NTLM Authentication: Supported by OpenSTA… except when…

An Experience Report to the WOPR

By Antony Marcano

Abstract

This report describes my experience of investigating and solving a problem where OpenSTA scripts failed to complete during playback in a specific configuration. The application was built on Microsoft ASP, using NTLM authentication. OpenSTA supports NTLM, however, the organisation experiencing the problem could not get any scripts to complete successfully. I was hired to solve the problem.

This paper is primarily about a troubleshooting exercise in the context of a scripting challenge, rather than the typical issues surrounding load modelling, non-functional requirements or performance-testing results analysis.

This report follows my journey of discovery and learning about:

- NTLM, some other Microsoft technologies and their use in this particular configuration
- The performance costs that accompany NTLM
- Internet Explorer, how it works with NTLM and an apparent misconception

And particularly focuses on:

- How I investigated the issue
- The tools that I used in the investigation
- Limitations in OpenSTA, a bug and a feature constraint
- Why OpenSTA didn't instantly work with NTLM in this specific circumstance
- Lessons learned with the benefit of hindsight
- Learning points and perspectives taken away from the WOPR following my presentation of the content herein

By sharing this with you, I hope you gain something from my experience and from the perspectives of the Workshop in Performance and Reliability WOPR3 attendees.

Table of Contents

Introduction

A large company, was implementing a transactional workflow system that carried significant importance in their organisation. Regulatory compliance was facilitated by the application, requiring precise functionality.

The infrastructure

More significant to this story, this application, the System Under Test (SUT), was to be used globally, by most of the key revenue-generating personnel and would be accessed across their WAN and LANs. The production environment was a London based web-farm of approximately 5 IIS web-servers, load balanced using Windows Network Load Balancing Service (NLBS), a COM/application Tier, comprising two servers, both with a COM layer and one with a commercial workflow engine and finally, a cluster of 2 servers running Microsoft SQL Server, 1 active and 1 redundant. All of the servers used Windows 2000 Server. The test environment was of a similar configuration, except with two web-servers, one COM Tier and a single database server.

Invisible Authentication

The SUT, and indeed all of their intranet applications, were developed to use Windows Integrated Authentication using the NT Lan Manager protocol (NTLM). This was for their users' convenience so that they didn't need to enter log-on details in each of their different intranet applications. NTLM, combined with Internet Explorer and clients running Windows 2000 in this case, authenticates the user by sending a hash of user credentials to the server. The credentials used are those of the current user logged into the domain from the client PC. This happens behind the scenes – so from the user's perspective, they connect to the intranet application and it 'magically' knows who they are and has securely authenticated them.

Risk-motivated load-testing

The organisation, one of the largest in its sector, needed to ensure that the application initially supported an expected 300 concurrent (real) users at initial launch, which was to grow to a potential 4000 users, all using the system at varying parts of the day, however, 90% were expected to use the system at peak-times. This was the first time they had developed a system that was to be used so widely within the organisation, and due to this lack of experience, it was purported to be too risky to deliver the system without load testing.

Tool selection

Following a prohibitive quote from one of the main load-testing tools vendors, they were motivated to find a tool that would affordably meet their needs. They turned to using OpenSTA, an open source (free to use) load testing tool, to test the performance of the SUT when under load and stress. Previously, they had little or no experience with open source tools or load and stress testing.

They chose OpenSTA as a load testing tool because, after a review of its capabilities 'on paper', it met their load and stress testing requirements. It provided similar functionality to the commercial alternatives, without the associated licensing costs.

Some high-level performance requirements had been defined and an in-house tester and developer had started trying to develop some scripts.

The brick-wall
Their hopes were dashed when none of their attempts to replay scripts were successful. The scripts would execute, but at unpredictable points in the script, authentication would fail (401 'UNAUTHORIZED' response code instead of a 200 'OK' response). Neither the tester or developer had the knowledge or experience to get past this point.

They were convinced that OpenSTA was not working with NTLM and hired me to confirm, one way or the other, whether OpenSTA was up to the challenge.

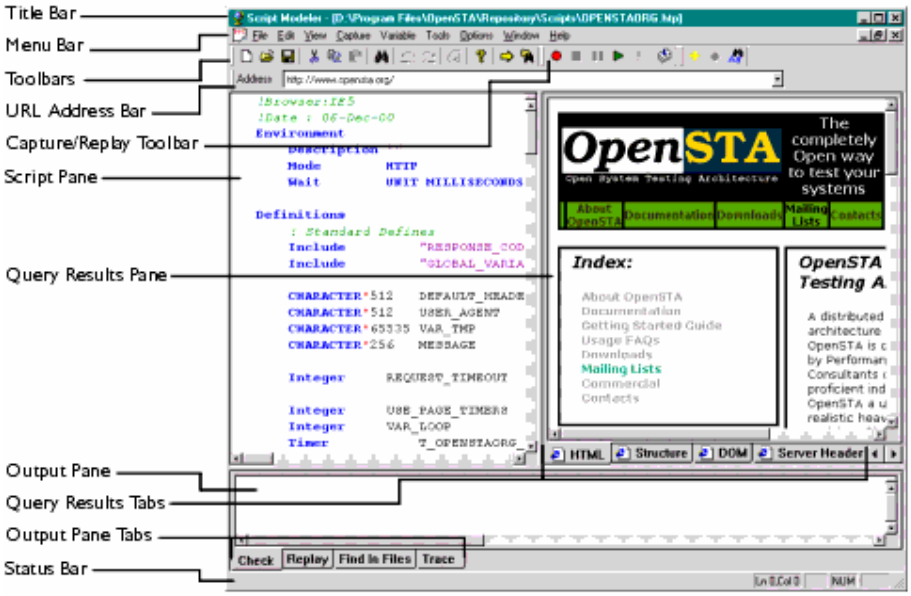
Unique challenge of this experience
Perhaps unusually, this experience report is about a scripting challenge. The details of the load-profiling, the load testing itself, and user-profile analysis, at most, are touched upon and aren't significant to the unique challenges that I associate with this particular experience.

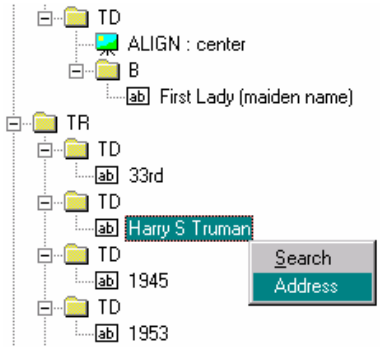
About…

…OpenSTA

"*OpenSTA is a distributed software testing architecture designed around CORBA, it was originally developed to be commercial software by CYRANO. The current toolset has the capability of performing scripted HTTP and HTTPS heavy load tests with performance measurements from Win32 platforms. However, the architectural design means it could be capable of much more.*" – www.opensta.org

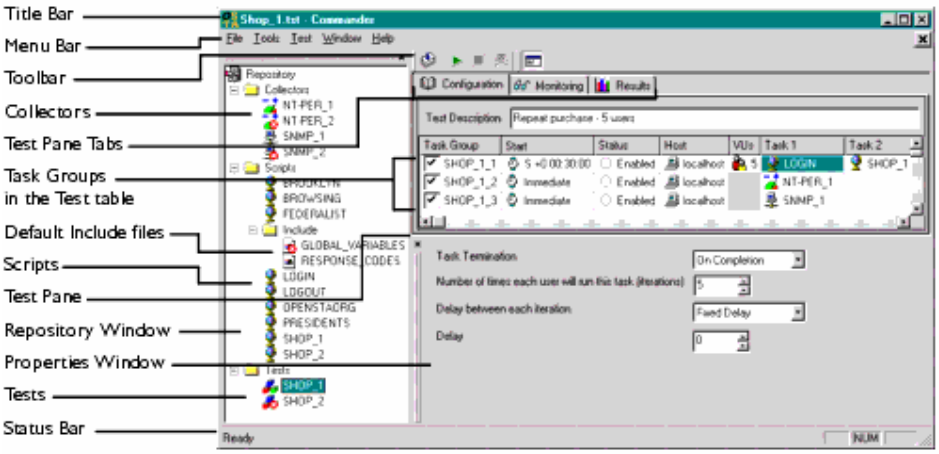
It uses an IDE called the Script Modeller where script development in Script Control Language (SCL) is carried out. Record and playback, using the chosen web-browser via the OpenSTA proxy can be used to facilitate script development.





Other powerful tools are also found in the modeller, including various views of the request and returned document. Particularly useful is a graphical view of the Document Object Model (DOM) with automatic code generation for reading in values from specific DOM addresses.

The commander is where design and control of the test is carried out. Structuring of scripts, timing and ramp-up of users and 'collectors' that retrieve statistical data from NT and/OR SNMP performance counters.



Images courtesy of www.opensta.org

…Windows Integrated Authentication using NTLM

"*Microsoft Internet Information Server (IIS) supports Microsoft Windows NT Challenge/Response authentication that a user identifies without requiring the transmission of actual passwords or account information across a network.*

However, the Web server authenticates users by carrying out a procedure that challenges the user's Web browser to carry out a specific mathematical computation involving the password and to respond by returning the results of the computation to IIS. IIS then duplicates this computation using the user's account password information, and compares this result to the user's result. If both results match, IIS recognizes that the user has the correct password and grants access." – from http://support.microsoft.com/default.aspx?scid=kb;en-us;174775

A good overview of NTLM authentication and HTTP can be found here for further reading: http://davenport.sourceforge.net/ntlm.html#ntlmHttpAuthentication

More information follows later in this paper.

The Symptoms

On my first day, all I had to go on was the information that the scripts were executing only past the first few requests and failing. I was given an internal URL for the application and a workstation with OpenSTA installed and a copy of their scripts so far.

The scripts that had been developed so far did fail due to connections that were being randomly rejected, despite being previously authenticated. I determined this by executing one of their existing scripts in single-user mode from the Modeller.

Each of the requests are shown, as they are executed, in the replay frame in the lower-half of the Modeller IDE, along with response codes and other information such as the content of new cookies. I inserted WAITs of 5 seconds after each request so that I could easily follow the execution of the script.

Requests were then failing from the 2nd or 3rd HTTP request (apparently arbitrarily) each time the script was played back. This wasn't the 2nd or 3rd page request, this was the 2nd or 3rd resource request (after the HTML had been retrieved).

Making OpenSTA Work in this Situation

This was my first time having to concern myself with NTLM – so the first thing I did was do some reading… Searching on Microsoft.com and with Google, I found several useful resources explaining NTLM (in intimate detail in some cases).

NTLM illustrated

It was all well and good reading about NTLM but I often find it's easier if I can see something in action. Since Internet Explorer happily maintained authentication, I used it to access the SUT while sniffing the HTTP on the wire using Ethereal (www.ethereal.com), although these days I tend to use Packetyzer (www.packetyzer.com).

For those who aren't familiar with standard HTTP, without NTLM, this illustrates what happens when the browser requests a single resource, such as an HTML file:

Client Port	*Message*	*Server HTTP Port*
1110 (new connection)	GET /file.html →	"
1110 ←	**[file.html]**	"

When NTLM is used, this is what happens…

Client Port	*Message*	*Server HTTP Port*
1110 (new connection)	GET /file.html →	"
1110 ←	401 Unauthorized	"
1111 (new connection)	GET /file.html (NTLM NEGOTIATE) →	"
1111 ←	401 Unauthorized(NTLM CHALLENGE)	"
1111	GET /file.html(NTLM AUTH) →	"
1111 ←	**[file.html]**	"

The Negotiate message contains a hash created from the username, password and domain. The Challenge is a server-generated key based on the user's domain credentials. The client must then use this to generate a valid response-hash which is the authorization code and return it on the same connection. If this matches what the server expects, then the server fulfils the request.

From this point forward, the connection can be reused for subsequent requests with the Authorization code in the HTTP Request Header.

Now at this point, I saw no reason why OpenSTA shouldn't handle this. One of the client's developers explained that the server randomly re-authenticates connections which, he explained, will become evident as the script sends subsequent requests. This was echoed in a report from an independent consultancy that had recently performed a performance-review of the client's intranet systems.

The decision to script around it

The client was under pressure to complete some load testing and was convinced that OpenSTA wasn't working correctly.

Rather than take the risk and allow me to investigate further, they preferred that I try to script around the issue… so this is what I did…

SCL and Dynamic Connections

The client's developer, mentioned previously, had started a subroutine that checked the response code from the server and if it was a 401 (unauthorized) it passed control to another subroutine that followed the NTLM Challenge/Response protocol with that URL.

This also didn't work, but it wasn't far off. The problem was that OpenSTA doesn't manage dynamic connections without some scripted intervention.

In an SCL script, each connection has a numeric ID, starting at 1. OpenSTA will not issue the next request on a given connection ID until the previous request on that connection is completed. Concurrent requests for a single VU must be given different connection IDs.

If a connection is closed by the server, a new connection must be opened for the next request. When the server returns a 401 response code, Internet Explorer closes the current connection and opens a new one, sending an initial NTLM Negotiate request.

If this happens during test execution, unless told to do so in the script, OpenSTA will not behave in the same way.

So, the recoded connection IDs had to be replaced with dynamic connection IDs (integer variable). In order to simulate concurrent connections and to maintain recorded browser-concurrency, I used an array to hold the connection IDs that were in use at a given time. The recorded connection ID was replaced with the appropriate reference to the position in the array, relevant to that request.

In this environment, it was only necessary to have 2 concurrent connections:

```
PRIMARY GET URI "http://" + App_Server + "/index.htm HTTP/1.1" ON iConnID[1]
…
GET URI "http://" + App_Server + "/styles.css HTTP/1.1" ON iConnID[2]
```

So, for each request, it was necessary to capture the response code by appending 'RETURNING CODE' to the request:

```
, RETURNING CODE iResponse
```

And then pass that to the Status Checker with the connection ID, position in the array, type of request (POST, GET, PRIMARY GET), the URL and finally the referrer:

```
Call CheckStatus[iResponse, iConnID[1], 1, 1, "/index.asp", ""]
```

If the response was a 401, the Status Checker passed control to the Authenticator subroutine.

Further complicating this, there is an OpenSTA bug (645735) that prevented reading the content of the returned page when the connection ID was an integer variable. This is necessary to ensure that the transaction had completed successfully. Fortunately, this bug had a work-around.

Additionally, more complexity resulted from the fact that Internet Explorer closes any active connections if they haven't been used for more than 60 seconds. If WAITs were longer than this, the scripts also had to close the connections and open new ones resulting in further re-authentications. A WAIT routine was created that closed any active connections after 60 seconds and incremented the connection ID – thus forcing re-authentication on the next request.

Authenticator Overview

The authenticator required approximately 130 lines of code to implement, handling different HEADER and REQUEST types as well as session IDs.

Part of the initialisation for a script was to establish the user's credentials. This was read from a file listing a number of real NT Domain Test Users. These could then be used in the Authenticator.

The Authenticator would build the header for the request depending on the type of request (PRIMARY GET, GET or POST) and construct the user-credentials data that had to be sent to the server:

```
BUILD AUTHENTICATION BLOB  &
  FOR NEGOTIATE    &
  FROM USER strUserName PASSWORD strUserPass DOMAIN strDomain    &
  INTO strAuthBlob
```

It would also need to drop the connection that was UNAUTHORIZED and create a new one (maintaining the array):

```
DISCONNECT FROM iConnection
Set iConnection = iConnection + 1
Set iConnId[iConnIDPosition] = iConnection
```

The Challenge response was then handled:

```
PRIMARY GET URI "http://" + APP_Server + Auth_URL + " HTTP/1.1" ON iConnection &
     HEADER DEFAULT_HEADERS               &
     ,WITH {WITH_HEADERS,            &
     "Authorization: "+strAuthBlob}

     Load Response_Info Header on iConnection            &
          Into strAuthBlob  &
          ,WITH "WWW-Authenticate"

BUILD AUTHENTICATION BLOB  &
     FOR NEGOTIATE  &
     FROM BLOB strAuthBlob &
     INTO strAuthBlob

PRIMARY GET URI "http://" + APP_Server + Auth_URL + " HTTP/1.1" ON iConnection  &
     HEADER DEFAULT_HEADERS               &
          ,WITH {WITH_HEADERS,            &
          "Authorization: "+strAuthBlob}          &
          ,RETURNING CODE iResponse
```

The resulting status was then checked (logging any errors) and then control returned to the calling script if the authentication was successful.

I mentioned earlier that the Authenticator also had to manage Session IDs. The developer that had made a start on this had already written the additional session ID code. I just made it work. It wasn't until this point that it occurred to me - *Why are they using session IDs if each HTTP connection is authenticated using NTLM AND the identity of the user was determined from the NT user-name?*

After putting this question to the lead developer, I was advised that he was 'pretty sure' that they weren't using ASP Session IDs at all!

This is significant, because IIS uses more memory and CPU resource to generate Session IDs than if they are disabled (Session State Value in the IIS configuraiton)[1]. This also affect the overhead on the server with each request.

Despite these assurances, I didn't want to change the environment at this stage as it would have presented too great an impact on the schedule if they were wrong.

I added a test-property in my global_variables.inc file SessionState which allowed me to switch session handling on or off in my scripts.

The IE "Connection Swap"

The network traces in Ethereal, showed other instances of re-authentication of connections, except this time cause by the browser. This is what I call the IE Connection Swap.

This is a strange quirk that I came across as I was doing more network traces during script development. It may or may not be specific to Internet Explorer but was nonetheless an interesting oddity and worth explaining. It is worth explaining so that you see why I thought its impact was insufficient to justify the effort of developing the script-code – and because you may find it useful to know.

Consider that you are at the beginning of requesting a page and your browser is only going to use 2 concurrent connections to collect the objects required to render the page. The first connection is opened to collect the HTML file, and this is what happens:

Client Port		Message		Server HTTP Port
1110 (new connection)		GET /file.html	→	"
1110	←	401 Unauthorized		"
1111 (new connection)		GET /file.html (NTLM NEGOTIATE)	→	"
1111	←	401 Unauthorized (NTLM CHALLENGE)		"
1111		GET /file.html (NTLM AUTH)	→	"
1111	←	**[file.html]**		"

Now, file.html is parsed and let's say there are two more objects to be retrieved, for example nav.js and search.gif. The connection on 1111 is already authenticated, so it requests nav.js on the same connection, and the browser then decides to open a new connection to retrieve search.gif:

Client Port		Message		Server HTTP Port
1111		GET /nav.js (NTLM AUTH)	→	"
1112		GET /search.gif	→	"
1112	←	401 Unauthorized		"

At this point, port 1111 has had its request satisfied:

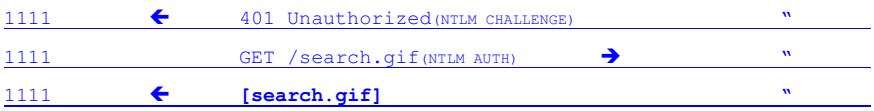
1111	←	**[nav.js]**		"

Based on the earlier pattern when requesting file.html, we would expect IE to open a new port, say 1113 to send the NEGOTIATE message for search.gif (previously requested on port 1112). But because port 1111 is now free, IE decides to send the NEGOTIATE packet on port 1111:

1111		GET /search.gif (NTLM NEGOTIATE)	→	"

[1] http://support.microsoft.com/default.aspx?scid=kb;en-us;244465

Note that IE sends the full NEGOTIATE message and not the AUTH code in the header. So, the server has no choice – operating in a stateless fashion, it responds with the challenge and the connection is re-authenticated, quite unnecessarily:



Being stateless, this behaviour makes sense, however, the connection itself isn't really stateless. It is authenticated or not-authenticated, so it seems strange that IE would use an authenticated connection as if it wasn't. I haven't investigated this further – but I will in due course and perhaps I will find that it works as specified or perhaps it is a bug.

At this stage, however, it was clearly a distraction and had no impact on OpentSTA other than the connection sequences that would be reflected in the script.

Modularity limitations

I have always tried to make my scripts as reusable as possible, separating them into concise sequences of user-interactions for re-use. OpenSTA supports this superbly in its drag and drop user interface for defining Task-Groups.

Unfortunately, on completion of a script, OpenSTA closes any connections in use. This means that when the next script in the task group takes over, any authenticated connections are no longer valid.

There is no exit-without-close option so this would mean that at the start of the next script-module, or 'Task' in the Task-Group, all connection IDs would be treated as new connections.

This was something of a concern since, multiplied out by 300 concurrent users (initial usage estimate) across 4 Task Groups and 4-6 scripts per Task Group; this could amount to approximately 4,800 unnecessary requests per task-group iteration, which happened on average 3-4 times over a 40 minute load-test:

> *300 users / 4 task groups = 75 users per task group (on average)*
>
> *4 re-authentications * 4 scripts * 75 users = 1200*
>
> *1200 additional requests * 4 task groups = 4800*
>
> *4800 requests * 3 = 14,400 (approximately) additional requests per test-run*

If this was just for the initial usage estimate, subsequent tests, including a stress test of up-to 4000 concurrent users, would be even worse… so I decided that it was necessary to remove these unnecessary re-authentications.

I would have preferred to have the issue addressed by having a exit-script-without-closing-connections feature in OpenSTA, however, the client insisted that we avoid this.

A simple work-around was to ensure that the modularity of the scripts corresponded to stages in the user's activity where each script module was completed just before a WAIT that was longer than the IE 60 second disconnect. This way, it wouldn't matter that the script had disconnected, because that would have happened anyway.

This constrained the modular design of the scripts significantly. The application under test was a workflow application with several pages that were closed questions (e.g. yes/no; option a/b/c..), completed by the user in under 60

seconds. Also, the workflow reused certain sequences of ASP pages that could be reached through various paths in the workflow. To optimise re-use of the scripts, the modularity of the scripts should, ideally, reflect the modularity of the workflow stages allowing for accurate modelling with minimal scripting effort.

After some analysis, I found that it wasn't possible to structure the scripts in such a way that I could meet both objectives (1. avoid unnecessary disconnects; 2. optimise re-use).

So, an alternative workaround was chosen. Each script was moved into a set of subroutines that could be called in sequence from a single script, using the 'Include' capability of the scripting language.

This was not ideal and lost the efficiency of simple drag & drop Task Group construction but met both objectives without a huge overhead.

Running the tests

After writing the subroutines and the scripts along with all of the manual edits required for each script to hook into these subroutines, 3-4 weeks had passed. Because the pressure was on building the scripts, I had not looked at the environment I was using in any detail up to this point. I felt very uncomfortable about this but it was the only way to have the scripts ready in time for the tests to be run.

I was aware that the servers that I was testing against were load balanced using the Microsoft Network Load Balancing Service (NLBS) and that in my test environment I had two web-servers and the soon-to-be production environment (the one I was to execute the tests on) was to have 5. I had multiple load-injectors at the ready.

By this time, I had an intimate understanding of NTLM, I was aware that the way the System Under Test was behaving was different to the specification – but I hadn't had a chance to investigate why.

My first tests were simple benchmarking tests. I executed them against each individual server and then via the load-balanced URL.

Instantly, I found my first real clue as to why OpenSTA wasn't happy using NTLM in this scenario...

The Summary Snapshots report (and my custom Test Report) in OpenSTA showed me that there were far fewer 401 status codes returned when running the same test against the single server, as opposed to the load-balanced URL. This was despite the tests being identical!

Immediately I knew there was an issue with the load-balancing service. I investigated the options available on Microsoft's knowledge base, one of which included:

> Q.How Does Single Affinity Mode Differ From No Affinity Mode? Which One Should I Use to Load Balance My Application?
>
> A. In *Single Affinity* mode, NLB load balances traffic based only on the Source IP Address of the incoming connection. So, *Single Affinity* mode ensures that all TCP connections originating from the same client (IP Address) are sent to the same host in the cluster. This will continue indefinitely until a host is either added or removed from the cluster, at which time the connections originating from that IP address may get mapped to a different host in the cluster.
>
> In *No Affinity* mode, NLB load balances traffic based on Source IP Address and Source Port of the incoming connection request. Therefore, in *No Affinity* mode, multiple connections from the same client may be handled by different hosts in the cluster as long as these connections have different source ports.

> Which mode to use really depends on the application being load balanced. If the application makes use of sessions which persist over multiple TCP connections, NLB should be configured in *Single Affinity* mode because you want to make sure that all TCP connections which are part of a single session are mapped to the same host in the cluster.
>
> On the other hand, *No Affinity* allows a better load distribution because it does not require client connections be handled by specific servers. Thus for applications that do not use sessions and for which it is acceptable for multiple incoming connections originating from the same client to be handled by different hosts in the cluster, *No Affinity* would be a better mode of operation.
>
> From: http://www.microsoft.com/technet/prodtechnol/windowsserver2003/technologies/clustering/nlbfaq.mspx

I then checked the server configuration, which was indeed set to "NONE". After some investigation, it was found that the operations team had not followed their documented procedure when configuring the Load Balancing service, as their default configuration was "Single" (contrary to Microsoft's recommendation).

I recorded a simple script that would play back without any script updates. First, I played it back on one server with a single VU, then against the load-balanced server. It failed on the load-balanced server as before.

We changed the setting to Single, and I played back the script again… and it worked… no dynamic connection management or special Authenticator sub-routines. I then repeated one of my previous tests and indeed the number of 401s reduced by over a third.

This was confusing since the definition (above) states that even if the Affinity is set to "None", individual connections should be routed to the same server, although multiple connections may be distributed across several. So, this shouldn't have made a difference… but the reality is that it did!

Since we were at a relatively safe stage now, we also disabled session state and repeated the tests. A small but noticeable improvement in performance was observed and no application errors were evident – so we could safely leave it disabled (after a little functional regression testing of course).

Finally, all images and script files that didn't contain sensitive information were opened up to anonymous access… since, prior to this point, even images could not be accessed without the NTLM overheads.

These changes accounted for approximately 15% improvement in performance, when tested on the LAN; however, this would have made all the difference since the system was to be accessed from numerous worldwide offices in Europe, Asia and the US via a WAN.

The additional latency of the WAN, combined with the number of round-trips (due to NTLM) for each object requested, this would have made a more noticeable difference to page performance for users in the more remote locations.

It would have been a nice ending to quantify this by using a WAN-latency emulator. The client had their own WAN-latency emulator, but it would have taken time to gather latency measurements from around the WAN and configure these parameters in the emulator. It was decided by the client that this would not deliver sufficient value, and they were content with the results, as they stood at that time.

A side-issue…

Development was at a crucial stage and no application tuning was possible at this point. The tests confirmed that performance was 'acceptable'. There was one more issue, however, that added a health-warning to the release…

After a test, using my browser, I accessed a task list in the application (which would get rather large). I found that the page wasn't loading, despite not having any errors or unusually high response times pertaining to it in the OpenSTA test results.

I used Ethereal to examine the HTTP and found that the page was being returned very quickly but wasn't being rendered. Using ethereal to look at the HTML, I noticed that there was a javascript function that was executed when the page loaded. It was sorting the task list… preventing the page from being displayed until it was finished. I don't know why the developers decided to sort the list on the client-side with scripting, rather than let SQL Server do it for them – but they did…. What was killing the page was the 3 minute meta-refresh.

Switching off meta-refresh allowed me to find that the page was taking 6-7 minutes to complete re-sorting the data (using a bubble-sort algorithm I might add). This was much longer than the refresh, so before the page was even able to finish executing the sort function, it was refreshing again, never actually being able to render itself.

This was a good example of when data volume impacted user-perceived performance, but because this was client-side performance, OpenSTA was unable to detect it (or even facilitate its detection).

The Conclusion

1. *The mythical random re-authentication* – All the local experts around me were insistent that there was an intentional server-driven random re-authentication involved in NTLM. This came from the technical lead for the client, the lead developer and from an expensive consultancy that was doing a general review of performance on their WAN.

   It was the real server-driven random re-authentication that left them convinced that OpenSTA didn't work. This was why they were pretty sure that OpenSTA couldn't work with NTLM. This was a myth, however. There were two causes for connection re-authentication in their specific configuration:

   a. Microsoft Network Load Balancing Service was distributing load on a per request basis. Thus, a request that was sent with valid authorisation for one server, was being distributed to another server – and thus that connection was, as far as the second server was concerned, was unauthenticated. So, a 401 response was returned and the connection re-authenticated. This was not part of the NTLM security protocol – just a coincidental by-product of NTLM AND their NLBS configuration.

   b. The IE Connection Swap characteristic resulted in re-authentications; however, this appears to be unintentional behaviour as a result of the stateless characteristics of connection re-use. This had no impact on the load testing scripts – OpenSTA recorded these and played them back quite happily.

2. Client-side performance isn't something that OpenSTA (or indeed many Load testing tools) allow you to measure or observe in any detail (although I have used this capability in a version of E-LOAD). Obviously, if you have a load injector emulating 100s of users, it is unlikely that you can get any indication of client-side performance. It would have been useful if one of the load injectors could have been dedicated to one VU and have a visual playback of the script – in a browser. The never-ending sort function could have been spotted sooner. There is a third-party (free) module that does provide this functionality in the modeller to facilitate

debugging. Perhaps this might be integrated into the commander as an option at a later date.

3. With hind-sight, 2 or more weeks work could have been avoided with a little more investigation – but then the investigation required in that two weeks of effort helped find the issue that may have otherwise been ignored if the tool had handled connections and NTLM transparently. This would depend on the experience of the Tester. I know that this Tester will not overlook it, regardless of how a tool might hide such issues.
4. The testing created several additional opportunities for database tuning, application tier optimisation and client-side-scripting improvements. Only the NLBS and NTLM tuning were taken up in that release, however, much of the application tier was refactored as a result over the next two releases. It still performed acceptably following the initial test and is continuing to perform well in the wild today.

Lessons Learned

These are the lessons that I feel I have learned from this… there is nothing especially profound, but you may take something away from my experience.

In General

1. Become intimate with your protocol(s) whenever practical. Obtaining a detailed understanding of the protocol made a big difference in identifying the issues in this experience.
2. If possible, develop & test the scripts against both a single web-server and a configuration that is comparable (but not necessarily scaled) to the environment you are to run your tests on.

   If the client had tried to develop the scripts and test them against a single server they would not have known about any potential issues until they tried to run their first real test. That would have left much less time to do anything about it. In this case, the test-development environment was adequate to develop the scripts against, having a load balanced configuration across two relatively low-spec servers.
3. Any connection specific authentication (such as NTLM) combined with round-robin (per request) load balancing rules are not a good mix as this can nearly double the number of round-trips required to satisfy each HTTP request. Without NTLM, this is perfectly acceptable; with NTLM enabled, it is something you really want to avoid. Connections should retain an affinity with a specific server. If it is not possible to have TCP/IP connection affinity then use IP affinity rules. If the server provides SSL services, then this will be necessary anyway. Despite this, alternatives should be considered carefully, since Session objects mapped to the HTTP session using Session IDs consume valuable server resources.
4. Client-side performance should not be ignored just because the tool you happen to be using doesn't provide features to measure it. It would be beneficial if the tool does, but if not, then other means should be used to test from the client. The down side to this is that there will be more scripting to do, since these functional scripts may be very different to your load testing scripts.
5. Respect the knowledge of local experts – but remain wary (and verify everything whenever possible):

   When called in as a trouble shooter, you are often given access to local experts who may have been trying to solve the issue for a while before

your arrival. They usually share their investigations so far, what they found and what they didn't find. It is tempting to take their word as correct, but sometimes you just have to see it for yourself… and follow your own path… skipping nothing, even if they say "I've tried that" (which is so tempting to do when there is pressure applied to get the job done ASAP).

It may be that their knowledge in a specific area is based on their research (which may not be as thorough as your own); They may have been under so much pressure that they overlooked the obvious… whatever the reason – whenever possible, verify what you are told independently.

For some local experts, inaccuracies in what they know is just a simple case of the individual having the wrong information… for others, perhaps subconsciously, they want you to fail – at least for a while. If you solve the problem too quickly – perhaps they will lose face?

In this case, I believe that the issue was propagation of invalid knowledge – with no conscious or subconscious intent. The various local experts insisted that NTLM, as standard, involved a server-initiated re-authentication when, after my own investigations, I found not to be true. This was further exemplified by the fact that the operations team, who had configured the test environment, insisted that the load balancing rules were configured correctly when they were not. Only after insisting that they check the rules, were my suspicions confirmed…

I should have learned this lesson on an earlier project. A totally different project for a totally different client, in that instance, a J2EE Portal system. I was trying to diagnose a resource leakage in their live environment. I was given limited access to a replica environment which, their IT rep assured me was identical. I spent some time investigating their issue but I couldn't even reproduce it! Only when I insisted that he 'humour me' and allow me to observe every step of how he verified that the environments was identical did I find that the versions of software were not the same. The live system had a slightly more recent version. By using the same version – the problem was reproduced immediately - I then isolated the issue to incorrect connection-pool management resulting from a specific class – this was fixed and their problems went away… I took the word of a local-expert in that instance and it caused unnecessary delay. Once bitten, twice shy? I hope so.

6. Sometimes you have to do what the client wants you to do… This isn't really a new lesson but I want to include it here as a reality check. I was convinced that there was an issue that we could have investigated and perhaps avoid some scripting effort and halved the time required to deliver the system. With all the best will in the world, sometimes, you have to accept the choices of the client. It is their business after-all. The risks impact them more than anyone else. All you can do is advise them of the risks, arm them with the appropriate information to make a decision – but ultimately it is their decision as to how to proceed. I may not always agree… I may note my disagreement – but I will always give the customer the opportunity to learn from their own mistakes.

OpenSTA Specific

7. OpenSTA, in my opinion, continues to be a worthy adversary of the commercial tools. With the level of support from the community that it deserves, it has the potential to make load and performance testing accessible to many organisations that would never have been able to justify the budget to purchase a commercial tool.
8. Something that appeared to be a limitation in OpenSTA, found an important performance tuning issue – again! Commercial tools make scripting a little easier – in this case by managing the connections behind the scenes - but I have always admitted that OpenSTA isn't the most productive tool when it comes to scripting. So far, however, I have found only one case where it wasn't up to the task at hand and that was a well known limitation with OpenSTA not handling compressed/Gzipped HTML.

   In every other scenario that I can recall – when an aspect of the technology starts to get difficult to script around - it has preceded a discovery about a significant performance issue in the System Under Test.

   I also haven't experienced a scenario where the additional scripting overheads justified the capital outlay for commercial tools licenses.
9. Yes – OpenSTA has bugs! Doesn't everything? The two issues that impacted progress only had a small impact. Thankfully, the openness of bugs and workarounds resulted in this having a minimal impact. Knowledge of bugs in open source tools, like OpenSTA, are very openly disclosed and shared… more so than with commercial tools in my experience. Especially since you get to see the comprehensive and official known issues list before you adopt the tool.

   A weird bug in OpenSTA required a simple work-around

   http://sourceforge.net/tracker/?group_id=10857&atid=110857&func=detail&aid=645735 (note the behaviour depending on the context of the INTEGER variable)
10. Coding around dynamically authenticated connections and using modular scripts highlighted the fact that connections close on EXIT of a script (http://sourceforge.net/tracker/?group_id=10857&atid=360857&func=detail&aid=790516). I didn't see this as a bug but having the ability to override it is certainly worthy of a feature request.
11. OpenSTA doesn't have built in support for dynamic connection management. This would be beneficial – however, I wouldn't want to completely loose control over this feature. Overriding it where appropriate would be beneficial.

Microsoft Windows Specific

12. NLBS appears to distribute load across each server on a per-request basis and not per connection when affinity is set to "None" and thus doesn't mix well with NTLM connection-specific authentication.
13. NTLM authentication offers various user benefits but should only be implemented after assessing the pros & cons and ensuring that the infrastructure is appropriately configured.

Personal lesson(s)

14. Always be mindful of keeping an open mind and remaining objective! Don't let other's preconceptions influence my judgement!

    There were times that I felt that maybe they were right – maybe OpenSTA just couldn't do it. It was frequently stated, by the client's representatives, that they "couldn't get OpenSTA to work" and had spent over two months trying to do so.

    This was a reasonable deduction, from their point of view since lack of information and knowledge in OpenSTA meant that they rated it as more likely to be at the root of the problem, rather than their own environment, which they felt they knew a lot about.

    This is reasonable, since risk is higher when information is lower (exactly what happens when a system is going live – if we haven't tested certain areas, risk is considered to be higher in those areas – again just lack of information). In their case, however, it was significantly influenced by their preconceptions about open source software being 'buggy' – partly because it is free – so 'something must be wrong with it'. I think that if OpenSTA had a development-tool price tag and a big brand behind it, they might not have been so quick to reach their conclusion.

    I took over trying to get 'OpenSTA to work'… and wasted time as a result… I should have stayed open minded and perhaps been more assertive about exploring why their environment AND OpenSTA weren't working together. I had learned this lesson previously when testing a .NET application that used ViewState – but obviously I hadn't learned it well enough!

Lessons and Perspective from WOPR Attendees

Many of the lessons listed above were reiterated by the attendees at the WOPR. There were few alternatives to the approach, except that having a better insight into the infrastructure may have identified the issue before investing time in developing complex scripts.

The lesson that OpenSTA is a competitor to commercial tools was also underlined, in my opinion.

Ed Glas, of Microsoft showed particular interest in the IE Connection Swap, which we both suspect might be a bug.

Most of all, I learned that I was not alone! Documentation about the environment wasn't available... The client sat on my shoulder, constantly reminding me of how "OpenSTA" wasn't working and pushing me down the path of investigation that they wanted me to take... finding that understanding the protocol is essential… these are issues that many other performance testing consultants can clearly identify with.

About Antony Marcano

Antony Marcano is a London (UK) based software engineering consultant, specialising in software testing. He has been a practitioner of Software Testing since 1995 and in performance and load testing since 1999. His load testing tools experience includes E-Load, WebLoad, LoadRunner and, more than anything else, OpenSTA (www.opensta.org). Antony co-founded etest associates (www.etest-associates.com), a small UK based consultancy that provided OpenSTA support and consultancy. More recently, he created testingReflections.com, a blog-based knowledge-share for all types of software testing.

Further to his experience in Performance, load and stress testing, he also specialises in working in agile development environments. First involved in an Extreme Programming project in 2000, he is now building on his experience in using FitNesse (extension of the FIT framework) for developing test-first automated acceptance & system tests.

Antony's thoughts on specific software testing related issues have been quoted by Corporate Insurance & Risk magazine and VNUNet. He has presented on Performance Testing with OpenSTA at a British Computer Society - Specialist Interest Group in Software Testing (SIGIST) conference and has had Security Testing related article published in the SIGIST journal: "the Tester".

His clients have included Freshfields Bruckhause Deringer, College of Law, Equant, Barclays Bank, Financial Times, Vodafone and British Telecom.

Workshop on Performance & Reliability 3 (WOPR3)

This paper was presented to the Third Workshop On Performance and Reliability (WOPR3) on 10th September 2004. The attendees of this workshop were (in no particular order):

Ross Collard; Paul Holland; Chris Johnson; Dawn Hayes; Sean Hull; Ed Hill; Henry Amistadi; Karen Johnson; Mike Kelly; Eric Proegler Roland Stens; Rob Sabourn; Richard Leeke; Antony Marcano; Ed Glas; Bob Sklar; Julian Harty; John McConda; Bret Pettichord.

www.performanceworkshop.org

Copyright notice