How To Limit URI Length Without Recompiling Apache

Lori MacVittie, 2010-27-12

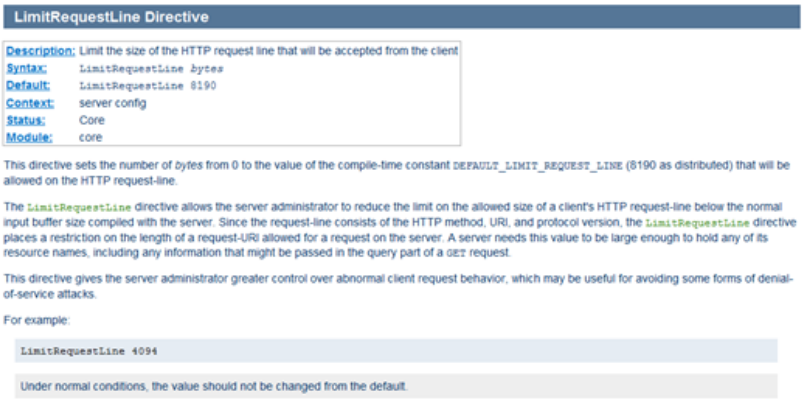
Use network-side scripting, of course!

While just about every developer and information security professional knows that a buffer-overflow exploit can result in the execution of malicious code not many truly grok the "why". Fortunately, it's not really necessary for either one to be able to walk through the execution stack and trace the byte-code as it overwrites registers and then jumps to execute it. They know it's A Very Bad Thing™ and perhaps more importantly they know how to stop it.

SECONDARY and TERTIARY DEFENSE REQUIRED

The best place to prevent a buffer-overflow vulnerability is in the application code. Never trust input whether from machine or man. Period. A simple check on the length of a string-based parameter can prevent vulnerabilities that may exist at the platform or operating system layer from being exploited. That's true of almost all vulnerabilities, not just buffer overruns, by the way. An overly long input parameter could be an attempt at XSS or an SQLi, as well. Both tend to extend the "normal" data and while often obfuscated, the sheer length of such strings can indicate the presence of something malicious and almost certainly something that should be investigated.

Assuming for whatever reason that this isn't possible (and we know from research and surveys and live data from organizations like WhiteHat Security that it isn't for a number of very valid reasons) it is likely the case that information security and operational administrators will go looking for a secondary defense. As the majority of applications today are deployed as web applications, that generally means looking to the web or application server for limitations on URL and/or parameter lengths as those are the most likely avenues of attack. One defense can be easily found if you're deploying on Apache in the "LimitRequestLine" compilation directive.



LimitRequestLine Directive

Description: Limit the size of the HTTP request line that will be accepted from the client
Syntax: LimitRequestLine *bytes*
Default: LimitRequestLine 8190
Context: server config
Status: Core
Module: core

This directive sets the number of *bytes* from 0 to the value of the compile-time constant DEFAULT_LIMIT_REQUEST_LINE (8190 as distributed) that will be allowed on the HTTP request-line.

The LimitRequestLine directive allows the server administrator to reduce the limit on the allowed size of a client's HTTP request-line below the normal input buffer size compiled with the server. Since the request-line consists of the HTTP method, URI, and protocol version, the LimitRequestLine directive places a restriction on the length of a request-URI allowed for a request on the server. A server needs this value to be large enough to hold any of its resource names, including any information that might be passed in the query part of a GET request.

This directive gives the server administrator greater control over abnormal client request behavior, which may be useful for avoiding some forms of denial-of-service attacks.

For example:

```
LimitRequestLine 4094
```

Under normal conditions, the value should not be changed from the default.

Yes, I said compilation directive. You'll have to recompile Apache but that's what open source is all about, right? Customization. Rapid solutions to security vulnerabilities. Agile infrastructure.

While you're in there, you might want to consider evaluating the "LimitRequestFields" and "LimitRequestFieldSize" variables, too. These two variables control the number of HTTP headers allowed as well as the length of a header field and could potentially prevent an exploit of the platform and underlying operating system coming in through the HTTP headers. Yes, such exploits do exist, and as always, better safe than sorry.

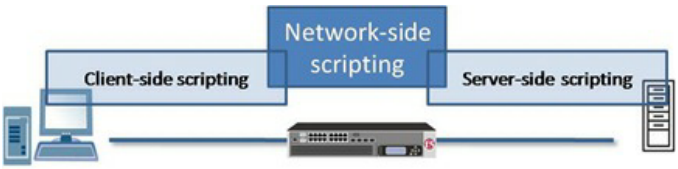
While all certainly true and valid statements regarding open source software the reality is that changing the core platform code results in a long-term commitment to re-applying those changes every time the core platform is upgraded or patched. Ask an enterprise PeopleSoft developer how that has worked for them over the past decade or so – but fair warning, you'll want to be out of spitting range when you do.

Compilation has a secondary negative – it's not agile, even though open source is. If you run into a situation in which you need to change these values you're going to have to recompile, retest, and redeploy. And you're going to have to regression test every application deployed on that particular modified platform. Starting to see why the benefit of customization in open source software is rarely truly taken advantage of?

There's a better way, a more agile way, a flexible way.

NETWORK-SIDE SCRIPTING

Whenever a solution involves the inspection and potential rejection or modification of HTTP-borne data based on, well, attributes like length, encoding, schema, etc… it should ring a bell and give pause for thought. These are variables in every sense of the word. If you decide to restrict input based on these values you necessarily open yourself up to maintaining and, in particular, updating those limitations across the life of the application in question. Thus it seems logical that the actual implementation of such limitations would leverage a location and solution that has as little impact on the rest of the infrastructure as possible. You want to maximize the coverage of the implementation solution while minimizing the disruption and impact on the infrastructure.



There happens to be a strategic point of control that very much fits this description: centralization of functionality at a point of aggregation that maximizes coverage while minimizing disruption. As an added bonus the coverage is platform-agnostic, which means Apache, IIS, can be automagically covered without modifying the platforms themselves. That strategic point in the architecture is a network-side scripting enabled application delivery controller (the Load balancer for you old skool operations folks).

See, when you take advantage of a full-proxy and really leverage its capabilities you can implement solutions that are by definition application-aware whilst maintaining platform-agnosticism. That's important because the exploits you're looking to stop are generally specific to an application; even when they target the platform they take advantage of application data manipulation and associated loopholes in processing of that data. While the target may be the platform, the miscreant takes aim and transports the payload via, well, the payload. The data.

That data is, of course, often carried in the query portion of the URI as a parameter. If it's not in the URI then it's in the payload, often as a www-url-encoded field submitted via HTTP POST method. The script can extract the URI and validate its total length and/or it can extract each individual name-value pairs (in the URI or in the body) and evaluate each one of *them* for length, doing whatever it is you want done with invalid length values: reject the request, chop the value to a specific size and pass it on, log it or even route the request to an application honey-pot.

> **Example** iRule snippet to check length of URI on submission: if {[string *length* $*uri*] > 1024} /* where $uri = HTTP::uri */

If you're thinking that it's going to be time consuming to map all the possible variables to maximum lengths, well, you're right. It will. You can of course write a generic across-the-board length-limiting script or you could investigate a web application firewall instead. A WAF will "learn" the mapping in real-time and allow you to fine-tune or relax limitations on a parameter by parameter basis if you wish, with a lot less time investment.

Both options, however, will do the job nicely and both provide a secondary line of defense in the face of a potential exploit that is completely avoidable if you've got the right tools in your security toolbox.

Related blogs and articles:

-

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