Herding Cats:
got sprintf()?

November 2009

BRANDENWILLIAMS
SECURE BUSINESS GROWTH

Job security. It's a nice thing to have, isn't it? I mean, anyone reading this month's column has to realize that security professionals' jobs are only getting more difficult, and our talents and skills will steadily grow in demand as we continue to the end of the world[1]. All you need to do is read the Verizon Business Data Breach Report[2], and you will see that not only are the numbers of breached records and incidents increasing, but in many cases hackers are using some of the same old attacks.

Application security is sometimes one of those "I'll get to it" projects that never seems to get off the ground until a data breach forces the issue. Back when I did application development, the lifecycle looked something like this.

Step 1: Have idea for software
Step 2: Build proof of concept
Step 3: Polish proof of concept
Step 4: Go to production

This is a slight exaggeration, but my reality prevented me from hiring someone to comb through the application looking for security vulnerabilities.

I wrote some code for a startup years ago that would display an invoice. We licensed a fancy barcode generator to make the order fulfillment process easier. All our folks had to do was scan the barcode that appeared at the top of the invoice, and a shipping label was automatically generated and printed. Unfortunately, this code stayed inside the application long after I moved on because it served its basic purpose. At the time I wrote this front end (VBScript ASP 1.0), I spent more of my free time busting buffers in poorly written C code. Those types of vulnerabilities were not present in the environment I wrote[3].

Here's where I screwed up. When presented with the order confirmation/status page once, you could change the order number in the GET string to see another order. Customer's didn't notice, but one competitor did and attempted to steal the customer base! Thankfully, the current development manager noticed funny activity in the logs and stopped the attack.

That simple application vulnerability could have lead to much worse. I'm certain that same code was vulnerable to SQL Injection. In the late 1990s hackers successfully executed SQL Injection attacks, but the problem was not nearly as widespread as it is today. According to the Verizon report, 23% of the breaches and 79% of the data loss included successful SQL Injection attacks (page 17). Verizon asserts that SQL Injection is increasing in complexity, number of attempts, and success rate for large scale breaches.

Six of the ten types of hacks listed in the report are application security related[4], and easily preventable with proper application security coding techniques. So why are these same attacks that have been around for years still successful in today's code bases?

FOOTNOTES

[1] Which according to some interpretations of the Mayan calendar occurs during the winter solstice in 2012, when the sun will perfectly align with the center of the Milky Way for the first time in 26,000 years.
[2] This report can be downloaded from http://www.verizonbusiness.com/resources/security/reports/2009_databreach_rp.pdf, and will be referenced in various places throughout.
[3] Maybe in the underlying system, but not at the abstraction layer in which I was coding.
[4] I left out buffer overflows, but arguably they could be thrown in to make it seven.

Part of the problem is how people are taught to write code. In some programming languages, security is built-in via abstraction. In other cases, developers lean too far toward the functionality side of the Security vs. Functionality continuum. Some of it is lazy coding, and others are an “Under the deadline at 4am and only injected caffeine will keep my fingers moving” mistake. One of those, “The printf() call was shorter and took less arguments than the sprintf() one” moments that we wish never happened when things blow up. And we totally promise to fix it in the next release.

I’ve had an interesting debate with some of my customers lately. Application code review is more expensive than application scanning or application penetration testing, but could it be more valuable? Finding vulnerabilities with static and dynamic code analysis can be fun and can yield actionable results much faster than a consultant working with a proxy can.

Do yourself a favor and find someone you trust to take a swim through your application code. The results will be enlightening, and the actions you take based on the report will close gaping holes in your security posture. Then be sure to educate your developers and check up on their code to make sure they learned from their mistakes! Track bugs with a sad-face sticker chart if you have to, but be sure to let your developers know when they introduce vulnerabilities into their code BEFORE it hits production.

<u>About the Author:</u>

Branden R. Williams, CISSP, CISM, CPISA/M, has been making a name for himself in the Information Technology and Security arena since 1994, as a high school Junior. Now, a graduate of University of Texas, Arlington earning his BBA in 2000 with a concentration in Marketing and the University of Dallas, where he earned an MBA in Supply Chain Management & Market Logistics, in 2004, Williams is sought after as both an Adjunct Professor and Information Technology & Security Strategy Leader in the corporate world.

Williams regularly assists top global retailers, financial institutions, and multinationals with their information security initiatives. Read his blog at or reach him directly at http://www.brandenwilliams.com/.

TEL 214 727 8227
FAX 214 432 6174

BLOG brandenwilliams.com

EMAIL brw@brandenwilliams.com

BRANDENWILLIAMS
SECURE BUSINESS GROWTH