

Securing your IT infrastructure

from Klocwork, a Ready-for-Rational partner

Gwyn Fisher, CTO, Klocwork

Summary: from The Rational Edge: Secure programming, as well as the practice of securing an enterprise's IT infrastructure, is an exercise in risk management. There is no single solution to all possible vulnerabilities, but knowing what solutions are available, their limitations, and where they fit in the spectrum of possible risk mitigation activities will put you far ahead of the pack. This content is part of the [The Rational Edge](#).

Date: 15 Sep 2007

Level: Introductory

Also available in: [Chinese](#)

Activity: 1383 views

Comments: 0 ([Add comments](#))

★★★★☆ Average rating (based on 4 votes)

Security is perhaps the hottest and most confusing catch-all word in information technology today. The myriad risks and associated solutions being presented to security professionals and IT managers alike can be overwhelming. As technology professionals, it is incumbent on us to identify where our IT infrastructure may have some risk, what solutions can address these problems, and to ultimately realize -- just as in the realm of physical security -- that there will never be a single answer to every security problem.



Surprisingly, only recently have security and IT professionals begun to realize that secure applications are mandatory when hardening a company's resistance to attacks. Whether you are driven by the conclusions and recommendations of leading security analysts, various regulatory guidelines, or the threat of brand damage associated with your vulnerabilities being made public, the end goal is the same: You want to remove security risks from your software before they can be exploited.

To learn the extent of the IT industry's security problems, simply read the daily security news bulletins for anecdotal evidence or visit a reputable, independent security organization like CERT (Computer Emergency Response Team) to view the latest statistics on exploited applications. Doing so will quickly expose the reader to the magnitude of this industry epidemic as measured by the growing number of known application vulnerabilities along with the resulting attack vectors such as code injection, input hijacking, blind rerouting and privilege escalation, as well as more traditional problems such as worms and viruses.

Software risk and corporate exposure

Historically, the risks associated with application development were fairly well contained. If the software did not operate as expected then a patch was created and applied with the fervent hope that the

problem would be taken care of. Rinse and repeat until either the problem was solved or the software vendor was removed from the equation.

In a world of protected internal networks and walled-off access points, this approach to software security and quality in general has long been viewed as acceptable, if not perhaps completely desirable, giving rise to the well-known and commercialized "ship, fix, patch" cycle of pragmatic software development.

In the world of connected computing, however, such risk-laden practices are completely unacceptable, and as recent high-profile cases have shown, they can easily turn into inordinately expensive and corporately-embarrassing situations when exploited. Spending hundreds of millions of dollars to redress flawed security assumptions in application software (what the cynical amongst us might term "bugs") is plainly unacceptable by anybody's definition of the word.

Every corporate worker in the world is armed with enough computing firepower to bring an organization to its knees, often with the most well-intentioned or naïve assumptions imaginable. Being able to perform order entry via the Web, for example, is a convenience enjoyed by many companies' workers. That is, until that order entry system is subverted by a black hat to ship customer credit details to some illegal operative.

The net toll of such occurrences on an enterprise is difficult to quantify, but what's certain is that as the security risks unfold, the impact mounts ever higher in terms of corporate accountability, diminished brand value, and reduced customer confidence.

The case for integrating risk management strategies with software development practices

For many development organizations, risk management as a discipline applied to secure programming is a relatively new idea. In the past, the exposure profile of any software product was limited to customer-internal networks, to protected and locked-down environments, and to a set of individuals who were being trusted with far more than just a new piece of code. With the advent of public network computing, however, even the most secure boundary protocols are subject to the vagaries and risks inherent in the software running behind, in front of, or through that boundary.

Let's use a real-world prison compound as a metaphor for the task of securing enterprise IT. The most obvious form of security is that which surrounds the compound itself in the form of a wall or barbed wire fence. Within that outer barrier another layer of security is deployed using regular patrols performed by guards and dogs. Finally, within the prison building itself are the inmates, under constant supervision and guidance as to what they can and can't do at any particular time of day.

All of these precautions build on the common requirement of managing the risk of any inmate being able to escape from the locked-down environment, or of anybody from the outside being able to reach into the prison and extract one or more of the inmates.

If we replace the nomenclature of that real-world prison compound with the virtual world of IT-delivered security, we can easily equate the outer barrier to the well-known and commercialized field of intrusion prevention -- firewalls, routers, Virtual Private Networks (VPNs), etc. Similarly, the patrols and guard dogs might well be equated to the software deployed within an enterprise network that is tasked with keeping that network "clean": e.g. anti-virus, anti-spyware, packet sniffers, bandwidth monitors, etc.

What we haven't yet covered is the prison building and the challenge of ensuring the security and

containment of its inmates. In other words, the applications running the enterprise's IT practice are in need of attention within our metaphor.

As countless films and written accounts remind us, the strongest outer defenses imaginable can be subverted by a prisoner determined enough to tunnel through to the outside world. But replace the word "determined" in that sentence with the word "naïve" and we begin to approach engagement with the world of application development with our eyes open. The sad fact is that most application developers have no idea whatsoever as to the risk profile of the software that they produce.

Sharing the risk responsibility for software -- it's not just an IT problem

Returning to our prison compound metaphor, increasing the height of the wall, or the aggregate wattage of the search lights, or the number and training level of the guards, does nothing to lower the likelihood of escape if the prison building simply has a dirt floor that can be dug and tunnelled into.

Similarly, increasing the power of the firewall or other intrusion detection mechanisms does nothing to help the actual security of the enterprise if applications that maintain holes, or trusted ports, through the firewall boundary are open to compromise.

Therefore, the responsibility to minimize the risks inherent in deploying software applications to corporate networks falls not only to the IT department and its ability to provide for a clean network environment, but also on the shoulders of the development organization producing those applications. After all, if the risk profile of any application being deployed is unknown, then the entire security profile of the enterprise is essentially compromised.

Why applications aren't secure

The unfortunate reality of life within a software development organization is that most people within that organization are completely ignorant of almost every aspect of the software's construction. This applies just as well to senior as to junior talent. Software construction has become an increasingly specialized field of endeavor, with developers in one area of the application having little or no knowledge of the capabilities or exposures of other areas, except as expressed via interface documentation.

It is common for architects of database interaction logic, for example, to have no knowledge of how their logic is specifically expressed via user interface code. In the race to bring new features to market faster than the next guy, requirements become increasingly complex, resulting in less time for senior staff to audit the subtle and unforeseen interactions that are a reality of large system design.

Couple this accelerating race and the pressure it forces upon an organization with the fact that very few application developers know very much about the attacker mind set, and it's easy to see the net result. Poor assumptions, naïve coding, and a motivated attacker make dangerous bed fellows.

What can be done

If we accept that corporate networks can't be secure as long as the applications running on those networks have an unknown risk profile, it behooves us to learn how to measure and manage that risk. Once we can measure the risk profile of any given application, we need to build processes and technologies into the application development lifecycle that cap and iteratively reduce that risk over time.

Testing for security

When managers hear that security is a software issue, their immediate reaction is to consult the "testing team." Unfortunately, when it comes to addressing software security, traditional testing approaches, and the traditional testing mindset, fall far short of what is required. Typical functional testing approaches are, after all, designed to verify expected behavior in software, not to find or experiment with unexpected reactions to malicious activities.

To combat this lack of knowledge within the test organization, tools such as penetration test and fuzz test have become popular weapons in the ongoing fight against software security exploitation. By submitting an application to coherent and well-known attack patterns, these tools attempt to force invalid behavior that can be exploited to achieve predictable chaos in the application's runtime context.

While incredibly valuable, the concern about such tools is that they are susceptible to the problem of test coverage. That is, the validation they perform is only as good as the number of attack vectors known to the publishers of the tool. As the set of well-known attack patterns increases almost daily, this is a battle that has "game over" written all over it.

Since the goal of most hackers is to find vulnerabilities by forcing a system to operate outside of its intended or designed behavior, an approach that uses penetration or fuzz test tools in addition to broader defect detection technology is required. Remembering that there is no silver bullet here, the practice of risk management as it pertains to validating an application's security profile must by definition be inclusive of the broadest possible set of techniques.

Software development organizations need to rigorously analyze and validate all feasible code paths in an application, not just those that happen to get executed as part of a functional or penetration testing process. While black-box testing will continue to play a vital role in identifying and mitigating exploitable attack vectors, it cannot be the only technique that is in use.

Code review

An important method that complements the kinds of black-box testing described above -- one that is particularly effective at uncovering security risk in application software -- is manual code review. Whether handled internally by developers or architects within the organization or outsourced to third-party security experts, rigorous code review enables detection of both implementation level and architectural vulnerabilities. The downside, of course, is that reviews of this type require both a disciplined method and a highly skilled review team in order to obtain consistent results over time.

In summary, while manual code review is a key component of any software quality program, and is certainly a highly plausible method for exposing security risk within application software, the review process can be very costly and is often prone to human error.

Developer training

Critical to any secure programming effort is ongoing developer education. There are whole categories of security vulnerabilities that many developers wouldn't recognize as a serious software flaw, simply because their mindset is more related to thinking of bugs in terms of "will it cause a crash" rather than "how can it be exploited?" Therefore, developer training on secure coding practices is a mandatory first step in ensuring that the team has a basic level of understanding of how to write secure code and how to understand whether a reported potential vulnerability is something that they need to address.

Continuing education in the face of ever-increasing attack capabilities is just as important, covering everything from the basics of coding practice to the more complex topics of threat modeling, threat-driven architecture, and behavior-derived attack vectors.

Source code analysis tools

Given that the key components of a mature security approach covered thus far all have shortcomings or natural limitations -- training, code review, and black-box testing -- the obvious question is, what else can be done to ensure accurate risk profiling and mitigation?

As I noted in the code review section, one of the best means of understanding an application and its potential for exploitation is inspection of its source code. Unfortunately, doing this manually is almost prohibitively expensive and time-consuming. But luckily, new technologies exist to help significantly with this investment profile.

Source code analysis, also known as static analysis, uses a compilation-based approach to representing and understanding the source of a given build unit (e.g., a library, a component, an entire application, etc.), and then applies sophisticated analysis techniques to that compiled representation in order to find situations that are exploitable, or which can cause operational failure.

Klocwork, as a leading vendor in the source code analysis space, provides several tools as part of a suite that allows developers, and development organizations as a whole, to apply static analysis to the source of their applications, to reverse-engineer the application's real architecture (as opposed to what you might believe the architecture to be), and to trend metrics over time in order to provide insight into the risk profile of the application under analysis.

With an automated code analysis solution, applications can be validated as to whether or not they contain many common security issues, such as:

- SQL injection, or the ability for an attacker to modify the statements that are used to execute queries against the underlying data store within the application
- Process injection, or the ability for an attacker to create arbitrary processes, or to hijack legitimately created processes within the application
- Code injection, whereby an attacker can provide invalidly formed input, thus causing corruption in the runtime state of the application that can be exploited in such a way as to execute code provided by the attacker
- Cross-site scripting, or the ability for an attacker to cause bad things to happen to the user's browser session, including the ability to reroute the user's session to phishing sites
- Denial of service, in which either by design or malicious intent the features of the application can be subverted to choke or otherwise significantly impact the ongoing viability of the application itself, or perhaps other network infrastructure

In order to realize the greatest benefit from source code analysis, the solution must deliver a cost-effective way of identifying and acting on security issues by being both automated as well as seamlessly integrated into existing business (for metrics) and development (for analysis, exploit identification, and enforcement of proper coding practices) environments.

Conclusion

No single solution can eliminate all possible risks associated with externally deployed software

applications. A solid developer education strategy that helps instil and reinforce secure coding practices, coupled with a combination of tools to assist the development team in understanding and then improving the risk profile of a software application over time, will significantly reduce a company's exposure. Source code analysis used by developers, and black-box testing techniques used by testers, represent an excellent combination of technologies that can be applied in earnest to begin the process of making your software secure.

About Klocwork

Klocwork is an enterprise software company providing automated source code analysis products that automate security vulnerability and quality risk assessment, remediation and measurement for C, C++, and Java software. Over 200 organizations have integrated Klocwork's automated source code analysis tools into their development process, thereby:

- Reducing risk by assuring their code is free of mission-critical flaws
- Reducing cost by catching issues early in the development cycle
- Freeing developers to focus on what they do best -- innovate

Klocwork provides the most comprehensive software security and software quality source code analysis solution available today. Contact Klocwork for more information at www.klocwork.com or info@klocwork.com.

Resources

- [Participate in the discussion forum.](#)
- A [new forum](#) has been created specifically for *Rational Edge* articles, so now you can share your thoughts about this or other articles in the current issue or our archives. Read what your colleagues the world over have to say, generate your own discussion, or join discussions in progress. Begin by clicking [HERE](#).
- [Global Rational User Group Community](#)

About the author

With over 15 years of global technology experience, Gwyn Fisher guides Klocwork's technical direction and development. Prior to joining Klocwork, he was senior vice president of research and development for LumaPath, where he led the creation of LumaPath's Active Integration product suite. Prior to LumaPath, Gwyn was the senior vice president of research and development with Hummingbird, Ltd., where he had operational responsibility for five product lines within eleven different locations across the United States, Canada, and Europe.

[Trademarks](#) | [My developerWorks terms and conditions](#)