freedom-to-tinker.com

Software Security: The Trinity of Trouble

Gary McGraw

8-11 minutes

[Ed Felten says: Please welcome Gary McGraw as guest blogger for the next week. Gary is CTO at Cigital and co-author of two past books with me. He's here to post excerpts from his new book, Software Security: Building Security In, which was released this week. The book offers practical advice about how to design and build secure software – a problem that is hugely important and often misunderstood. Now here's Gary….]

The Trinity of Trouble: Why the Problem is Growing

Most modern computing systems are susceptible to software security problems, so why is software security a bigger problem now than in the past? Three trends—together making up the trinity of trouble—have a large influence on the growth and evolution of the problem.

Connectivity. The growing connectivity of computers through the Internet has increased both the number of attack vectors and the ease with which an attack can be made. This puts software at greater risk. More and more computers, ranging from home PCs to systems that control critical infrastructure, such as the supervisory control and data acquisition (SCADA) systems that run the power grid, are being connected to enterprise networks and to the Internet. Furthermore, people, businesses, and governments are increasingly dependent on network-enabled communication such as e-mail or Web pages provided by information systems. Things that used to happen offline now happen online. Unfortunately, as these systems are connected to the Internet, they become vulnerable to software-based attacks from distant sources. An attacker no longer needs physical access to a system to exploit vulnerable software; and today, software security problems can shut down banking services and airlines (as shown by the SQL Slammer worm of January 2003).

Because access through a network does not require human intervention, launching automated attacks is easy. The ubiquity of networking means that there are more software systems to attack, more attacks, and greater risks from poor software security practices than in the past. We're really only now beginning to cope with the ten-year-old attack paradigm that results from poor coding and design. Ubiquitous networking and attacks directly related to distributed computation remain rare (though the network itself is the primary vector for getting to and exploiting poor coding and design problems). This will change for the worse over time. Because the Internet is everywhere, the attackers are now at your virtual doorstep.

To make matters worse, large enterprises have caught two bugs: Web Services and its closely aligned Service Oriented Architecture (SOA). Even though SOA is certainly a fad driven by clever marketing, it represents a succinct way to talk about what many security professionals have always known to be true: Legacy applications that were never intended to be internetworked are becoming inter-networked and published as services.

Common platforms being integrated into megasolutions include SAP, PeopleSoft, Oracle, Informatica, Maestro, and so on (not to mention more modern J2EE and NET apps), COBOL, and other ancient mainframe platforms. Many of these applications and legacy systems don’t support common toolkits like SSL, standard plug-ins for authentication/authorization in a connected situation, or even simple cipher use. They don’t have the builtin capability to hook into directory services, which most large shops use for authentication and authorization. Middleware vendors pledge they can completely carve out the complexity of integration and provide seamless connectivity, but even though they provide connectivity (through JCA, WBI, or whatever), the authentication and application-level protocols don’t align.

Thus, middleware integration in reality reduces to something ad hoc like cross-enterprise FTP between applications. What’s worse is that lines of business often fear tight integration with better tools (because they lack skills, project budget, or faith in their infrastructure team), so they end up using middleware to FTP and drop data globs that have to be mopped up and transmogrified into load files or other application input. Because of this issue, legacy product integrations often suffer from two huge security problems:

1. Exclusive reliance on host-to-host authentication with weak

passwords
2. Looming data compliance implications having to do with user privacy (because unencrypted transport of data over middleware and the middleware's implementation for failover and load balancing means that queue cache files get stashed all over the place in plain text)

Current trends in enterprise architecture make connectivity problems more problematic than ever before.

Extensibility. A second trend negatively affecting software security is the degree to which systems have become extensible. An extensible system accepts updates or extensions, sometimes referred to as mobile code so that the functionality of the system can be evolved in an incremental fashion. For example, the plug-in architecture of Web browsers makes it easy to install viewer extensions for new document types as needed. Today's operating systems support extensibility through dynamically loadable device drivers and modules. Today's applications, such as word processors, e-mail clients, spreadsheets, and Web browsers, support extensibility through scripting, controls, components, and applets. The advent of Web Services and SOA, which are built entirely from extensible systems such as J2EE and .NET, brings explicit extensibility to the forefront.

From an economic standpoint, extensible systems are attractive because they provide flexible interfaces that can be adapted through new components. In today's marketplace, it is crucial that software be deployed as rapidly as possible in order to gain market share. Yet the marketplace also demands that applications provide new features with each release. An extensible architecture makes it easy to satisfy both demands by allowing the base application code to be shipped early, with later feature extensions shipped as needed.

Unfortunately, the very nature of extensible systems makes it hard to prevent software vulnerabilities from slipping in as unwanted extensions. Advanced languages and platforms including Sun Microsystems' Java and Microsoft's .NET Framework are making extensibility commonplace.

Complexity. A third trend impacting software security is the unbridled growth in the size and complexity of modern information systems, especially software systems. A desktop system running Windows XP and associated applications depends on the proper functioning of the kernel as well as the applications to ensure that vulnerabilities cannot compromise the system. However, Windows XP itself consists of at least forty million lines of code, and end-user applications are becoming equally, if not more, complex. When systems become this large, bugs cannot be avoided.

The figure above shows how the complexity of Windows (measured in lines of code) has grown over the years. The point of the graph is not to emphasize the numbers themselves, but rather the growth rate over time. In practice, the defect rate tends to go up as the square of code size. Other factors that significantly affect complexity include whether the code is tightly integrated, the overlay of patches and other post-deployment fixes, and critical architectural issues.

The complexity problem is exacerbated by the use of unsafe programming languages (e.g., C and C++) that do not protect against simple kinds of attacks, such as buffer overflows. In theory, we could analyze and prove that a small program was free of problems, but this task is impossible for even the simplest desktop systems today, much less the enterprise-wide systems used by businesses or governments.

Of course, Windows is not alone. Almost all code bases tend to grow over time. During the last three years, I have made an informal survey of thousands of developers. With few exceptions (on the order of 1% of sample size), developers overwhelmingly report that their groups intend to produce more code, not less, as time goes by. Ironically, these same developers also report that they intend to produce fewer bugs even as they produce more code. The unfortunate reality is that â€œmore lines, more bugsâ€ is the rule of thumb that tends to be borne out in practice (and in science, as the next section shows). Developers are an optimistic lot.

The propensity for software systems to grow very large quickly is just as apparent in open source systems as it is in Windows. The problem is, of course, that more code results in more defects and, in turn, more security risk.