9ecea`_z`01db11"
"ab3ceb89`p^h\f`d79085d8740de756052293a2fbdb7afcb8a43381dd3193b8`K3`e414`u"
"Q`f0ea`=_`8c`k?[,`dc`t.`048d2fc9cf741a7f08a6bfd129bff74f0dc9efe6ab6bcd24"
"d3c4e42d007f`S,f$,;`c5e0b2ae0ef01ae13b1673b70ccc`*5`aa96b73ccaf8`a;`d9`s"
"H`8245c5002cb4859396ad10a8de09cde283160a0c4b86fbf3a194024d8a`d/`0bb63bf9"
"`#oB`9a1bf78adf`c*`d7b6eccae9d3c1c411f0b077841fe5cbd9`L69`9ad497`WiW`b49d"
"4d870d0cee618d`o";

sock_nrecv ["test006"] 520;
```



Listing 5: Sample "Encoded" Protocol Script

In both cases there are two basic things to notice. First, it is worth noting that TestSudio handles the protocol headers (and therefore socket connections) automatically regardless of the data that is being transmitted (see listing 6). It is generally safe leave everything that isn't between the quotes ("") in the sock_send block.

```
hostname_1 = sock_connect("test004", "hostname:port");
set Server_connection = hostname_1;

sock_send
   "somekindofcontent";

sock_nrecv ["test005"] 146;
sock_disconnect(hostname_1);
```

Listing 6: Sample Socket Connections

The second part is that between the quotes ("") in the sock_send block that we discuss in the sections below.

Working with Unsupported "Clear Text" Protocols

Working with "clear text" protocols is really pretty straight forward. TestStudio will handle the manipulation of the socket connections for you. All you have to do is manipulate the parameters you may want to vary (or datapool). For purposes of demonstration, a modified version of the SMTP script from listing 4 is shown below. The script was modified to send the mail message to a variety of recipients as entered into a datapool.

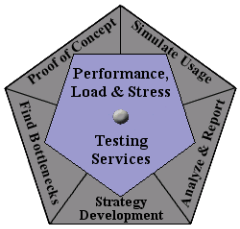
```
#include <VU.h>
int n;
{

DP1 = datapool_open("smtp_datapool");
datapool_fetch(DP1);

Dmail_yahoo_com = sock_connect("SMTP001", "mail.yahoo.com:25");

set Server_connection = Dmail_yahoo_com;
n = sock_isinput();
sock_nrecv ["SMTP002"] n;

sock_send "helo john.doe.com\r\n";
n = sock_isinput();
sock_nrecv ["SMTP003"] n;
```




```
sock_send "mail from: sbarber@noblestar.com\r\n";
n = sock_isinput();
sock_nrecv ["SMTP006"] n;
sock_send "rcpt to: “
+ http_url_encode(datapool_value(DP1, ename"))+
”@noblestar.com\r\n";
n = sock_isinput();
sock_nrecv ["SMTP008"] n;
…

DATAPOOL_CONFIG " smtp_datapool " OVERRIDE  DP_SEQUENTIAL DP_SHARED
DP_NOWRAP
{
      INCLUDE, "ename", "string", "cwalters";
}
```

Listing 7: Modified SMTP script

It is also possible to build custom functions for relatively simple protocols like this one, but detailing that process would take too long for the confines of this article. For an example of custom SMTP functions that allow you to build SMTP scripts without even recording, see the PowerPoint slideshow {link} that Chris Walters and I presented at RUC 2002.

Working with Unsupported “Encoded” Protocols

“Encoded” protocols are more complicated. The key to working with them is identifying what changes between each user, and what series of characters represents the changes between users. Detecting these differences and handling them programmatically is not always easy, but the basic process is simple. To determine what changes and what needs to be handled use these steps.

Step 1: Record the script that you will ultimately want to play back (script 1)

Step 2: Record that script again, EXACTLY the same way you did before (script 2)

*Step 3: Record that identical script yet again (script 3)

Step 4: Compare script 1, script 2 and script 3 to identify all of the differences in between the quotes (“”) in the sock_send block.

Step 5: Parameterize those differences from the _response file. (see articles 11 and 12 for more information on capturing return data from the _response file using regular expressions.

Step 6: Record the script again, except this time change any transactional data that will be different among users (i.e. username, password, account number, etc) (script 4)

Step 7: Compare script 1 and script 4 and identify all of the differences in between the quotes (“”) in the sock_send block that were not identified in Step 4.

Step 8: Add datapools to handle varied data identified in Step 7.

Step 8a: Since it may not be easy to determine what 'encoded' characters represent different usernames, passwords etc, you may need to record several scripts changing only a single parameter to either determine the encryption or simply capture all of the different options you wish to datapool. This can be a very tedious process.

*Often Step 3 is unnecessary, but there is no way to know until after you determine the patterns of differences among scripts. It is always safer to compare 3 scripts initially to be sure that you have identified the entire pattern.

Comparing these files visually can be very difficult. To assist in this process I have found two quality file compare tools. CSDiff http://www.componentsoftware.com/products/csdiff/download.htm and ExamDiff http://www.prestosoft.com/ps.asp?page=edp_examdiff are similar programs that automatically identify differences between two similar files. Simply launch the program and browse to the recorded scripts (*.s files) and it will show you the differences between these files. I actually recommend both as they are each better with different types of files. Screenshots comparing the same section of code from each are included below.

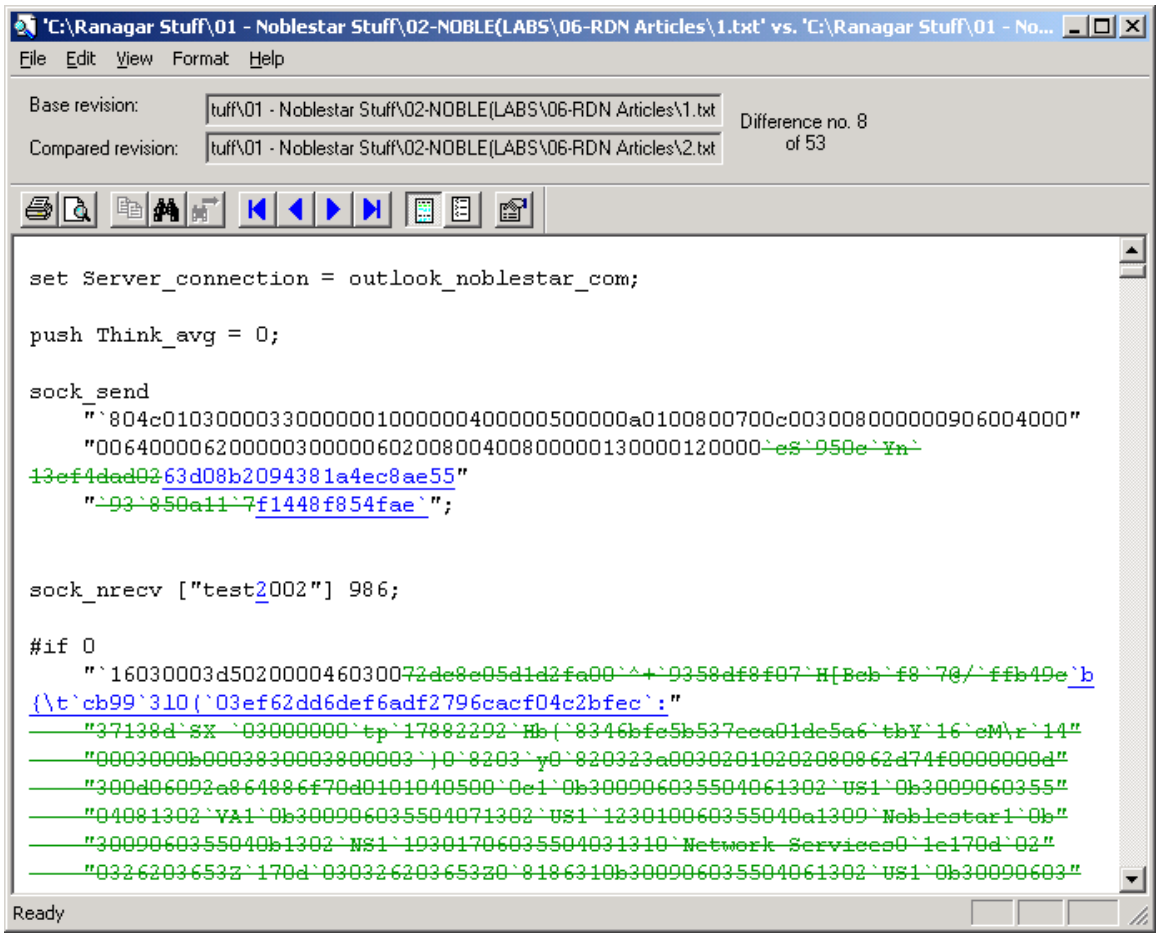


Figure 5: CSDiff screenshot

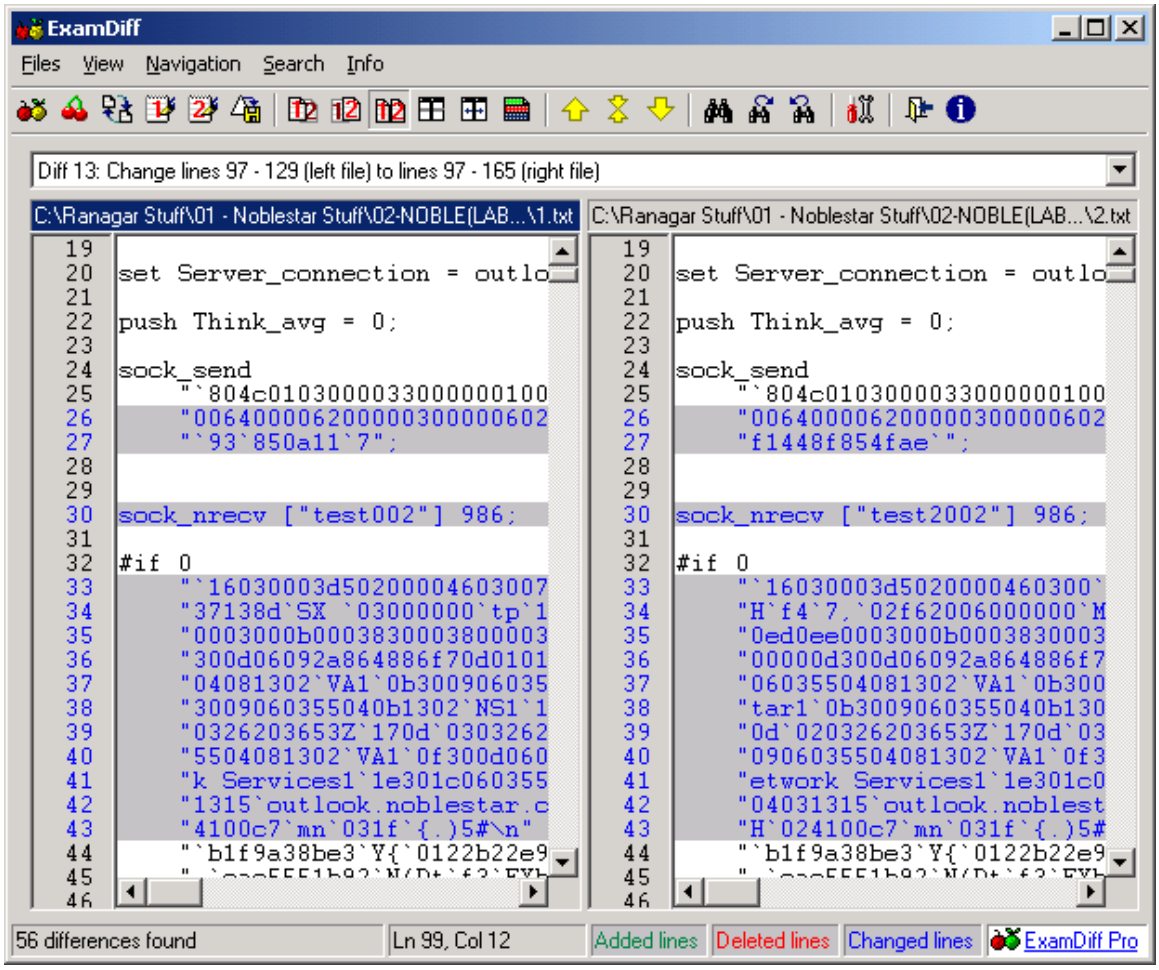


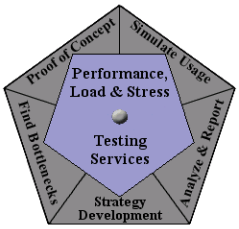
Figure 6: ExamDiff screenshot

One thing that makes this comparison process easier for known, or documented, protocols is the Request For Comment (RFC) for that protocol. Any protocol that isn't completely custom or proprietary will have an RFC on the Internet Engineering Task Force (IETF) home page http://www.ietf.org/. The RFC's will describe the exact specification for what characters and positions in the sends and receives represent.

Now You Try It

Every protocol varies so drastically that there really is no way to create a "typical" example or exercise. Added to that, I do not have access to any application that uses an unsupported protocol, either "clear text" or "encoded" that you would also have access to. The best ways to practice what we have discussed in this article are below.

For "clear text" type unsupported protocols, open an internet mail account (like Hotmail or Yahoo), record and edit scripts of sending mail to yourself. Ensure that you do not use more than a single user



during playback.

For “encoded” type unsupported protocols, record, edit and playback scripts against any SSL enabled website. If you use the settings above for recording, SSL recordings will look and act very much like an unsupported “encoded” protocol. Just make sure that if HTTP is one of your manual filtering options upon script generation, it is excluded.

Summing It Up

Just because a protocol is unsupported, does not mean that Rational TestStudio cannot be used for performance testing an application using that protocol. Proper configuration of recording settings and identifying subtle differences among recorded scripts can make a performance test that seemed to be impossible, completely accomplishable. While scripting unsupported protocols will likely take a little more time, I’m sure you will find that the extra time is justified when you find the first performance related defect.

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.