Software Insecurity Thrives

Michael Barwise

The fourth half-yearly State of Software Security Report from Cloud-based application security tester Veracode makes painful reading. Based on analysis of 9910 application builds over the last 18 months, Veracode found that over 80 per cent of them were vulnerable to one of more of the OWASP Top 10 consensus list of the most prevalent security-related coding flaws. For web applications, cross-site scripting (XSS) dominated any other specific technical vulnerability, affecting 68 per cent of the tested applications. Second and third place were taken by CRLF injection and SQL injection, found in 54 and 32 per cent of applications respectively. Incidentally (or maybe not) more government-developed applications contained XSS vulnerabilities than those of any other sector.

Veracode note that these figures do not tally with those of the Web Hacking Incident Database, which is compiled from reported incidents and sees SQL injection as more prevalent than XSS. But I argue that incident reports are necessarily a less reliable source of real vulnerability prevalence than the examination of code, as they only include those that have already been both exploited and reported. This makes me consider the Veracode data as more representative of the true state of application insecurity, since the most dangerous vulnerability is the one you don't know about yet.

As well as coming top of the susceptible applications count, XSS also constituted 57 per cent of all detected vulnerabilities. However, "insufficient input validation" comes much lower down the Veracode ranking, as a completely separate category with only 24 per cent of applications deemed susceptible. But a successful XSS attack depends entirely on inadequate validation of maliciously crafted input. And incidentally, as adequate input validation would fix CRLF injection and pretty much all of SQL injection as well, it would seem to me to be the highest priority of all. Nevertheless, although inadequate validation is thus the real root cause of the most significant cluster of vulnerabilities as a whole, Veracode, and indeed the industry at large, clearly fail to recognise this. To me this speaks - indeed shrieks - of failure to acknowledge the significance of first principles.

The Veracode report's section on developer training is of particular interest. Veracode is here not an entirely disinterested party as it provides e-learning courses for developers. Nevertheless the figures do speak for themselves. Aggregating the top two of Veracode's five result grades yields a minimum pass mark of 80%, which is about as low as I would accept in a practitioner examination for something as critical as secure software development. After all, getting more than a fifth of a mission-critical job wrong isn't really on, is it? Against this (admittedly personal but reasonable) criterion, 66 per cent qualified in Secure Coding for .Net and 52 percent in Secure Coding for Java. In Applications Security Fundamentals, however, only 45 per cent reached this standard. Looking at it the other way up, a third of candidates failed in .Net security and nearly half failed in Java security, but the worst failure rate of all was in security fundamentals. This again smacks of inadequate understanding of principles, but I wonder how many of those who failed these tests considered resigning from their programming jobs.

A competent engineer doesn't just create something that works, but something that both works and is proof against all anticipated modes of failure. So if we want to bring the escalating online threat under some measure of control, we must raise the population of software developers from the level of rote trained mechanics to that of competent engineers. Instead of primarily assembling from stock modules the equivalent of wobbly flat-pack furniture that falls apart at the slightest push, they must be capable of creating applications and services in which we can safely place our trust. There's a good reason why we don't build aircraft and bridges the way we write software - the stakes are well recognised to be too high in event of failure. But although it may not be quite so obvious, the stakes can be just as high these days if software is breached, so why do we accept such low standards? After all, the OWASP Top 10 are not rocket science. They're well known flaws, most of which have been around for years and are quite simple to avoid provided one understands the principles, is paying attention, and is prepared to make the effort. So is it acceptable that the top three of the top 10 are still present in over 80 per cent of our code?

The 2011/12 CAST Report on Application Software Health illuminates the problem from a different angle but to much the same effect. Based on examination of 745 applications submitted by 160 organisations and representing some 365 million lines of code, this research found that, on average, COBOL code was the most robust, that .Net code was quite a bit less secure than C++ code, and Java code only marginally more secure than C++. It also found that the level of insecurity did not correlate

with object size or complexity. So it would seem that in general the more abstracted - and more recent - the development environment, the poorer the security of the code. This might seem counter-intuitive at first sight, but it supports my argument that developers increasingly assume the language, libraries and environment they are using will do all the difficult bits for them automatically, so they don't need to contribute independently to the security of their code. I'm not alone in coming to this conclusion. In an interview with Computerworld, CAST's chief scientist Bill Curtis explained *"We may just be seeing the fact that there is an awful lot of people writing code who aren't gurus in software engineering"*. But we don't actually need gurus. We need all our developers to be competent in the basics of software engineering, and considering that the worldwide sustainability of commerce, government and critical utilities relies heavily on the code they produce, we need it now.

I am convinced that developer training must instil a solid grounding in first principles - not the rather ethereal and abstract "computer science" currently widely taught in universities, but a solid understanding of the fundamentals of robust algorithm design and the principles of defensive coding. These should be taught entirely independently of the facilities offered by any specific language or development system. Professional developers must be able to produce code consistently that does what it is expected to do and nothing else, regardless both of the language they are using and what will be thrown at their code as input. There should not be, as at present, a separate discipline of "secure coding" - it must become the only way anyone is taught to design and code software.

Originally appeared on the Infosecurity Network, December 2011