Automated Web Analytics iRule Style

Joe Pruitt, 2008-20-03

If you are running a website, then odds are you are doing some form of analysis on the usage patterns of your end users. There are many ways to do this but one of the most common is to use a client side JavaScript interface to "ping" a external service. This article will discuss how to use an iRule to inject analytics code into HTML responses to enable the automation of analytics into your website software.

Introduction

Web Analytics is the study of the behavior of website visitors. To perform this analysis, data must be stored that identifies information about the client's identity (source ip address, browser user agent, etc), the current website being visited, the previous referring website (from the HTTP Referer header), along with other information.

Several vendors offer services to parse server logs for this information, but that incurs an administrative burden on the website admin when he/she has to make sure the software/service that is crunching the log files is up to date with all the logs from all the servers in all geographic locations.

Other options for web administrators are service offerings where a certain piece of JavaScript code can be inserted into each web page that you would like monitored. When the browser loading a page from your web server processes the HTML response, it processes the JavaScript code that performs a separate HTTP request to a 3rd party server that is storing all the details for that page request. The beauty of this is that it completely eliminates the need for the network admin to troll all web servers for HTTP server logs with the added bonus of providing almost instantaneous access to analysis on the data. Vendors in this space include Google, WebTrends, Quantcast, Omniture, VisiStat, Coremetrics, and many others.

The problem

For the simplest website that contains a couple of pages of content, manually adding the JavaScript needed to integrate with these 3rd party service is not usually an issue. But imagine the following scenarios:

- HTML responses are generated by application code, not by static files on the web server.
- Release criteria for application code requires testing and adding content to pages in production is not allowed.
- Multiple products from multiple application groups reside on a given server.
- 3rd party code is present where you don't have access to all the source that controls page generation.
- Server application updates have long dev/deploy cycles.

Separation of IT an Dev

This list can go on and on, but you get the point. If you decide to integrate web analysis to existing applications, you will need to have access to the source that generates those web pages. Unfortunately the team that is in charge of the application analysis typically is not the same team that has access or control to modify that source code of the application.

Wouldn't it be great if the analytics code could be added to the application without having to modify the application code?

Enter iRules

If you happen to have a BIG-IP sitting in front of your application server, then this is your lucky day. Since iRules have full access to inspection and modification of response content flowing through your network, it is fairly straight forward to write an iRule to inject client side JavaScript into the application stream without the application knowing a thing about it.

The iRule

The following example uses Google Analytics as an example, but it is fairly easy to replace the content of the "analytics" variable in RULE_INIT with the replacement code for any other service you might be using.

```
when RULE_INIT {
  # replace the value of the _uacct variable with your Google Analytics account.
  set analytics "<!-- Begin Analytics -->
<script src='http://www.google-analytics.com/urchin.js' type='text/javascript'></script>
<script type='text/javascript'>
_uacct = 'UA-XXXXX-X';
urchinTracker();
</script>
<!-- End Analytics -->
"
  set exist_search "urchinTracker()"
}
when HTTP_REQUEST {
  # Don't allow data to be chunked
  if { [HTTP::version] eq "1.1" } {
    if { [HTTP::header is_keepalive] } {
       HTTP::header replace "Connection" "Keep-Alive"
    }
    HTTP::version "1.0"
  }
}
when HTTP_RESPONSE {
  if { [HTTP::header Content-Type] starts_with "text/html" } {
    if { [HTTP::header exists "Content-Length"] && [HTTP::header "Content-Length"] <= 1048576} {
      #log local0. "content length: [HTTP::header {Content-Length}]"
      set content_length [HTTP::header "Content-Length"]
    } else {
      set content_length 1048576
    }
    #log local0. "Collecting $content_length bytes"
    if { $content_length > 0 } {
       HTTP::collect $content_length
    }
  }
}
when HTTP_RESPONSE_DATA {
```

```
when HTTP_RESPONSE_DATA {
  #log local0. "Content Type: [HTTP::header Content-Type]"
  if { ! ([HTTP::payload] contains $::exist_search) } {
    #log local0. "Payload didn't contain $::exist_search!"
    set idx [string last "</body>" [HTTP::payload]]
    if { -1 == $idx } {
      set idx [string last "</BODY>" [HTTP::payload]]
    }
    #log local0. "html end tag found at $idx"
    if { -1 == $idx } {
      set offset [HTTP::payload length]
    } else {
      set offset $idx
    }
    HTTP::payload replace $offset 0 $::analytics
  }
}
```

The code is pretty straight forward. In the RULE_INIT event, define the analytics script code to inject as well as the search term to use for inspection of existing analytics code. The HTTP_RESPONSE event collects the content length from the server and issues a HTTP::collect method to have the payload passed off to the HTTP_RESPONSE_DATA event. In the HTTP_RESPONSE_DATA event, determine if the response is an HTML page by the content type of "text/html" (appending analytics code to the end of a .gif image would likely cause some problems). If it's a HTML page, then check the payload for the existence of analytics code as defined in the "exist_search" global variable. If the iRule determines that analytics code is not present in the page, then it needs to determine where to inject the JavaScript code. It does this by searching for the "body" end tag (Google recommends adding the script code directly preceding the closing body tag). If no closing body tag is found, then the injection point is set to the end of the payload. Finally the "HTTP::payload replace" command is used to "inject" the script code at either the point where the end body tag begins or the end of the file. It acts as an injection since I'm setting the replacement length value to zero. Since zero characters are replaced, the new code is injected and the existing content is pushed to the end of the script code.

Conclusion

This example illustrates just an single example on how an iRule can be used to augment and enhance applications on the network easing the integration between the network administrator and the application development team. In a matter of minutes, as opposed to days/weeks to do it manually across code bases, Analytics have been added to every single HTML page returned through a virtual server.

Get the Flash Player to see this player.

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

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