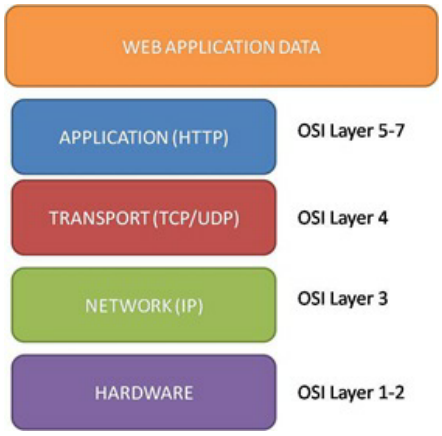
4 Reasons We Must Redefine Web Application Security

Lori MacVittie, 2009-11-03

Mike Fratto loves to tweak my nose about web application security. He's been doing it for years, so it's (d)evolved to a pretty standard set of arguments. But after he tweaked the debate again in a tweet, I got to thinking that part of the problem is the definition of web application security itself.

Web application security is almost always about the application (I know, duh! but bear with me) and therefore about the developer and secure coding. Most of the programmatic errors that lead to vulnerabilities and subsequently exploitation can be traced to a lack of secure coding practices, particularly around the validation of user input (which should never, ever be trusted). Whether it's XSS (Cross Site Scripting) or SQL Injection, the root of the problem is that malicious data or code is submitted to an application and not properly ferreted out by sanitization routines written by developers, for whatever reason.

But there are a number of "web application" attacks that have nothing to do with developers and code, and are, in fact, more focused on the exploitation of protocols. TCP and HTTP can be easily manipulated in such a way as to result in a successful attack on an application without breaking any RFC or W3C standard that specifies the proper behavior of these protocols. Worse, the application developer really can't do anything about these types of attacks because they aren't aware they are occurring.



It is, in fact, the four layers of the TCP/IP stack - the foundation for web applications – that are the reason we need to redefine web application security and expand it to include the network where it makes sense. Being a "web" application necessarily implies network interaction, which implies reliance on the network stack, which implies potential exploitation of the protocols upon which applications and their developers rely but have little or no control over. Web applications ride atop the OSI stack; above and beyond the network and application model on which all web applications are deployed. But those web applications have very little control or visibility into that stack and network-focused security (firewalls, IDS/IPS) have very little awareness of the application.

Let's take a Layer 7 DoS attack as an example. L7 DoS works by exploiting the nature of HTTP 1.1 and its ability to essentially pipeline multiple HTTP requests over the same TCP connection. Ostensibly this behavior reduces the overhead on the server associated with opening and closing TCP connections and thus improves the overall capacity and performance of web and application servers.

So far, all good.

But this behavior can be used against an application. By requesting a legitimate web page using HTTP 1.1, TCP connections on the server are held open waiting for additional requests to be submitted. And they'll stay open until the client sends the appropriate HTTP Connection: close header with a request *or* the TCP session itself times out based on the configuration of the web or application server.

Now, imagine a single source opens a lot – say thousands – of pages. They are all legitimate requests; there's no malicious data involved or incorrect usage of the underlying protocols. From the application's perspective *this is not an attack.*

And yet it *is* an attack. It's a basic resource consumption attack designed to chew up all available TCP connections on the server such that legitimate *users* cannot be served.

The distinction between legitimate users and legitimate *requests* is paramount to understanding why it is that web application security isn't always just about the application; sometimes it's about protocols and behavior external to the application that cannot be controlled let alone detected by the application or its developers.

The developer, in order to detect such a misuse of the HTTP protocol, would need to keep what we in the network world call a “session table”. Now web and application servers keep this information, they have to, but they don’t make it accessible to the developer. Basically, the developer’s viewpoint is from inside a single session, dealing with a single request, dealing with a single user. The developer, and the application, have a very limited view of the environment in which the application operates.

A web application firewall, however, has access to its session table necessarily. It operates with a view point external to the application, with the ability to understand the “big picture”. It can detect when a single user is opening three or four or thousands of connections and ***not*** doing anything with them. It understands that the user is not legitimate because the user’s behavior is out of line with how a normal user interacts with an application.

A web application firewall has the context in which to evaluate the behavior and requests and realize that a single user opening multiple connections is not legitimate after all.

Not all web application security is about the application. Sometimes it’s about the protocols and languages and platforms supporting the delivery of that application. And when the application lacks visibility into those supporting infrastructure environments, it’s pretty damn difficult for the application developer to use secure coding to protect against exploitation of those facets of the application.

We really have to stop debating ***where*** web application security belongs and go back to the beginning and redefine what it ***means*** in a web driven distributed world if we’re going to effectively tackle the problem any time ***this*** century.

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