

# Seven Steps to Reducing Software Security Risks

## Introduction

If you are a software developer, software development manager, or software quality assurance staff member, you probably know that developing secure software is no longer simply desirable – it’s completely essential.

Some developers might assume that most security problems arise from the operating system or networking layers, well below the application code they are working on. However, recent figures for Web-based applications show that over three-quarters of security exploits arose from applications (see Table 1).<sup>1</sup>

So, you know you need secure code, but how to get there? What are your security risks? What security failures and bugs do you have? What do these security risks, failures, and bugs mean? How can you reduce security risk in a way that doesn’t create new problems? How do you monitor my progress over time? This article will outline seven steps that will allow you to answer these and other questions as you improve your software’s security.

<b>Exploited Vulnerability</b>	<b>Percent Occurrence</b>
Server Applications	41%
Non-Server Applications	36%
Operating System Issues	15%
Hardware Issues	4%
Communication Protocol Issues	2%
Others	2%
Network and Protocol Stack Issues	1%
Encryption Issues	0%

**Table 1: Occurrence of Security Exploits by Vulnerability**

## Assess the Risks

Applications tend to have characteristic security risks. These risks often arise from the implementation technology. For example, C and C++ are notorious for their lack of inherent array range checking, and consequent buffer-overflow bugs, which allow hackers to insert malicious code into very long input strings. People writing applications with databases have to worry about SQL injection, where hackers put queries into otherwise-benign fields and gain access to sensitive data.

---

<sup>1</sup> Figures from the Open Web Application Security Project Web site, [www.owasp.org](http://www.owasp.org).

Security risks can also arise from the business application domain. For example, since they deal in money, banking applications are attractive targets for criminals and a major source of worry for bank IT departments. Applications that store personal information, such as medical history, are subject to regulations like HIPAA that require strict privacy controls.

Risk awareness is the first step in risk reduction. Companies have been reluctant to let outsiders know about the security failures they've had, but some of their failures make the news, and users report others. For example, the Open Web Application Security Project, [www.owasp.org](http://www.owasp.org), provides good information for those developing Web applications, as does the World Wide Web Consortium's security page, [www.w3.org/Security](http://www.w3.org/Security). Carnegie-Mellon's Software Engineering Institute's CERT Coordination Center, [www.cert.org](http://www.cert.org), provides a broader look at computer security issues. Last but not least, check out the searchable Risk Digest archives, [catless.ncl.ac.uk/Risks](http://catless.ncl.ac.uk/Risks), for great anecdotes and commentary on software risks, including security-related risks.

In addition to being aware of the failures, you need to be aware of the underlying bugs themselves. Depending on the kind of applications you're writing, you'll want to read appropriate books and Web sites for hints on common insecure coding constructs and how to avoid them. For example, entering "secure programming" in the Amazon.com search engine yields dozens of books, some general, some quite specific.

Once you are aware of the kinds of security risks that could affect your software, do a security risk analysis. Identify the specific risk items that you should be aware of. Meet with stakeholders to determine the level of risk in terms of likelihood and impact. Likelihood relates to the chances of any given risk becoming an actual security bug in your software. Impact relates to the effect on customers, users, and your software should the bug be exploited. Your analysis of the risks and their associated levels of risk will allow you to create a prioritized list of potential security failures.<sup>2</sup>

## Test to Know Where You Stand

If you're like most software development organizations, you don't have the luxury of starting over with new code on every project. How secure is that collection of existing code? If you're like many organizations, you haven't really had a chance to check. So, check the security of your existing software through a security test.

This type of test is often called a penetration test. Its purpose, as the name suggests, is to discover ways in which hackers and other unauthorized users can

---

<sup>2</sup> I describe the process of risk analysis in my book, *Managing the Testing Process*, 2e.

penetrate your system. Such a test is useful to check for security failures that your application already presents to the real world.

Remember that the best lock in the world does no good if it's installed in a door made of rotten wood. Similarly, applications with great security features that are installed in insecurely-configured environments can be hacked.

Do your installation procedures, user documentation, provisioning processes, and notification mechanisms support or impede security? I recently signed up for an account on an e-commerce site that seemed to have good security at first. I was asked to create a user name and password. The application enabled SSL encryption during this process. The input field masked the password when I entered it. I was then told that the application would e-mail me an activation notice after it verified my information. When I received the activation notice, the user name and password were in the e-mail, unencrypted and available to anyone who saw or intercepted that e-mail! Private and identifying information should not be stored or transmitted in an insecure fashion.

Consider identifying risk cases for each security requirement. Risk cases are like use cases – though perhaps more properly termed “misuse cases” – that lay out various scenarios of security failure. If you think about end-to-end processes that users go through, along with the environments in which your software will be deployed, you may think of some possible failures or issues you otherwise would have missed. You can confirm the presence or absence of these failures through specific tests.

You can learn how to run penetration tests yourself. Alternatively, you can hire a security-testing firm to handle it for you. On the one hand, you might have to make a significant investment in training and books to learn how to perform penetration tests properly and therefore decide a professional external resource can do a better job. On the other hand, you might feel more comfortable having security expertise in your team and therefore decide to invest in growing it.

Thoroughly testing applications that will run in various installed environments can be a real challenge. Such tests are a combination of end-to-end process testing, compatibility testing, and penetration testing. Depending on the multiplicity of environments, users, and procedures that your application can support, such tests cost a lot of money in terms of systems and effort. To save money on setting up a large variety of test configurations in-house, consider using an outside testing service.

Your prioritized list of risks should guide the penetration test, but you should also test for other failures that you might not have thought of. Based on the failures you find, revise your list of risks. Add new risks where you find unexpected failures. Increase the likelihood and impact based on the failures you find. You might also decrease the likelihood and impact for risks that don't relate to observed failures, or relate to failures that were less important than you

expected. However, be careful about assuming that a risk that isn't exploitable today won't be exploitable in future releases of the software.

Keep a list of the security problems you find and where you found them. You'll need this list to fix the problems, of course. However, I also recommend that you classify the problems in a few ways. One classification is based on the type of security flaw.<sup>3</sup> Another is the date on which the code was written or the version of the software in which it was introduced. Yet another is the major subsystem or component the code is part of. In addition, classify the severity (impact on the system) and priority (impact on the user) of each failure. Finally, classify each problem based on the security risks you identified earlier.

### Analyze to Know Where You Stand

The security test mentioned above will find security-related failures. However, not every security bug in the code will always exhibit a security failure. In other words, it is possible to have underlying bugs that did not exhibit any symptoms during the penetration test. Therefore, to find additional problems, do a static analysis of the code.

Static analysis means going through your code to look for bugs that could cause failures. You might have input fields which are not appropriately checked for size or syntax before being handed off for processing. You might have weak error handling. You might have situations where unauthorized users can pass snippets of languages like SQL or Korn shell into the system where they would be executed. Just because these bugs didn't result in failures doesn't mean they aren't bugs, and you should look for them.

You can automate your static analysis using tools.<sup>4</sup> For a large, existing code base, these tools will identify a large number of problems. Not all of these problems are of the same severity and importance. Somehow, you'll need to focus your attention on the most important of them. Fortunately, good tools will allow you to turn on and off particular rules and tune your static analysis at a level of granularity as fine as individual lines of code. Again, your list of risks can help guide you as you determine where to focus.

Based on your static analysis, add to your list of security problems, where you found each problem, and its classification.

---

<sup>3</sup> For example, you can use the OWASP's Top Ten Web application security flaws if you are creating Web applications ([www.owasp.org/index.php/Category:OWASP\\_Top\\_Ten\\_Project](http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project)).

<sup>4</sup> For example, one of our business partners, Parasoft, has tools like C++Test, Jtest, and .TEST that provide this capability. You can find an extensive list of testing tools, including static analysis tools, at [www.tejasconsulting.com](http://www.tejasconsulting.com).

## Evaluate to Understand Where You Stand

You've gathered a lot of data in the first few steps. Time to evaluate that data. What does the data mean; i.e., what information and patterns are hiding in the data? What is a smart plan of action for improving software security?

First of all, sort the problem list by priority and severity. You will likely want to immediately fix the problems with the highest levels of priority and severity. Microsoft famously reached a point where the number of critical security bugs became so high that they embarked on a crash problem to resolve these bugs. For months, Microsoft programmers did nothing but address security bugs. You might not be in as deep a hole – or be able to spare that much effort – but you'll want to address the urgent items right away.

However, you should also do some further evaluation before wading into battle with the security bugs. Bugs do not tend to be evenly distributed across the code base, but rather tend to exist in clusters. Decades ago, IBM studied their MVS software and found that 38% of the bugs that caused problems in production lived in 4% of the modules. On an Internet appliance project, I found that 69% of the bugs we discovered during testing lived in 25% of the modules. By looking for modules with particularly high numbers of security bugs, you might find that completely refactoring one or two modules is the smartest way to improve your software's security.

As you start to think about the long-term, evaluate how many bugs arise from each kind of security flaw. This will tell you which are the most typical problems you and your team face. Can you reduce the incidence of such problems through training for your programmers? Better code reviews? Better design reviews? All three? After all, you don't want to be fighting a constant battle against security problems with every release, so you and your team need to learn how to create better software.

You should also evaluate the incidence of security bugs based on the age of the code in which they were found. Software tends to "wear out" over the years, not as physical devices do, but rather through on-going maintenance which reduces the quality of the code. In addition, older code that was written when a programming language was new – or when the team was new to the language or technology – might contain more bugs. Plan for long-term refactoring of decrepit modules that are disproportionate contributors to software insecurity.

## Repair the Problems—Carefully

Any time you repair a bug in software, you take a risk that you might introduce a new bug. Many people call these *regression bugs*, because they represent some reduction in the level of software quality that was present before.

The risk of regression bugs applies to security bugs as much as any other bug. In addition, you can't assume that repairing a security bug would necessarily

introduce either no bug at all or another security bug. Fixing a security bug might introduce a functionality bug. So, as you repair the security bugs, make sure you have a plan to deal with regression risk. How can you do so?

A professional, independent test team typically deals with part of the regression problem. They might have created an automated suite of regression tests for functionality, performance, reliability, or other important quality characteristics. However, waiting for the end-stage testing is not ideal, as the cost and schedule implications of dealing with a bug increase the longer that bug is in the system.<sup>5</sup>

So, before delivering code to the test team, use code reviews, static analysis, and automated unit tests to help manage regression risk for each change you make to the system. Code reviews, ideally performed by at least two experts in addition to the author, should help catch many problems. Using the static analysis tool you've already invested in to check your new code is a best practice, and good static analysis tools can find many types of problems, not just security problems. Finally, the use of an automated unit testing harness – e.g., J-unit if you prefer open source or J-Test if you prefer a commercial tool – will provide a framework for an automated set of tests that will allow you to modify and refactor your code with confidence.<sup>6</sup>

## Examine Results in the Real World

Any time you make a process change, you should monitor how those process changes affect the real world. For example, I'm currently training for a marathon, but I hurt my ankle by overtraining in hills. So, I switched to a training schedule that focuses on low-impact aerobic exercises like bicycling and elliptical machines while my ankle heals. Will this process change help me achieve success? Two real world measures apply:

1. Based on the symptoms in my ankle, is it healing while continuing this training regimen, and can I gradually reintroduce running to the training?
2. Will I actually be able to run the marathon without pain and without re-injuring myself?

---

<sup>5</sup> For more on the economics of defects, see my presentation, "Investing in Testing," at the Library page of our Web site, [www.rbc-us.com](http://www.rbc-us.com). You can also listen to an audio recording of the presentation, if you like.

<sup>6</sup> For a detailed case study of how we helped one client implement a process of code reviews, static analysis, and automated unit testing, including creation of an automated test tool framework, see my article, "Mission Made Possible," written with Greg Kubackowski, at the Library page of our Web site, [www.rbc-us.com](http://www.rbc-us.com).

Similarly, you want to make sure that your new development process reduces the number of known security bugs in your code over time, and that the number of security-related incidents that occur in the field gradually goes down.

You should not expect that these two numbers would go down monotonically. Some natural variation in the testing and development processes will mean the number of known bugs might go both up and down. However, the trend over the long-term (say, one year or more) should be that the average number of known security bugs in any given month has gone down.

Similarly, you might have good months and bad months – months where no field security incidents are reported and months where a rash of them are – but this might simply be natural variation in usage patterns or seasonal usage. For example, you would expect that financial application security bugs related to fiscal-year closing operations would increase at the end of the year. However, again, the trend over the long-term should be that the average number of security incidents in any given month has gone down.

In addition to monitoring your own security bugs and failures, follow the news. The Internet and trade magazines can help you check for problems in applications similar to yours in business domain, implementation technology, or both. If you hear stories about problems that you think might constitute a risk for your application, update your risk analysis and re-evaluate accordingly.

## **Institutionalize Success**

The last step of this process is to do everything all over again, on every single project. That's something of an overstatement, since you don't need to start from a clean slate. You will need to repeat the first six steps, though, using your existing work as a baseline:

1. Re-assess security risks.
2. Re-test the application for security failures.
3. Re-analyze the software for security bugs.
4. Re-evaluate patterns in security risks, failures, and bugs.
5. Repair with care.
6. Re-examine the real-world results.

In each of these steps, make sure you look both at new concerns related to changes to your applications and concerns you might have previously overlooked.

Institutionalizing success, the final step of process improvement, is very easy to overlook. After a big push to improve software security, you might be tempted to celebrate success, relax your guard, and gradually slip back into old practices of coding.

I recently had a client that asked us to run a penetration test of their systems. We found a number of security failures during this test, and reported our findings to the engineering team. Later in the project, shortly before release, we re-ran the penetration test. The engineers had resolved all of the failures we had found previously. However, they had also built a bunch of new stuff, which had the exact same kinds of underlying security bugs exhibiting similar security failures. My client had dealt with the manifestations of bad security practices by repairing the security bugs we had found the first time, but had not changed the bad security practices themselves.

## Conclusions

Software security is an important concern, and it's not just for operating system and network vendors. If you're working at the application layer, your code is a target. In fact, the trend in software security exploits is away from massive, blunt-force attacks on the Internet or IT infrastructure and towards carefully crafted, criminal attacks on specific applications to achieve specific damage, often economic.

In this article, I have laid out a seven-step process to reduce your software's exposure to these attacks.

1. Assess security risks to focus your improvements.
2. Test the software for security failures.
3. Analyze the software for security bugs.
4. Evaluate patterns in security risks, failures, and bugs.
5. Repair the bugs with due care for regression.
6. Examine the real-world results by monitoring important security metrics.
7. Institutionalize the successful process improvements.

Carefully following this process will allow your organization to improve your software security in a way which is risk-based, thoroughly tested, data-driven, prudent, and continually re-aligned with real-world results.

## Author Biography

Rex Black is President of RBCS ([www.rbc-us.com](http://www.rbc-us.com)), an international consulting company that leads the field in many areas of quality and testing, including functional, security, and performance. Rex Black is also Chief Technical Officer of Pure Testing ([www.puretesting.com](http://www.puretesting.com)). RBCS and Pure Testing serve clients through assessment, consulting, training, staff augmentation, and offsite and offshore outsourcing, with over one hundred clients in countries around the world. Rex's bestseller, *Managing the Testing Process*, has reached over 30,000

readers on six continents. Rex thanks his colleagues at Pure Testing, including Harinath Pudipeddi, for their contributions to this article.