iRules 101 - #17 – Mapping Protocol Fields with the Binary Scan Command

Jason Rahm, 2010-24-06

Introduction

An iRule is a powerful and flexible feature of BIG-IP devices based on F5's exclusive TMOS architecture. iRules provide you with unprecedented control to directly manipulate and manage any IP application traffic. iRules utilizes an easy to learn scripting syntax and enables you to customize how you intercept, inspect, transform, and direct inbound or outbound application traffic. In this series of tech tips, we'll talk about the TCL language, its usage and control structures, as well as iRule extensions to the TCL language. Other articles in the series:

The Binary Scan Command

The **binary scan** command, like the **scan command** covered in the last entry in this series, parses strings. Only, as the adjective indicates, it parses binary strings. In this article, I'll highlight the command syntax and a few of the format string options below. Check the man page for the complete list of format options.

> **binary scan** *string formatString ?varName varName … ?*
>
> **c** - The data is turned into count 8-bit signed integers and stored in the corresponding variable as a list. If count is *, then all of the remaining bytes in string will be scanned. If count is omitted, then one 8-bit integer will be scanned.
>
> **S** - The data is interpreted as count 16-bit signed integers represented in big-endian byte order. The integers are stored in the corresponding variable as a list. If count is *****, then all of the remaining bytes in string will be scanned. If count is omitted, then one 16-bit integer will be scanned.

> **H** - The data is turned into a string of count hexadecimal digits in high-to-low order represented as a sequence of characters in the set "0123456789abcdef". The data bytes are scanned in first to last order with the hex digits being taken in high-to-low order within each byte. Any extra bits in the last byte are ignored. If count is *****, then all of the remaining hex digits in string will be scanned. If count is omitted, then one hex digit will be scanned.

Do the Research First!

Before you can start mapping fields, you need to know where the delimiters are, and that takes a little research. In this example, I want to list out the cipher suites presented to the BIG-IP LTM by the browser (more on that later), so I'm going to decode the fields in an SSLv3/TLSv1 client hello message. I took a capture with Wireshark to find the field delineations, but you could also pull this information from RFC 2246 if you are so inclined. Table 1 below shows the fields I'll extract or skip over in the course of the iRule code.

BYTE #	FIELD	INTERESTING VALUES		NOTES
0	RECORD TYPE	22 (0x16)	HANDSHAKE	Some browsers, like IE6, still submit the SSLv2 client hello message, even the though the handshake is still for version 3, the initial fields are different and won't match record type or version.
1-2	VERSION	768 (0x0300) 769 (0x0301)	SSL version 3 TLS version 1	
3-4	LENGTH			length of record
5	HANDSHAKE TYPE	1 (0x01)	Client Hello	If version is SSLv3/TLSv1, we only care about the clienthello, so we check the handshake type for that value. If not client hello, discard.
6-8	LENGTH			length of handshake
9-10	VERSION			
11-14	TIMESTAMP			
15-42	RANDOM DATA			
43	SESSION ID LENGTH			
44-45	CIPHER SUITES LENGTH			Variable, but we need this to correctly account for all the ciphers presented
46 - N	CIPHER SUITES			Will use these to compare against acceptable list in datagroup

Table 1

So why do I want to look at the cipher suites? I was contacted by a friend who wanted to increase security to 256-bit ciphers for those browsers that supported it, but would still allow users that don't to connect as well so they could be redirected to an error page requesting they upgrade. With the cipher settings as they were, the unsupported browser (in this case, IE6) could connect, but the newer browsers connected at 128-bit as well instead of 256-bit. I later learned (courtesy of none other than hoolio) that I could use @STRENGTH in my cipher settings to force the capable browsers up, but for the benefit of demonstrating the use the binary string command, I'll proceed with the iRule.

Standard Behaviors

I set up a clientssl profile with these cipher settings: "DEFAULT:!ADH:!EXPORT40:!EXP:!LOW". Those settings result in my test browsers (Chrome 5, FF 3.6.4, IE 8) all using the RC4-SHA 128-bit cipher. I used a simple iRule to display the information back to the client:

SSL Cipher in Use

```
when HTTP_REQUEST {
  set cipher_info "[SSL::cipher name]:[SSL::cipher bits]"
  HTTP::respond 200 content "$cipher_info"
```

```
}
```

Again, the easy (and better) fix here is to add strength to the cipher settings in the SSL profile like so: “DEFAULT:!ADH:!EXPORT40:!EXP:!LOW:@STRENGTH”, but instead I created an additional clientssl profile, adding not medium to the settings: “DEFAULT:!ADH:!EXPORT40:!EXP:!LOW:!MEDIUM”. Once I had that profile in place, work on the iRule could begin.

Using Binary Scan

Enough prep talk already, let me get to the good stuff! The iRule starts in CLIENT_ACCEPTED, disabling SSL and doing a TCP collect:

Binary Scans

```
when CLIENT_ACCEPTED {
  set lsec 0
  SSL::disable
  TCP::collect
}
when CLIENT_DATA {
  if { ! [info exists rlen] } {

  binary scan [TCP::payload] cSS rtype sslver rlen
  #log local0. "SSL Record Type $rtype, Version: $sslver, Record Length: $rlen"

  # SSLv3 / TLSv1.0
  if { $sslver > 767 && $sslver < 770 } {
    if { $rtype != 22 } {
      log local0. "Client-Hello expected, Rejecting.  \
                       Src: [IP::client_addr]:[TCP::remote_port] -> \
                       Dst: [IP::local_addr]:[TCP::local_port]"
      reject
      return
    }
```

As shown in the binary scan syntax, “c” grabs one byte and returns a signed integer, and “S” grabs two-bytes and returns a signed integer. The rest of the payload is thrown away. From table 1, we know the first three fields of SSLv3/TLSv1 client hello’s are one-byte, two-byte, two-byte, so a format string of cSS followed by three variable names takes care of the parsing. The clientssl profile doesn’t yet support TLS 1.1 or 1.2, so I’m only testing for SSLv3/TLSv1. The SSLv2 and earlier record format is different altogether, so the initial fields wouldn’t map this way anyway. This is true of IE6, which by default issues a SSLv2 client hello but supports SSLv3/TLSv1. Confused yet? Anyway, once the test for SSLv3/TLSv1 is complete, I then make sure the record type is a client hello, if not, no reason to continue.

Once the verification is complete that this really is a client hello, there is a string (pun intended!) of binary scans to sort the rest of the details out:

Binary Scans

```
#Collect rest of the record if necessary
if { [TCP::payload length] < $rlen } {
  TCP::collect $rlen
  #log local0. "Length is $len"
}
#skip record header and random data
set field_offset 43

#set the offset
binary scan [TCP::payload] @${field_offset}c sessID_len
set field_offset [expr {$field_offset + 1 + $sessID_len}]

#Get cipherlist length
binary scan [TCP::payload] @${field_offset}S cipherList_len
#log local0. "Cipher list length: $cipherList_len"

#Get ciphers, separate into a list of elements
set field_offset [expr {$field_offset + 2}]
set cipherList_len [expr {$cipherList_len * 2}]
binary scan [TCP::payload] @${field_offset}H${cipherList_len} cipherlist
#log local0. "Cipher list: $cipherlist"
```

I'm using the "@" symbol to skip bytes in the binary string. Also, the curly brackets are necessary to surround the variable name to distinguish your variable from the significance of the format string characters. In the first binary scan above, I'm skipping the record header and all the random data as I don't need that information, but I do need the session ID length so I can set my offset properly to get to the cipher list. In the second binary scan, again I'm skipping (from original payload) the new offset and then storing another two-byte signed integer that tells me how long the cipher list is. From that information, I set a new offset, set the cipher list length (number of ciphers * 2, each cipher is two-bytes) and then perform the final binary scan to store the entire cipher list (in hex high->low characters, hence the "H" in the string format).

The rest of the rule is processing that cipher list (thanks to Colin for some tcl-fu in the for loop below), and sending users to a different clientssl profile if any of the ciphers in the client hello list match the ciphers in a string datagroup you define, in this case, accepted_ciphers (values were four hex character pairs, like c005, 0088, etc). For browsers incapable of the higher security profile, they would be redirected to an error page instructing them to upgrade.

Rule Conclusion

```
set profile_select 0
for { set x 0 } { $x < $cipherList_len } { incr x 4 } {
  set start $x
  set end [expr {$x+3}]
  #Check cipher against 256-bit ciphers in accepted_ciphers data group
  if { [matchclass [string range $cipherlist $start $end] equals accepted_ciphers] } {
    #If 256-bit capable, select High Security profile
    set profile_select "SSL::profile clientssl_highSecurity"
    break
```

```
    }
  }

  #Set New Profile with eval since SSL::profile errors in CLIENT_ACCEPTED event
  if { $profile_select != 0 } {
    eval $profile_select
    #log local0. "Profile changed to High Security"
  } else { set lsec 1 }

      event CLIENT_DATA disable
      SSL::enable
      TCP::release
  } else {
      set lsec 1
      event CLIENT_DATA disable
      SSL::enable
      TCP::release
  }
}
}
```

Conclusion

The rule is useless in production, but is a good teaching example on the use of binary scan. There are a few more good examples in the wiki. Check out this DNS modification example and this tftp file server iRule (companion article for the tftp iRule here). That one throws in a binary format to boot. Happy coding!

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