Cookie LoJack vi iRules

Colin Walker, 2008-08-07

Web sites used to be simple. There were pages you would browse to on the web, and they would display information to you, via simple HTML. Things have progressed since then, taking many numerous strides along the way. From static to dynamic content, HTML to XHTML, CSS, and many other more dynamic languages, the face of web sites has changed. Most recently the web has largely been moving to truly interactive experiences using technologies such as AJAX and the like. These days, web sites share the web with web applications, and most places an average web user will travel to requires some amount of interaction between the client and back end systems where it used to be just a one way pull from the client.

To allow for this kind of interaction, the client that's accessing these sites and applications has to be able to not only be recognized as unique, but also has to be able to store relevant data in real-time, without being forced to download it and store it in a folder somewhere. How could you go shopping online without tracking what's in your cart? How could you get back to the same page in a web application after a browser crash if your progress wasn't stored somewhere? How can my web app know that you've already logged in, and not ask you to authorize yourself again after every page click? One of the most commonly used mechanisms for storing this data is to wrap it in a cookie and send it to the client.

Cookies are simple chunks of text that are passed from the serving web server to the client browser, where they are temporarily stored. The browser then returns the cookie, unchanged, with each HTTP request that it makes. This effectively gives state to an otherwise stateless transaction. Because the web server can store pieces of data with the client, it can note authentication, progress through an application, etc. Without them, each and every HTTP request would effectively be its own event, separate and individual from other requests that may have occurred in the same connection, with no way of being associated to earlier or later requests in the same transaction.

As you can see, cookies have become a rather large part of the way we use modern-day web sites and applications. It's largely because of this that there continues to be rising concern, in some areas, about the security implications of cookie use. As I mentioned before, a cookie is really just a chunk of text stored by the client, sent to and from the server. What, then, is to stop the malicious spoofing, modification or replacement of cookies in order to achieve things like unauthorized access to applications? How can you be sure that the cookie your application is reading was really the cookie your application sent to the client?

One possible solution is Cookie Encryption. Cookie encryption isn't a new concept. People have been encrypting the cookies being transmitted in some cases for a while, now. This, of course, introduces more load on back-end systems that are now having to encrypt and decrypt the cookies, not to mention there's no reason that a particular encrypted cookie couldn't be identified as a cookie representing a successful auth, for instance, then copied by a deviant user and used to spoof an authorized connection from somewhere else, or sometime in the future.

To offer a possible solution to this even more complex problem, I once again turn to iRules. In the example iRule below, contributed by one of our Field Systems Engineers here at F5, we present a way that you can enforce an added level of security with your cookies. This iRule is effectively a way to turn on "Cookie LoJack". What it does is take the original cookie that your server would be sending to a client browser and first pre-pend the requesting IP address before AES encrypting, then b64 encoding it. The client browser will get the encrypted cookie and store it, unable to read or change it, and then pass it back on the next request. The iRule again goes to work, decoding and un-encrypting the received cookie so it can be read in plaintext. As I said, though, it goes one step further than just encryption.

Once the cookie is again readable in plaintext, the iRule will check the string that it added to ensure that this cookie is indeed being received from the same IP address that it was sent to. In this way you can be even more certain that this cookie is valid and should be trusted. Not only was it encrypted, so it would be impossible for someone to have modified it, but we've now just verified that it was not copied and re-used from another location, as well.

This level of security may not be necessary in every web application deployment, but the number in which it could be greatly useful is quickly growing, and this is yet one more valuable solution offered by way of iRules. Many thanks go to Aidan Clark for submitting this example.

```
when RULE_INIT {
  set ::key [AES::key 128]
  set ::myCookieName CookieLoJack
  set ::protectedCookieName PREF
  set ::encryption 1
}

when HTTP_RESPONSE {
  if {[HTTP::cookie exists $::protectedCookieName]} {
    set serverIssuedCookie [HTTP::cookie value "$::protectedCookieName"]
    HTTP::cookie remove "$::protectedCookieName"
    # This is where we set the cookie and encrypt it
    if { ("$::encryption" eq "0") } {
      HTTP::cookie insert name "$::myCookieName" value "ClientIP=${clientIP}:ServerCookie=__${serverI
    } else {
      HTTP::cookie insert name "$::myCookieName" value [b64encode [AES::encrypt $::key "ClientIP=${cl
    }
  }
}

when HTTP_REQUEST {
  #replace with $client_addr
  set clientIP [IP::remote_addr]

  # If we find the protectedCookieName being submitted, snot it as we should not see it anymore.
  if {[HTTP::cookie exists $::protectedCookieName]} {
    HTTP::cookie remove "$::protectedCookieName"
  }

  if {[HTTP::cookie exists $::myCookieName]} {
    #We want to match the ClientIP in the cookie against the IP address of the Request
    #First we need to extract the cookie variables
    if { ("$::encryption" eq "0") } {
      set receivedCookie [HTTP::cookie "$::myCookieName"]
    } else {
      set receivedCookie [AES::decrypt $::key [b64decode [HTTP::cookie "$::myCookieName"] ]]
    }

    # Extract the IP address the cookie was issued to and store it in $receivedCookieClientIP
    set receivedCookieClientIP [getfield [getfield $receivedCookie ":" 1] "=" 2]

    # Extract the original cookie we were protecting and store it in $receivedCookieServerCookie
    set receivedCookieServerCookie [getfield $receivedCookie "__" 2]

    # Check the $clientIP against the $receivedCookieClientIP
    if { ( "$receivedCookieClientIP" eq "$clientIP") } {
      HTTP::cookie insert name "${::protectedCookieName}" value "${receivedCookieServerCookie}"
      HTTP::cookie remove "$::myCookieName"
    } else {
      HTTP::cookie remove "$::myCookieName"
      log local0. "Cookie $::myCookieName has been stolen. Cookie $::myCookieName has been stripped"
    }
  }
}
```


20080707_cookielojack.mp3

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