Rapid iRule Removal via Tmsh Script

Jason Rahm, 2010-14-05

The BIG-IP GUI is pretty slick and there have been massive improvements in function and efficiency since my first exposure in version 4.2. As good as it is, however, some tasks are just better suited to the CLI. Take maintenance, for example. How long will it take you to remove an iRule from a virtual server? A quick attempt from login to removal was about 14 seconds. While already logged in, it took me about nine seconds. So assuming you know exactly what virtuals an iRule needs to be removed from, and that you can stay on top of which you've pulled from without duplicating effort, you can see how efficiency flies right out the window if you have 20, or 30, or maybe 300 virtuals to touch, right?

Enter scripting. This scenario is why scripts exist. It takes a mundane manual task and automates it, taking not only less time, but removing the human error element as well (assuming you don't design the errors into your script!) In this tech tip, I'm going to look at using tmsh scripting to evaluate all the virtual servers on the BIG-IP and if an iRule called "x" is present, remove it.

Why remove iRules? They Rock, right?

Well, yes, they do! But, there are differently purposed iRules deployed. Some are critical to site functionality—remove them, and the site is down hard. Other rules are present for logging, d

ebugging, injecting content, etc. These aren't necessary, and during times of troubleshooting or resource concerns, could be removed. As indicated above, this is trivial from a standpoint of procedure, but it's the time involved that counts. So scripting it is. The problem now is what kind of script to go with? The options are a shell script, a perl script, a tmsh script, or an iControl script. The first two are certainly possible, but must utilize raw BIG-IP configuration data and then parse it into objects for manipulation. For the latter two, the data is already presented as objects, so the complexity of the script is reduced dramatically. iControl gets the stage all the time, so I chose tmsh for this job.

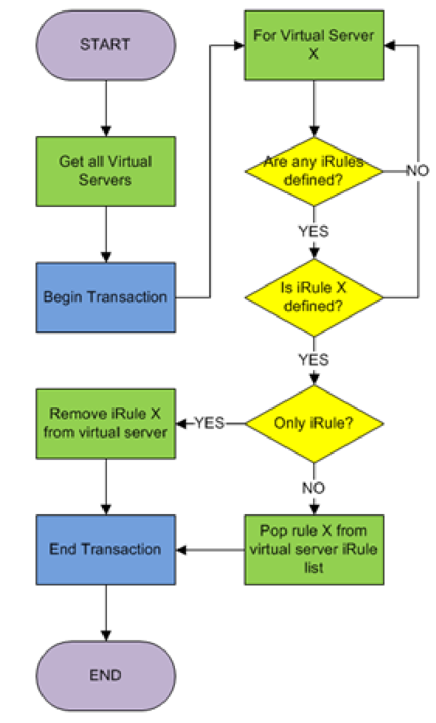

Figure 1

A Little Planning Goes a Long Way

Sometimes I get ahead of myself and start scripting immediately. This is more fun, but always results in rewriting something as I've certainly made logic errors and assumptions that don't pan out. So let's start with the high level approach, and then work our way into code.

Consider this virtual server configuration:

Virtual Server Configuration

```
ltm virtual tmshtest10 {
    destination 10.10.20.159:http
    ip-protocol tcp
    mask 255.255.255.255
    pool att_rtr
    profiles {
        http { }
        stream { }
        tcp { }
    }
    rules {
        filler1
        filler3
        filler4
        filler5
    }
}
```

Each of these keywords and values can be accessed in tmsh as an object. I don't really care about any of the keywords in a virtual except **rules**, since it's only a specific iRule I want to remove. So that's the first test in the script. After collecting all the vips into a variable, I want to loop through that list and test for the presence of that keyword. If it is not defined, I can move on to the next virtual. Once I've verified the keyword is present, however, I next need to see if the iRule I want to remove is defined within the rules keyword. If it is, then I want to take action on this virtual server. Since there is no remove action associated with iRules, I need to test how many iRules are currently applied. If the iRule I want to remove is the only one, I can just modify the virtual server with the **rules none** attributes. However, if there are more than one, I need to "pop" the specific rule from the list, then re-apply the iRules that will remain. Finally, I'll wrap all the looping and modification into a transaction, so if there are any failures in the script the desired actions will not be committed. If you are a visual learner, the approach I've just described is presented in a workflow diagram in Figure 1.

The Code

Every tmsh script requires the **script::run** procedure. I start with a conditional to make sure the rule name has been supplied as an argument (the only argument). I then store the rule name and the virtual server configurations in a couple variables, and set a couple other variables for use in the loop. The first two conditionals in the for loop test for presence of rules and then the specific rule. If either conditional fails, that instance of the loop is terminated and the loop moves on to the next virtual. I originally had a little more "logic" in there on both conditionals, but a second set of eyes from Mark Crosland, the author of tmsh, cleaned it up significantly, taking advantage of the zero return from tmsh::get_field_value in the absence of the keyword. The third conditional evaluates the number of rules present. If only one, the script utilizes the none attribute for rules. If more than that, the rule must be removed from the list, which is done with lreplace. The lappend action is just for printing the actions the script took on the active configuration.

tmsh script

```
proc script::run {} {
    if { $tmsh::argc != 2 } {
        puts "A single rule name must be provided"
        exit
    }
    set rulename [lindex $tmsh::argv 1]
    set rules ""
    set vips [tmsh::get_config /ltm virtual]
    set vips_in_play ""
    tmsh::begin_transaction

    foreach vip $vips {
        if { [tmsh::get_field_value $vip "rules" rules] == 0 } {
            continue
        }
        if { [lsearch -exact $rules $rulename] == -1 } {
            continue
        }
        if { [llength $rules] < 2 } {
            tmsh::modify /ltm virtual [tmsh::get_name $vip] rules none
            lappend vips_in_play $vip
        } else {
            set id [lsearch -exact $rules $rulename]
            set keepers [lreplace $rules $id $id]
            tmsh::modify /ltm virtual [tmsh::get_name $vip] rules "{ $keepers }"
            lappend vips_in_play $vip
        }
    }

    tmsh::commit_transaction

    puts "The $rulename iRule was removed from the following virtuals: "
    foreach x $vips_in_play {
        puts "\t[tmsh::get_name $x]"
    }
}
```

The Demo

For a quick walkthrough of the process involved in removing an iRule, and a demonstration of the script, check out the video below, which is also viewable in the Monitoring & Management section of the Tutorials Page. The script above is posted in the tmsh wiki.

Remove iRules from Virtual Servers with Tmsh