



Security Band-Aids: More Cost-Effective than “Secure” Coding

Greg Hogland

Patching systems against the latest virus is a full-time job, and most corporations have heavier near-term problems facing them.

The war between hackers and software is being fought on the front lines—in the users’ trenches. But hunting down the “engineers” who write bad software won’t win this war, at least not in the short run. With the best of intentions, development shops are trying to address bad software by learning secure coding practices.

Just tracing the problem is difficult enough. In many cases, the system’s original developers are long gone. And even if they’re still around, the application has evolved continuously since they first wrote it. Moreover, companies often inherit applications (through mergers and acquisitions) that have their own unique problems. Even simple software gives rise to incredibly complex behavior,¹ and with complexity comes failure. (Over 70 percent of all software projects fail.²)

Although the right place to solve software problems is in development, most companies do not have the luxury of time or money to rebuild old code. So, most developers try to fix problems in the easiest way possible, which usually translates to cheapest, not best. Apply-

ing a software patch costs far less than, say, eliminating all buffer overflows from your code.

Moreover, patching systems against the latest virus is a full-time job, and most corporations have heavier near-term problems facing them—their numbers for the next few quarters. Timesaving point solutions such as application firewalls have an instant return on investment. And consider the human element. Typically, developers throw their code “over the wall” to an understaffed security department, which is seen as a roadblock to progress. Required to approve hundreds of changes per week, these organizations often let stuff slide through.

The developer’s world is far removed from the true universe of deployed software—a hostile, overcrowded network full of threats and unknowns. Furthermore, most software engineers are actually construction workers—just like the so-called civil engineers who dig up freeways. Many people who write code have upgraded their paychecks from other disciplines by obtaining <fill in vendor>

Continued on page 30



Building Secure Software: Better than Protecting Bad Software

Gary McGraw

Software has become essential to business and to many other aspects of our daily lives. Yet creating software that works remains hard, especially where security and reliability are concerned. Trying to protect software from attack by filtering its input and constraining its behavior in a post facto way (application security) is nowhere near as effective as designing software to withstand attack in the first place (software security). Simply put, we can't bolt security to the side of a software product.

Software is the biggest problem in computer security today.¹ Most organizations invest in security by buying and maintaining a firewall, but they go on to let anybody access multiple Internet-enabled applications through that firewall. These applications are often remotely exploitable, rendering the firewall impotent (not to mention the fact that the firewall is often a piece of fallible software itself). Real attackers exploit software.

By any measure, security holes in software are common, and the problem is growing. The *trinity of trouble* exacerbates the problem of insecure software:

- Modern software operates in a hostile networked environment.
- Extensible systems such as Java virtual machines and .NET common runtime environments (not to mention dynamically loaded libraries) are becoming common and introduce mobile code risks.
- System complexity is rising.

The ultimate answer to the computer security problem clearly lies in making software behave. The question at hand is "What is the most effective way to protect software?"

We can cleanly divide the software/application security space into two distinct subfields. *Software security* is about building secure software. Issues critical to this subfield include software risk management, programming languages and platforms, software audits, designs for security, security flaws, and security tests. Software security is mostly concerned with designing software to be secure, making sure that software is secure, and educating software developers, architects,

Simply put, we can't bolt security to the side of a software product.

Continued on page 30

certifications; although their education is good, it doesn't guarantee they will stop writing bad code. And, with the development tools available today, they fight an uphill battle to write secure code. Sadly, quality assurance testing today is inadequate to overcome most engineers' skill levels and limited tools. Thus, the potential for security failure in hostile environments remains high.

Band-aid security—consisting of using shunts and limiters on data input—could be the answer. Band-aids do not fix the disease: they protect the wound. They are designed to “detect the bad,” but they do nothing to stop the threat of an unknown attack. Think of a facial recognition burglar alarm with a camera that operates by the side of a speeding freeway. It can scan only one out of 10 cars, and some of the criminals are hiding in the trunk. This is the sort of risk you must live with in a band-aid system.

Because we'll uncover new threats “in the wild,” band-aid security will prove itself useful time and again. Moreover, deploying such systems is incredibly easy; people who don't really have a clue about software can manage them. Such systems do, however, require maintenance. They are knowledge-driven devices that require a steady diet of information about new exploits.

Will deploying band-aid systems reduce your vulnerability? I could argue that it won't, but it's difficult to do so when a deadly virus rips through your network, and a band-aid is the only tool you have to stop it. Any case study of recent viruses Nimda or Code Red will prove the value of these tools. Using band-aids does nothing to prevent the attack from a solitary hacker, but they do protect you from the masses of idiots who could download that exploit and target your systems. A band-aid

could also protect you from a worm's brainless automaton.

Application security tools are the most effective way your organization can protect itself today. Building more secure software is a goal, but it won't stop the virus that gets released tomorrow. It comes down to this: secure coding practices are not going to produce 100 percent bug-free software: application security tools should always play a part in your risk mitigation plan.

References

1. S. Wolfram, *A New Kind of Science*, Wolfram Media, Champaign, Ill., 2002.
2. The Standish Group, *CHAOS: A Recipe for Success*, The Standish Group, West Yarmouth, Mass., 1999; www.pm2go.com/sample_research/chaos1998.pdf.

Greg Hoglund is chief technology officer and cofounder of *Cenzic*, a security quality assurance company. Contact him at greg@cenzic.com.

and users.

Application security is about protecting software and the systems that the software runs after development is complete. Issues critical to this subfield include sandboxing code, protecting against malicious code, locking down executables, monitoring programs (especially their input) as they run, enforcing software use policy with technology, and dealing with extensible systems.

Application security follows naturally from a network-centric approach to security (embracing standard approaches such as penetrate-and-patch² and input filtering), and providing value

in a reactive way. Put succinctly, application security is primarily based on finding and fixing known security problems only after they are exploited in fielded systems. However, this approach addresses security symptoms in a reactive way, ignoring the problem's root cause. In general, application security takes the same approach to security as firewalls do. In fact, application security vendors often refer to their products as “application firewalls.” Although there is value in stopping buffer overflow attacks by observing HTTP traffic as it arrives over port 80, a superior approach is to fix the broken code and avoid the buffer overflow completely.

Software security—the process of designing, building, and testing software for security—gets to the heart of the matter by identifying and expunging problems in the software itself. In this way, software security attempts to build software that can withstand attack proactively. Software security follows naturally from software engineering, programming languages, and security engineering.

Both subfields are relevant to the idea of preventing software's exploitation. Software security defends against exploit by building the software to be secure in the first place, mostly by getting the design right

(hard) and avoiding common mistakes (easy). The process of securing applications defends against software exploit by enforcing reasonable policy about what kinds of things can run, how they can change, and what software does as it runs.

In the fight for better software, treating the disease itself (poorly designed and implemented software) is better than taking an aspirin to stop the symptom. There is no substitute for working software security as deeply into the software development process as possible and taking advantage of the engineering lessons software practitioners have learned over the years.

Good software security practices can help ensure that software behaves properly. Safety-critical and high-assurance system designers have always taken great pains to analyze and track software behavior. Security-critical system designers must follow suit. We can avoid the band-aid-like penetrate-and-patch approach to security only by considering security as a crucial system property. This

requires integrating software security into the software engineering process.³

References

1. G. McGraw, "Software Assurance for Security," *Computer*, vol. 32, no. 4, Apr. 1999, pp. 103–105.
2. G. McGraw, "Testing for Security During Development: Why We Should Scrap Penetrate-and-Patch," *IEEE Aerospace and Electronic Systems*, vol. 13, no. 4, Apr. 1998, pp. 13–15.
3. J. Viega and G. McGraw, *Building Secure Software*, Addison-Wesley, New York, 2001; www.buildingsecuresoftware.com.

Gary McGraw is chief technology officer at Cigital, a software quality management consultancy. Contact him at gem@cigital.com.

Greg Responds

I fundamentally agree with most of Gary's arguments—most importantly, that software is the root of computer security problems. I also agree with the "trinity of trouble" and the distinction between application and software security: not only is system complexity rising, the system is growing more interconnected. This means that software bugs can result in cascading failures across the system, which spells trouble for most companies because the system includes the environment and all the software that it communicates with.

We'll never be able to simulate real-world complexity in the lab or on the developer's desktop. By this argument, software developers will never have the opportunity to explore what failure really means. Everything will be a close approximation. Thus, it would seem that Gary and I are debating the effectiveness of approach. I certainly agree that the strongest way to fix a problem is to cure the disease—that is, to fix the software in development. This is something that businesses can and should afford. However, I don't agree that fixing problems in development is the only solution. I'll go out on a limb and suggest that software development alone will never fully solve the problem. The playing field is hostile, and there will always be fire-fights. Based on costs, many problems will first be solved by the IT department before going to development. Advanced development tools will soon eliminate trivial grammatical errors (such as buffer overflows), but complex problems take time to address. The band-aid extends the time cushion. That being said, I applaud anyone who actually takes advantage of that time to implement secure coding practices.

Gary Responds

Microsoft's highly touted Trustworthy Computing Initiative, spurred by the Gates memo of January 2002, is a direct business-driven response to a changing software market. Software users now demand high-quality software that works. Of course, software security reaches far beyond shrink-wrapped software of the sort that Microsoft produces.

Building software to be secure and reliable from the start is cost effective. TRW reports that the cost of fixing software defects in late life-cycle stages (testing and maintenance) is over US\$10,000 per fix, whereas the cost of fixing a defect early in the life cycle (requirements, design, and coding) is an order of magnitude less—under \$1,500. The pervasive "penetrate and patch" approach to security is obviously unacceptable from a business standpoint. We must avoid the problem of desperately trying to come up with a fix to a problem that attackers are actively exploiting.

Those software users who cannot directly impact software quality by building things properly can and should use application security technologies that attempt to protect fielded software. COTS software has problems too, and application security technologies can protect bad software from some kinds of harm.

But postponing the hard work of building better software will not solve the problem. With software complexity growing—the source code base for Windows XP is 40 million lines—we have our work cut out for us. Building secure software requires educating developers and architects, retraining QA, and making better business decisions about security. CERT reports that the number of reported software vulnerabilities has risen from less than 500 a year for all years prior to 1999 to over 1,000 in 2000 and almost 2,500 in 2001. There are not enough band-aids to stop the bleeding with a laceration of this size.