Rip and Replace Won’t Solve Twitter’s (Or Your) Security Problems

Lori MacVittie, 2009-20-07

The "replace" in "rip and replace" essentially means getting rid of old security problems and replacing them with new ones.

Twittergate is (thankfully) behind us but it's almost assuredly going to be the case that we'll be rehashing this one for a while. This certainly isn't the first time Twitter and security issues have clashed, and as in the past Twitter (and really any very public application in a similar situation) is the clear loser. And of course there comes the unsolicited advice offered regarding what Twitter needs to do to address its security issues. I am, of course, ignoring the fact that it wasn't really even Twitter's security that was breached and thus led to the offering of said advice. But let's just pretend for a moment that Twitter still has security problems based on other, documented breaches in its security. You know, just like you and the 94% of other organizations out there that indicated they've experienced a breach in security in the last 6 months which goes along way toward validating that yes, you (and everyone else) has a security problem of some sort, it just might not yet have been exploited.

Now Stan Schroeder makes it sound so simple as he explains what Twitter (and by extension you and everyone else) needs to do to address its security-related problems:

> *One thing is certain. Twitter needs to burn everything security-related down to the ground and build it all anew to make sure this won't happen again.*

I'm trying very hard to ignore the fact that "it" didn't happen in the first place, and that if Stan – or anyone else – has some sort of crystal ball through which they can predetermine *how* a miscreant is going to twist and exploit an application in the future that it would be really nice to share that with the rest of us. That's why they call it "discovery" of an exploit; because no one *knew about it before then.* It's kind of hard, I've noticed, to put into place protections against vulnerabilities you don't know exist.

Okay, enough of that. Let's focus on the advice: *burn everything security-related down to the ground and build it all anew*.

If it were only that easy don't you think Twitter – and every other application out there with security problems – would have done that already?

Twitter, like most Web 2.0 sites and applications, *appears* to be highly integrated. Not in the application-to-application sense but in the sense that seemingly disparate application functions are tightly-coupled together. This is often true of enterprise applications, especially those that have evolved over time because security "fixes" are often integrated right into the vulnerable code at the function level. If the security is not a separate entity from the application logic making suggestions to essentially "rip and replace" is not only unrealistic but unreasonable as well. In fact the suggestion is quite unrealistic even if it's *not* security we're talking about. "Rip and Replace" has never been a winning strategy in the field of software development. As Joel Spolsky (Joel on Software) has emphatically said in the past:

> *They [Netscape] did it [lost its market force] by making the single worst strategic mistake that any software company can make:*

> *They decided to rewrite the code from scratch.*

He goes on to support his statement with other examples such as Borland and Microsoft's near-miss with Word. He goes on to point out that refactoring and re-architecture is a much better (and more likely to succeed) strategy than rip and replace. This is certainly true of software that isn't built with agility in mind; that tightly couples security with networking with presentation with data. It's also true of applications written "just because" that no one expects to become a tool on which millions of people rely.

As Joel points out, replacing code with new code introduces the very real possibility of ***new*** errors, of new vulnerabilities, and new problems. The old code has been tested, improved, and retested. You know what's wrong with it and where the problems lie. Ripping it out and replacing it means you have to start over, from scratch, and figure out where all the problems with ***that*** code lies – from functionality to security to performance. It's not just the time and effort required to rebuild from scratch, it's the long, tedious and time-consuming process of stabilizing that code over time that makes it a risky and expensive proposition.

WHAT HAPPENED TO SOA, WOA, and ABSTRACTION?

This was something SOA was supposed to solve. Placing security as another "tier" in the architecture provided the means by which meta-data driven security policies and functions could be easily applied to any service without the lengthy and difficult task of disentangling the security code from the application code first. WOA (Web Oriented Architecture) was supposed to be the marriage of SOA and Web 2.0, but it seems Web 2.0 got cold feet and left SOA standing at the altar because modern Web applications seem to ignore the benefits of a service-oriented architecture and just start building without really considering the long term implications. Oh, the APIs are RESTful and provide the appearance of a well-designed, abstracted architecture but covering a troglodyte with a pretty dress doesn't change the ugly underneath.

Why do apps grow that way? Perhaps because there are no guarantees on the web. Twitter could have become mired in obscurity like its wanna-be cousin, Plurk, just as easily at it could have become the darling of the Internet. It's unlikely the founders and creators of Twitter ever considered how popular the site would become, and thus their interest was in building something cool and useful, not necessarily something that was air-tight secure ***especially*** as they could not have foreseen the ways in which Twitter might be leveraged in the future. Twitter – like many Web 2.0 applications that appear out of nowhere and skyrocket to stardom - almost certainly wasn't designed with the need for separation of functionality in mind. Security and privacy functionality are often intertwined in the code, like many enterprise applications, and that makes it nigh impossible to strip out and replace it with an alternative.

And while refactoring code as a means to achieve results similar to a "rip and replace" effort is certainly a viable methodology, it is only efficient and effective if there are chunks of functionality that can be refactored and replaced with functions/methods throughout the application. If the code is not ***exactly the same*** then refactoring is not really the panacea it's made out to be.

Enterprise applications face even more challenging issues as applications become "mission-critical" and myriad developers and architects put their fingers into the code over years. The code becomes bloated as fixes and changes are not always applied with agility to change in mind; rather the code is tweaked wherever the problem may lie, which is often inside a given function or method. Over time this makes the code tightly-coupled, with no real easy way to separate out the "security" code from the application logic.

THERE'S MORE EFFICIENT MEANS OF SKINNING THE SECURITY CAT

Application delivery platforms are one of the easiest ways in which to address security and overall delivery / performance concerns because they do not require modification of the application code. Need to add cookie encryption/decryption? Need to rewrite a URI? Redirect a user? Add client authentication that integrates with existing or new identity management systems? Need to add a way to scan requests for malicious code?

An application delivery platform can provide the application-specific functionality in a way that does not require the application code to be changed and in doing so removes the long-term ramifications of tightly-coupling security functionality to application code. Because it's implemented externally, such functionality can be applied to multiple applications – not just one – without the time and cost associated with updating, testing, and redeploying ***each and every*** application. An application delivery platform essentially introduces the abstraction that ***should*** have existed in the first place but often does due to any number of factors that affect application development.

This is ***not*** just a "WAF". While that's certainly one of the features available on an application delivery platform, what we're talking about here is a highly flexible, programmatic method of implementing application logic on an external application delivery platform. It's called network-side scripting and it provides an alternative, often more efficient, platform for deploying specific kinds of application services: usually those involved with inspection, extraction, and transformation of HTTP headers and application data.

What does network-side scripting bring to the table? The ability to apply application functionality across all applications in an immediate, agile way. It also provides the means by which vulnerabilities can be immediately addressed (a la virtual patching) and gives developers the ***time*** needed to determine the best course of action regarding the vulnerability without leaving the application vulnerable to exploitation.

Does Twitter need to address security problems? Sure. Everyone does. But the suggestion to simply burn it to the ground and start-over is more than a bit naive. The reality is that the ramifications of a "rip and replace" methodology are almost certainly so negative as to be a non-option for Twitter and enterprises in general. There are other options, depending on what the specific security issues may be. In some cases "rip and replace" may well and truly be the ***only*** option, but it's unlikely and in all cases would be a major undertaking likely to take ***years*** to accomplish. Not months. Not weeks. Not days. ***Years***. In other cases it may require ingenuity and innovative thinking to come up with a solution that works for the specific application.

That's part of the problem with the advice being handed out to Twitter; until we understand ***what*** the security issues may be from a technical perspective it's impossible to provide any really useful advice to Twitter other than "you really do need to fix it."

And I'm guessing they already know that.

- Forklifts, Rip and Replace, and Other IT Fairy Tales
- Twitter's Security Meltdown (Mashable)
- Dear Plurk: We're Through. Kthxbye.
- Things You Should Never Do, Part I (Joel on Software)
- Advanced Load Balancers for Developers: ADCs - The Code
- True or False: Application acceleration solutions teach developers to write inefficient code
- 9 Ways to use network-side scripting to architect faster, scalable, more secure applications
- Understanding network-side scripting

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