HTTP Request Cloning

Use Case:

In load balancing environment, due to specific nature of traffic or specific action like external parsing or logging, it is needed to clone the incoming HTTP requests. While load balancing decision is taken, the original request is sent to selected server but the cloned request can be sent to another server which may or may not be part of the load balancing vserver pool.

F5 iRules:

```
############
# First Rule #
############
rule http_request_clone_one_pool {
# Clone HTTP requests to one clone pool
when RULE_INIT {
        # Log debug locally to /var/log/ltm? 1=yes, 0=no
        set static::hsl_debug 1

        # Pool name to clone requests to
        set static::hsl_pool "my_syslog_pool"
}
when CLIENT_ACCEPTED {

        if {[active_members $static::hsl_pool]==0}{
                log "[IP::client_addr]:[TCP::client_port]: [virtual name] $sta
tic::hsl_pool down, not logging"
                set bypass 1
                return
        } else {
                set bypass 0
        }

        # Open a new HSL connection if one is not available
```

```
        set hsl [HSL::open -proto TCP -pool $static::hsl_pool]

        if {$static::hsl_debug}{log local0. "[IP::client_addr]:[TCP::client_po
rt]: New hsl handle: $hsl"}

}

when HTTP_REQUEST {


        # If the HSL pool is down, do not run more code here

        if {$bypass}{

                return

        }

        # Insert an XFF header if one is not inserted already

        # So the client IP can be tracked for the duplicated traffic

        HTTP::header insert X-Forwarded-For [IP::client_addr]


        # Check for POST requests

        if {[HTTP::method] eq "POST"}{


                # Check for Content-Length between 1b and 1Mb

                if { [HTTP::header Content-Length] >= 1 and [HTTP::header Cont
ent-Length] < 1048576 }{

                        HTTP::collect [HTTP::header Content-Length]

                } elseif {[HTTP::header Content-Length] == 0}{

                        # POST with 0 content-length, so just send the headers

                        HSL::send $hsl "[HTTP::request]\n"

                        if {$static::hsl_debug}{log local0. "[IP::client_addr]
:[TCP::client_port]: Sending [HTTP::request]"}

                }

        } else {

                # Request with no payload, so send just the HTTP headers to th
e clone pool

                HSL::send $hsl "[HTTP::request]\n"

                if {$static::hsl_debug}{log local0. "[IP::client_addr]:[TCP::c
lient_port]: Sending [HTTP::request]"}

        }

}

when HTTP_REQUEST_DATA {
```

```
        # The parser does not allow HTTP::request in this event, but it works
        set request_cmd "HTTP::request"
        if {$static::hsl_debug}{log local0. "[IP::client_addr]:[TCP::client_port]: Collected [HTTP::payload length] bytes,\
                sending [expr {[string length [eval $request_cmd]] + [HTTP::payload length]}] bytes total"}
        HSL::send $hsl "[eval $request_cmd][HTTP::payload]\nf"
}
}



#############
# Second Rule #
#############
rule http_request_close_xnum_pools {
# Clone HTTP requests to X clone pools
when RULE_INIT {

        # Set up an array of pool names to clone the traffic to
        # Each pool should be one server that will get a copy of each HTTP request
        set static::clone_pools(0) http_clone_pool1
        set static::clone_pools(1) http_clone_pool2
        set static::clone_pools(2) http_clone_pool3
        set static::clone_pools(3) http_clone_pool4

        # Log debug messages to /var/log/ltm? 0=no, 1=yes
        set static::clone_debug 1

        set static::pool_count [array size static::clone_pools]
        for {set i 0}{$i < $static::pool_count}{incr i}{
                log local0. "Configured for cloning to pool $clone_pools($i)"
        }
}
when CLIENT_ACCEPTED {
```

```
        # Open a new HSL connection to each clone pool if one is not available
        for {set i 0}{$i < $static::pool_count}{incr i}{
                set hsl($i) [HSL::open -proto TCP -pool $static::clone_pools($i)]
                if {$static::clone_debug}{log local0. "[IP::client_addr]:[TCP::client_port]: hsl handle ($i) for $static::clone_pools($i): $hsl($i)"}
        }
}
when HTTP_REQUEST {

        # Insert an XFF header if one is not inserted already
        # So the client IP can be tracked for the duplicated traffic
        HTTP::header insert X-Forwarded-For [IP::client_addr]

        # Check for POST requests
        if {[HTTP::method] eq "POST"}{

                # Check for Content-Length between 1b and 1Mb
                if { [HTTP::header Content-Length] >= 1 and [HTTP::header Content-Length] < 1048576 }{
                        HTTP::collect [HTTP::header Content-Length]
                } elseif {[HTTP::header Content-Length] == 0}{
                        # POST with 0 content-length, so just send the headers
                        for {set i 0}{$i < $static::pool_count}{incr i}{
                                HSL::send $hsl($i) "[HTTP::request]\n"
                                if {$static::clone_debug}{log local0. "[IP::client_addr]:[TCP::client_port]: Sending to $static::clone_pools($i), request: [HTTP::request]"}
                        }
                }
        } else {
                # Request with no payload, so send just the HTTP headers to the clone pool
                for {set i 0}{$i < $static::pool_count}{incr i}{
                        HSL::send $hsl($i) [HTTP::request]
```

```
                        if {$static::clone_debug}{log local0. "[IP::client_add
r]:[TCP::client_port]: Sending to $static::clone_pools($i), request: [HTTP::re
quest]"}

                }

        }

}

when HTTP_REQUEST_DATA {

        # The parser does not allow HTTP::request in this event, but it works

        set request_cmd "HTTP::request"

        for {set i 0}{$i < $static::pool_count}{incr i}{

                if {$static::clone_debug}{log local0. "[IP::client_addr]:[TCP:
:client_port]: Collected [HTTP::payload length] bytes,\

                        sending [expr {[string length [eval $request_cmd]] + [
HTTP::payload length]}] bytes total\

                        to $static::clone_pools($i), request: [eval $request_c
md][HTTP::payload]"}

                HSL::send $hsl($i) "[eval $request_cmd][HTTP::payload]\n"

        }

}

}
```

URL: https://devcentral.f5.com/codeshare/http-request-cloning

NetScaler Solution:

```
add httpcallout clone_req -vServer vip1 -fullReqExpr
'HTTP.REQ.FULL_HEADER+HTTP.REQ.BODY(HTTP.REQ.CONTENT_LENGTH)' -
returnType BOOL -resultExpr TRUE


add responder policy clone_req_pol
'SYS.NON_BLOCKING_HTTP_CALLOUT(clone_req) &&
SYS.VSERVER("vip1").STATE.EQ(UP)' NOOP
```

Here we are creating a HTTP Callout which clones and sends the request out to vserver “vip1”. Callout is invoked by the Responder policy which can then be bound to VSERVER (which needs to be cloned) or to Global responder bind point. Here we are not going to look at the response coming back for the original request. If you want to clone the request to multiple endpoints then just create those many HTTP Callouts.