Vulnerability Patching via iRules: VU#520827 for PHP

Colin Walker, 2012-17-05

Security is a top level priority in nearly every IT infrastructure these days. Whether it's keeping server patching up to date, putting in place hardened firewalls, password security models, denial of service prevention or any of the other thousands of means of securing your application infrastructure from those intent on doing harm, it tends to be a high priority. As such, it also often times ends up being a relatively large cost to the organization, both in terms of finances and man hours. Simply put, people spend a lot of time thinking about and implementing security. Even with a bevy of security products in place there is often time a large amount of time trying to respond to the array if new vulnerabilities released every week, due to the highly flexible nature of the attacks.

iRules can be a powerful ally in the fight against packet-based evil. Because it is a massively flexible tool, lives on a proxy that is often near the edge of the network, and can perform full inspection in real-time, often times the quickest way to mitigate a given vulnerability might be a few minutes invested into writing up an iRule. This was demonstrated recently by Joel Moses, who forwarded me some code to resolve a recent PHP-CGI query vuln: http://www.kb.cert.org/vuls/id/520827. Here's what PHP's website has to say about the issue:

> "PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML." When PHP is used in a CGI-based setup (such as Apache's mod_cgid), the php-cgi receives a processed query string parameter as command line arguments which allows command-line switches, such as -s, -d or -c to be passed to the php-cgi binary, which can be exploited to disclose source code and obtain arbitrary code execution."

So basically, in this vulnerability the attack vector is through passing command-line switches to force the PHP binary to output more data than it should, expose the source code, or even arbitrarily execute code sent by the attacker. As you can imagine, none of those things are good. The problem is, a solution is not necessarily close at hand. You could upgrade your version of PHP, which may often times not be possible without extensive application testing. This testing takes time, which is both money and risk with a possible remote execution vuln in the wild. A faster approach, offered up by cert.org in the above link is to implement a mod_rewrite rule via Apache. That's certainly lighter weight than a PHP upgrade, but in many deployments will still require a fair amount of red tape, change control windows and a push to all PHP hosting Apache servers.

If you want to implement the fix in one place, quickly and easily however, a relatively simple iRule will do the job nicely. By recreating the prescribed rewrite logic in iRule form you now have a single fix to put into place that will protect all webservers behind it ... right now. After the vuln is patched via network side scripting you can do whatever testing and change control is needed to put in place a permanent fix if you decide to do so. Doing that with the peace of mind that you are no longer exposed to the vuln, however, is a pretty comforting feeling.

All right, on to the iRule!

```
when HTTP_REQUEST {
  switch -exact -- [HTTP::method] {
    "GET" -
    "POST" {
      if { not ([HTTP::query] equals "") && ([HTTP::path] ends_with ".php")} {
        set queryname_list [split [HTTP::query] "&"]
        set scan_index [lsearch -all -regexp $queryname_list "^-{1,2}.*"]
        if { [llength $scan_index] > 0 } {
          foreach query_item $scan_index {
            set queryname_list [lreplace $queryname_list $query_item $query_item]
          }
        }
        set fixed_query_length [llength $queryname_list]
        if { $fixed_query_length == 0 } {
          HTTP::uri "[HTTP::path]"
        } else {
          if { $fixed_query_length > 1 } {
            set queryname_list [join $queryname_list "&"]
          }
          HTTP::uri "[HTTP::path]?${queryname_list}"
        }
```

```
22:         }
23:       }
24:     }
25: }
```

As you can see this is no monster chunk of code. In a few easy lines, you can have the vulnerability behind you and carry on with business as usual until it's convenient for you to implement whatever permanent fix your app and security teams decide appropriate. I don't know about you, but to me that sounds a lot nicer than sounding the alarm and rush upgrading or forcing through config changes to n boxes across the DMZ.

This is also a prime example of how you can stay up to date to the day, if not within a couple of hours, if you've got a keen eye on the vuln notifications and a decent grasp on iRules. This iRule could have been in place within hours of release without having to jiggle the app stack at all. Ask your app developer/owner how much that is worth to them.

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.	F5 Networks	F5 Networks Ltd.	F5 Networks
Corporate Headquarters	Asia-Pacific	Europe/Middle-East/Africa	Japan K.K.
info@f5.com	apacinfo@f5.com	emeainfo@f5.com	f5j-info@f5.com

©2016 F5 Networks, Inc. All rights reserved. F5, F5 Networks, and the F5 logo are trademarks of F5 Networks, Inc. in the U.S. and in certain other countries. Other F5 trademarks are identified at f5.com. Any other products, services, or company names referenced herein may be trademarks of their respective owners with no endorsement or affiliation, express or implied, claimed by F5. CS04-00015 0113