Intermediate iRules: Validating Your Logic

Jason Rahm, 2016-07-06

Sometimes an iRule will load and run without producing any errors, but does not achieve the desired results.

In this article, I will outline the basic process of examining your iRule logic against live traffic by walking through a simple example: An iRule intended to perform bi-directional HTTP Host header modifications.

I recently helped a customer who had a fairly common configuration: The webservers are configured to respond to a different domain name than the one in the URLs the customer advertises for the virtual server: The "correct" host header containing the internal server name is required for the webservers to process a request. In addition, any server responses which contain the internal hostname need to be modified to reflect the publicly advertised name instead.

These are the host names we will use for this example:

Advertised name: easyname.domain.com

Real/internal name: long.internal.name.domain.com

For all HTTP requests, the hostname "easyname.domain.com" must be translated to "long.internal.name.domain.com". For all HTTP responses, the hostname "long.internal.name.domain.com" must be translated to "easyname.domain.com".

Here is the initial iRule the customer created:

```
when HTTP_REQUEST {
 if { [HTTP::host] equals "easyname.domain.com"} {
 HTTP::header replace Location \
[string map -nocase {easyname.domain.com long.internal.name.domain.com} [HTTP::header value Location]]
 }
}

when HTTP_RESPONSE {
 if { [HTTP::host] equals "long.internal.name.domain.com"} {
 HTTP::header replace Location \
[string map -nocase {long.internal.name.domain.com easyname.domain.com} [HTTP::header value Location]]
 }
}
```

When traffic was run against the rule, no errors were seen in the logs and traffic flowed normally between client and server, but the intended replacements were not performed.

Given such unexpected behaviour, the first step in validating your logic is to ensure that the information your iRule sees and acts upon is what you expected it to see and act upon. To do so, you can add some logging around conditional decisions. The best practice is to first log the value of the variables, objects, or commands used to make the decision just prior to each decision point, and then log another message after each decision point to indicate the expected code block is indeed executing.

Here is the customer's rule modified to include some informational logging around the conditions:

```
when HTTP_REQUEST {
 # First we'll log the 2 header values used in the conditional code block
 log local0. "Host = [HTTP::host]"
 log local0. "Location = [HTTP::header Location]"

 if { [HTTP::host] equals "easyname.domain.com"} {
```

```
 # inside the conditional block, add another log line saying that's where you are
 log local0. "Host matched, performing replacement operation"
 HTTP::header replace Location \
"[string map -nocase {easyname.domain.com long.internal.name.domain.com} [HTTP::header value Location

 # you can even log the result of the replacement operation by running it again with the log command
 log local0. "Replacement text = \
[string map -nocase {easyname.domain.com long.internal.name.domain.com} [HTTP::header value Location]
 }
}

when HTTP_RESPONSE {
 # For the response, we'll again log the header values used in the conditional code block
 log local0. "Host = [HTTP::host]"
 log local0. "Location = [HTTP::header Location]"

 if { [HTTP::host] equals "long.internal.name.domain.com"} {
 # inside the conditional block, add another log line saying that's where you are
 log local0. "Host matched, performing replacement operation"
 HTTP::header replace Location \
"[string map -nocase {long.internal.name.domain.com easyname.domain.com} [HTTP::header value Location

 # and again, you can log the result of the replacement operation
 log local0. "Replacement text = \
[string map -nocase {long.internal.name.domain.com easyname.domain.com} [HTTP::header value Location]
 }
}
```

When requests for "easyname.domain.com" were pushed through this new iRule, the result was the same (no replacements performed) but the following (very helpful!) log entries were generated:

```
HTTP_REQUEST: Host = easyname.domain.com
HTTP_REQUEST: Location =
HTTP_REQUEST: Host matched, performing replacement operation
HTTP_REQUEST: Replacement text =
```

Responses sometimes logged like this:

```
HTTP_RESPONSE: Host =
HTTP_RESPONSE: Location =
```

and sometimes like this:

```
HTTP_RESPONSE: Host =
HTTP_RESPONSE: Location = http://long.internal.name.domain.com/uri
```

Sooo... what got replaced? Not surprisingly, nothing. (At least the wrong "something" didn't get replaced. Or did it...? More on that later.)

You can tell by the absence of a value after some of the “=” that something wasn't seen where we expected it. You can tell by the absence of some log lines that some of the expected conditions were not met. In a nutshell, these are the 2 issues at hand:

1. No replacement text was generated on request even though the conditional code block was obviously executed.
2. No hostname was seen on response, so the conditional code block was NOT executed and no replacement text was generated.

Taking a closer look at the actual traffic being sent both ways (using HTTPwatch, tcpdump, or your favorite alternative trace tool) revealed some interesting and relevant details:

1. Request from the client does not include the Location header, only the Host header. Requests on the serverside of LTM have both Host and Location headers.
2. Response does not include the Host header, and only contains a Location header for redirect responses.
3. The internal hostname is clearly visible in hyperlinks in the HTTP payload (not just the Host and Location headers).

These observations lead to the following revelations and adjustments:

1. "Replacing" the wrong request header here -- actually LTM is inserting a new blank "Location" header into each request. (That would the the "wrong something" I mentioned earlier...) Use [HTTP::host] for the request condition and [HTTP::header replace] for the replacement operation. (No [string map] is necessary here since we already know the host header value.)
2. Using the wrong information for the response condition. Use [HTTP::status] for the response condition to limit processing only to redirects, then use [HTTP::header replace] with [string map] to replace only the hostname.
3. Payload replacement isn't being handled by the iRule at all, and rightly so: Use a stream profile applied to the same virtual server to translate the links in the HTTP payload.

The final iRule, including all the correct references, manipulations and optimizations, and implemented along with the stream profile, is as follows:

```
when HTTP_REQUEST {
 if { [HTTP::host] equals "easyname.domain.com"} {
 # replace header completely if it matches
 HTTP::header replace Host "long.internal.name.domain.com"
 }
}

when HTTP_RESPONSE {
 if { [HTTP::status] starts_with "3" }{
 # replace the Location header only if the response is a redirect,
 # since no other HTTP server responses contain the hostname in a header.
 HTTP::header replace Location \
[string map -nocase {long.internal.name.domain.com easyname.domain.com} [HTTP::header value Location]]
 # depend on stream profile to perform the hostname replacements in the HTTP payload
 }
}
```

Note: Information in this article written by Deb Allen.

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc. Corporate Headquarters info@f5.com	F5 Networks Asia-Pacific apacinfo@f5.com	F5 Networks Ltd. Europe/Middle-East/Africa emeainfo@f5.com	F5 Networks Japan K.K. f5j-info@f5.com

©2016 F5 Networks, Inc. All rights reserved. F5, F5 Networks, and the F5 logo are trademarks of F5 Networks, Inc. in the U.S. and in certain other countries. Other F5 trademarks are identified at f5.com. Any other products, services, or company names referenced herein may be trademarks of their respective owners with no endorsement or affiliation, express or implied, claimed by F5. CS04-00015 0113