# RADIUS Load Balancing with iRules

**Colin Walker, 2008-04-04**

### What is RADIUS?

“Remote Authentication Dial In User Service” or RADIUS is a very mature and widely implemented protocol for exchanging ”Triple A” or “Authentication, Authorization and Accounting” information.

RADIUS is a relatively simple, transactional protocol. Clients, such as remote access server, FirePass, BIG-IP, etc. originate RADIUS requests (for example, to authenticate a user based on a user/password combination) and then wait for a response from the RADIUS server.

Information is exchanged between a RADIUS client and server in the form of attributes. User-name, user-password, IP Address, port, and session state are all examples of attributes. Attributes can be in the format of text, string, IP address, integer or timestamp. Some attributes are variable in length, some are fixed.

### Why is protocol-specific support valuable?

In typical UDP Load Balancing (not protocol-specific), there is one common challenge: if a client always sends requests with the same source port, packets will never be balanced across multiple servers. This behavior is the default for a UDP profile.

To allow load balancing to work in this situation, using "Datagram LB" is the common recommendation or the use of an immediate session timeout. By using Datagram LB, every packet will be balanced. However, if a new request comes in before the reply for the previous request comes back from the server, BIG-IP LTM may change source port of that new request before forwards it to the server. This may result in an application not acting properly. In this later case, “Immediate timeout” must then be used. An additional virtual server may be needed for outbound traffic in order to route traffic back to the client.

In short, to enable load balancing for RADIUS transaction-based traffic coming from the same source IP/source port, Datagram LB or immediate timeout should be employed. This configuration works in most cases. However, if the transaction requires more than 2 packets (1 request, 1 response), then, further BIG-IP LTM work is needed.

An example where this is important occurs in RADIUS challenge/response handshakes, which require 4 packets:

> * Client ---- access-request ---> Server
> * Client <-- access-challenge --- Server
> * Client --- access-request ----> Server
> * Client <--- access-accept ----- Server

For this traffic to succeed, all packets associated with the same transaction must be returned to the same server. In this case, custom layer 7 persistence is needed.

iRules can provide the needed persistency. With iRules that understand the RADIUS protocol, BIG-IP LTM can direct traffic based on any attribute sent by client or persist sessions based on any attribute sent by client or server. Session management can then be moved to the BIG-IP, reducing server-side complexity. BIG-IP can provide almost unlimited intelligence in an iRule that can even re-calculate md5, modify usernames, detect realms, etc. BIG-IP LTM can also provide security at the application level of the RADIUS protocol, rejecting malformed traffic, denial of service attacks, or similar threats using customized iRules.

### Solution

- Datagram LB UDP profile or immediate timeout may be used if requests from client always use the same source IP/port.
- If immediate timeout is used, there should be an additional VIP for outbound traffic originated from server to client

- If immediate timeout is used, there should be an additional VIP for outbound traffic originated from server to client and also an appropriate SNAT (same IP as VIP).
- Identifier or some attributes can be used for Universal Inspection Engine (UIE) persistence.
- If immediate timeout/2-side-VIP technique are used, these should be used in conjunction with session command with "any" option.

**iRules**

**1)** Here is a sample iRule which does nothing except decode and log some attribute information. This is a good example of the depth of fluency you can achieve via an iRule dealing with RADIUS.

```
when RULE_INIT {
  array set ::attr_code2name {
          1      User-Name
          2      User-Password
          3      CHAP-Password
          4      NAS-IP-Address
          5      NAS-Port
          6      Service-Type
          7      Framed-Protocol
          8      Framed-IP-Address
          9      Framed-IP-Netmask
         10      Framed-Routing
         11      Filter-Id
         12      Framed-MTU
         13      Framed-Compression
         14      Login-IP-Host
         15      Login-Service
         16      Login-TCP-Port
         17      (unassigned)
         18      Reply-Message
         19      Callback-Number
         20      Callback-Id
         21      (unassigned)
         22      Framed-Route
         23      Framed-IPX-Network
         24      State
         25      Class
         26      Vendor-Specific
         27      Session-Timeout
         28      Idle-Timeout
         29      Termination-Action
         30      Called-Station-Id
         31      Calling-Station-Id
         32      NAS-Identifier
         33      Proxy-State
         34      Login-LAT-Service
         35      Login-LAT-Node
         36      Login-LAT-Group
         37      Framed-AppleTalk-Link
         38      Framed-AppleTalk-Network
         39      Framed-AppleTalk-Zone
         60      CHAP-Challenge
         61      NAS-Port-Type
         62      Port-Limit
         63      Login-LAT-Port
    }
}
```

```
when CLIENT_ACCEPTED {
    binary scan [UDP::payload] cH2SH32cc code ident len auth \
        attr_code1 attr_len1
    log local0. "code       = $code"
    log local0. "ident      = $ident"
    log local0. "len        = $len"
    log local0. "auth       = $auth"

    set index 22
    while { $index < $len } {
        set hsize [expr ( $attr_len1 - 2 ) * 2]
        switch $attr_code1 {
            11 -
            1 {
                binary scan [UDP::payload] @${index}a[expr $attr_len1 - 2]cc \
                    attr_value attr_code2 attr_len2
                log local0. " $::attr_code2name($attr_code1) = $attr_value"
            }
            9 -
            8 -
            4 {
                binary scan [UDP::payload] @${index}a4cc rawip \
                    attr_code2 attr_len2
                log local0. " $::attr_code2name($attr_code1) =\
                [IP::addr $rawip mask 255.255.255.255]"
            }
            13 -
            12 -
            10 -
            7 -
            6 -
            5 {
                binary scan [UDP::payload] @${index}Icc attr_value \
                    attr_code2 attr_len2
                log local0. " $::attr_code2name($attr_code1) = $attr_value"
            }
            default {
                binary scan [UDP::payload] @${index}H${hsize}cc \
                    attr_value attr_code2 attr_len2
                log local0. " $::attr_code2name($attr_code1) = $attr_value"
            }
        }
        set index [ expr $index + $attr_len1 ]
        set attr_len1 $attr_len2
        set attr_code1 $attr_code2
    }
}
when SERVER_DATA {
    binary scan [UDP::payload] cH2SH32cc code ident len auth \
        attr_code1 attr_len1
    log local0. "code       = $code"
    log local0. "ident      = $ident"
    log local0. "len        = $len"
    log local0. "auth       = $auth"

    set index 22
    while { $index < $len } {
        set hsize [expr ( $attr_len1 - 2 ) * 2]
```

```
        set hsize [expr ( $attr_len1 - 2 ) * 2]
        switch $attr_code1 {
            11 -
            1 {
                binary scan [UDP::payload] @${index}a[expr $attr_len1 - 2]cc \
                    attr_value attr_code2 attr_len2
                log local0. " $::attr_code2name($attr_code1) = $attr_value"
            }
            9 -
            8 -
            4 {
                binary scan [UDP::payload] @${index}a4cc rawip \
                    attr_code2 attr_len2
                log local0. " $::attr_code2name($attr_code1) =\
                    [IP::addr $rawip mask 255.255.255.255]"
            }
            13 -
            12 -
            10 -
            7 -
            6 -
            5 {
                binary scan [UDP::payload] @${index}Icc attr_value \
                    attr_code2 attr_len2
                log local0. " $::attr_code2name($attr_code1) = $attr_value"
            }
            default {
                binary scan [UDP::payload] @${index}H${hsize}cc \
                    attr_value attr_code2 attr_len2
                log local0. " $::attr_code2name($attr_code1) = $attr_value"
            }
        }
        set index [ expr $index + $attr_len1 ]
        set attr_len1 $attr_len2
        set attr_code1 $attr_code2
    }
}
```

This iRule could be applied to many areas of interest where a particular value needs to be extracted. For example, the iRule could detect the value of specific attributes or realm and direct traffic based on that information.

**2)** This second iRule allows UDP Datagram LB to work with 2 factor authentication. Persistence in this iRule is based on "State" attribute (value = 24). Another great example of the kinds of things you can do with an iRule, and how deep you can truly dig into a protocol.

```
when CLIENT_ACCEPTED {
    binary scan [UDP::payload] ccSH32cc code ident len auth \
        attr_code1 attr_len1
    set index 22
    while { $index < $len } {
        set hsize [expr ( $attr_len1 - 2 ) * 2]
        binary scan [UDP::payload] @${index}H${hsize}cc attr_value \
            attr_code2 attr_len2
        # If it is State(24) attribute...
        if { $attr_code1 == 24 } {
            persist uie $attr_value 30
            return
```

```
        }
        set index [ expr $index + $attr_len1 ]
        set attr_len1 $attr_len2
        set attr_code1 $attr_code2
    }
}
when SERVER_DATA {
    binary scan [UDP::payload] ccSH32cc code ident len auth \
        attr_code1 attr_len1
    # If it is Access-Challenge(11)...
    if { $code == 11 } {
      set index 22
      while { $index < $len } {
        set hsize [expr ( $attr_len1 - 2 ) * 2]
        binary scan [UDP::payload] @${index}H${hsize}cc attr_value \
            attr_code2 attr_len2
        if { $attr_code1 == 24 } {
            persist add uie $attr_value 30
            return
        }
        set index [ expr $index + $attr_len1 ]
        set attr_len1 $attr_len2
        set attr_code1 $attr_code2
      }
    }
}
```

**Conclusion**

With iRules, BIG-IP can understand RADIUS packets and make intelligent decisions based on RADIUS protocol information. Additionally, it is also possible to manipulate RADIUS packets to meet nearly any application need.

Contributed by: Nat Thirasuttakorn



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