OWASP
The Open Web Application Security Project

The OWASP Testing Project

Version 1.0
December 2004

Endorsements

Trademarks

Author Credits

Initial Draft Author: Mark Curphey (mark@curphey.com)

Contributing Author and Technical Editor: Daniel Cuthbert (Daniel@deeper.co.za)

Contributing Author and Technical Editor: Andrew van der Stock (vanderaj@greebo.net)

Contributing Author and Technical Editor: Larry Shields (larry.shields@fmr.com)

Contributing Author: Javier Fernández-Sanguino (jfernandez@germinus.com)

Contributing Author: Stan Guzik (stan.guzik@owasp.org)

Contributing Author: Harinath Pudipeddi (harinath.puddipeddi@softrel.org)

Contributing Author: Glyn Geoghegan (glyng@corsaire.com)

Contributing Author: Mauro Bregolin (mauro_bregolin@yahoo.com)

Document History

- December 2004 Release of 1.0

Contents

Figures

Chapter 1: Introduction

About OWASP

The Open Web Application Security Project (OWASP) is a volunteer project dedicated to sharing knowledge and developing open source software that promotes a better understanding of web application security.

The project was founded in September 2000, and it has grown today to have participation from over 40 highly talented and enthusiastic participants from all around the world. The OWASP Foundation, a not-for-profit company, manages the OWASP project.

For more information about OWASP, see http://www.owasp.org/

About the OWASP Testing Project

The OWASP Testing Project has been in development for over two years. We wanted to help people understand the what, why, when, where, and how of testing their web applications, and not just provide a simple checklist or prescription of issues that should be addressed. We wanted to build a testing framework from which others can build their own testing programs or qualify other people's processes.

Writing the Testing Project has proven to be a difficult task. It has been a challenge to obtain consensus and develop the appropriate content, which would allow people to apply the overall content and framework described here, while enabling them to work in their own environment and culture. It has been also a challenge to change the focus of web application testing from penetration testing to testing integrated in the software development life cycle. Many industry experts and those responsible for software security at some of the largest companies in the world are validating the Testing Framework, presented as OWASP Testing Parts 1 and 2.

This framework aims at helping organizations test their web applications in order to build reliable and secure software rather than simply highlighting areas of weakness, although the latter is certainly a byproduct of many of OWASP's guides and checklists. As such, we have made some hard decisions about the appropriateness of certain testing techniques and technologies, which we fully understand will not be agreed upon by everyone.

However, OWASP is able to take the high ground and change culture over time through awareness and education based on consensus and experience, rather than take the path of the "least common denominator."

The Economics of Insecure Software

The cost of insecure software to the world economy is seemingly immeasurable. In June 2002, the US National Institute of Standards (NIST) published a survey on the cost of insecure software to the US economy due to inadequate software testing[1].

Most people understand at least the basic issues, or have a deeper technical understanding of the vulnerabilities. Sadly, few are able to translate that knowledge into monetary value and thereby quantify the costs to their business. We believe that until this happens, CIO's will not be able to develop an accurate return on a security investment and subsequently assign appropriate budgets for software security. See Ross Anderson's page at http://www.cl.cam.ac.uk/users/rja14/econsec.html for more information about the economics of security.

The framework described in this document encourages people to measure security throughout their entire development process. They can then relate the cost of insecure software to the impact it has on their business, and consequently develop appropriate business decisions (resources) to manage the risk.

Insecure software has its consequences, but insecure web applications, exposed to millions of users through the Internet are a growing concern. Even now, the confidence of customers using the World Wide Web to purchase or cover their needs is decreasing as more and more web applications are exposed to attacks.

OWASP Testing Project Parts 1 and 2

The Testing Project comprises two parts.

Part 1 (this document) covers the processes involved in testing web applications:

- The scope of what to test
- Principles of testing
- Testing techniques explained
- The OWASP testing framework explained

Part 2 (due for release Q2 of 2005 covers how to test each software development life cycle phase using techniques described in this document. For example, Part 2 covers how to test for specific vulnerabilities such as SQL Injection by code inspection and penetration testing.

Scope of this Document

This document is designed to help organizations understand what comprises a testing program, and to help them identify the steps that they need to undertake to build and operate that testing program on their web applications. It is intended to give a broad view of the elements required to make a comprehensive web application security program.

This guide can be used as a reference and as a methodology to help determine the gap between your existing practices and industry best practices. This guide allows organizations to compare themselves against industry peers, understand the magnitude of resources required to test and remediate their software, or prepare for an audit.

This document does not go into the technical details of how to test an application, as the intent is to provide a typical security organizational framework. The technical details about how to test

[1] *The economic impacts of inadequate infrastructure for software testing*. (2002, June 28). Retrieved May 4, 2004, from http://www.nist.gov/public_affairs/releases/n02-10.htm

an application, as part of a penetration test or code review will be covered in the Part 2 document mentioned above.

What Do We Mean By Testing?

During the development lifecycle of a web application, many things need to be tested.

The Merriam-Webster Dictionary describes testing as:

- To put to test or proof
- To undergo a test
- To be assigned a standing or evaluation based on tests.

For the purposes of this document, testing is a process of comparing the state of something against a set of criteria.

In the security industry, people frequently test against a set of mental criteria that are neither well defined nor complete. For this reason and others, many outsiders regard security testing as a black art. This document's aim is to change that perception and to make it easier for people without in-depth security knowledge to make a difference.

The Software Development Life Cycle Process

One of the best methods to prevent security bugs from appearing in production applications is to improve the Software Development Life Cycle (SDLC) by including security. If a SDLC is not currently being used in your environment, it is time to pick one! The following figure shows a generic SDLC model as well as the (estimated) increasing cost of fixing security bugs in such a model.

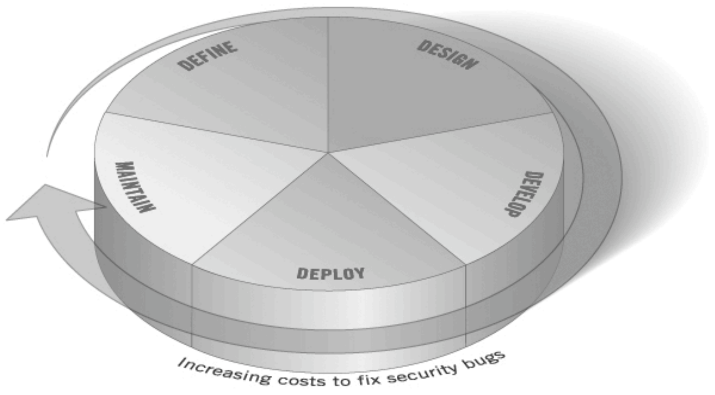

Figure 1: Generic SDLC Model.

Companies should inspect their overall SDLC to ensure that security is an integral part of the development process. SDLC's should include security tests to ensure security is adequately covered and controls are effective throughout the development process.

The Scope of What To Test

It can be helpful to think of software development as a combination of ***people***, ***process***, and ***technology***. If these are the factors that "create" software then it is logical that these are the factors that must be tested. Today most people generally test the technology or the software itself. In fact most people today don't test the software until it has already been created and is in the deployment phase of its lifecycle (i.e. code has been created and instantiated into a working web application). This is generally a very ineffective and cost prohibitive practice.

An effective testing program should have components that test;

People – to ensure that there is adequate education and awareness

Process – to ensure that there are adequate policies and standards and that people know how to follow these policies

Technology – to ensure that the process has been effective in its implementation

Unless a holistic approach is adopted, testing just the technical implementation of an application will not uncover management or operational vulnerabilities that could be present. By testing the people, policy and process you can catch issues that would later manifest them into defects in the technology, thus eradicating bugs early and identify the root causes of defects. Likewise only testing some of the technical issues that can be present in a system will result in an incomplete and inaccurate security posture assessment. Denis Verdon, Head of Information Security at Fidelity National Financial (http://www.fnf.com) presented an excellent analogy for this misconception at the OWASP AppSec 2004 Conference in New York.

"If cars were built like applications…safety tests would assume frontal impact only. Cars would not be roll tested, or tested for stability in emergency maneuvers, brake effectiveness, side impact and resistance to theft."

Feedback and Comments

As with all OWASP projects, we welcome comments and feedback. We especially like to know that our work is being used and that it is effective and accurate. We plan to update and revise this document as needed.

Contact the project team via the following methods:

- Project Leaders – Daniel Cuthbert (daniel@deeper.co.za), Mark Curphey (mark@curphey.com)
- Project Team – testing@owasp.org

Please mark any email with the Subject [OWASP Testing]

Chapter 2: Principles of Testing

There are some common misconceptions when developing a testing methodology to weed out security bugs in software. This chapter covers some of the basic principles that should be taken into account by professionals when testing for security bugs in software.

There is No Silver Bullet

While it is tempting to think that a security scanner or application firewall will either provide a multitude of defenses or identify a multitude of problems, in reality there are no silver bullets to the problem of insecure software. Application security assessment software, while useful as a first pass to find low-hanging fruit, is generally immature and ineffective at in-depth assessments and at providing adequate test coverage. Remember that security is a process, not a product.

Think Strategically, Not Tactically

Over the last few years, security professionals have come to realize the fallacy of the *patch and penetrate* model that was pervasive in information security during the 1990's. The patch and penetrate model involves fixing a reported bug, but without proper investigation of the root cause. This patch and penetrate model is usually associated with the window of vulnerability[2] show in the figure below. The evolution of vulnerabilities in common software used worldwide has shown the ineffectiveness of this model. Vulnerability studies[3] have shown that the with the reaction time of attackers worldwide, the typical window of vulnerability does not provide enough time for patch installation, since the time between a vulnerability is uncovered and an automated attack against is developed and released is decreasing every year.

There are also several wrong assumptions in this patch and penetrate model: patches interfere with the normal operations and might break existing applications, and not all the users might (in the end) be aware of a patch's availability. Consequently not all the product's users will apply patches, either because of this issue or because they lack knowledge about the patch's existence.

[2] Fore more information about the *window of vulnerability* please refer to Bruce Shneier's Cryptogram Issue #9, available at http://www.schneier.com/crypto-gram-0009.html

[3] Such as those included Symantec's Threat Reports

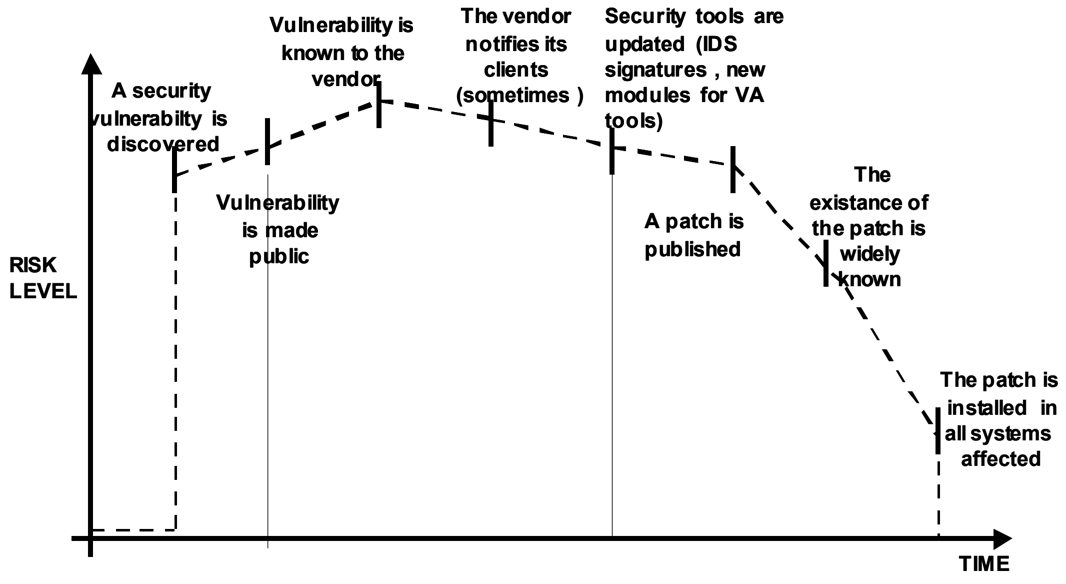

Figure 2: Window of exposure

To prevent reoccurring security problems within an application, it is essential to build security into the Software Development Life Cycle (SDLC) by developing standards, policies, and guidelines that fit and work within the development methodology. Threat modeling and other techniques should be used to help assign appropriate resources to those parts of a system that are most at risk.

The SDLC is King

The SDLC is a process that is well known to developers. By integrating security into each phase of the SDLC, it allows for a holistic approach to application security that leverages the procedures already in place within the organization. Be aware that while the names of the various phases may change depending on the SDLC model used by an organization, each conceptual phase of the archetype SLDC will be used to develop the application (i.e. define, design, develop, deploy, maintain). Each phase has security considerations that should become part of the existing process, to ensure a cost-effective and comprehensive security program.

Test Early and Test Often

By detecting a bug early within the SDLC, it allows it to be addressed more quickly and at a lower cost. A security bug is no different from a functional or performance based bug in this regard. A key step in making this possible is to educate the development and QA organizations about common security issues and the ways to detect & prevent them. Although new libraries, tools or languages might help design better programs (with fewer security bugs) new threats arise constantly and developers must be aware of those that affect the software they are developing. Education in security testing also helps developers acquire the appropriate mindset to test and application from an attacker's perspective. This allows each organization to consider security issues as part of their existing responsibilities.

Understand the Scope of Security

It is important to know how much security a given project will require. The information and assets that are to be protected should be given a classification that states how they are to be handled (e.g. confidential, secret, top secret). Discussions should occur with legal council to ensure that any specific security needs will be met. In the USA they might come from federal regulations such as the Gramm-Leach-Bliley act (http://www.ftc.gov/privacy/glbact/), or from state laws such as California SB-1386 (http://www.leginfo.ca.gov/pub/01-02/bill/sen/sb_1351-1400/sb_1386_bill_20020926_chaptered.html). For organizations based in EU countries, both country-specific regulation and EU Directives might apply, for example, Directive 96/46/EC[4] makes it mandatory to treat personal data in applications with due care, whatever the application.

Mindset

Successfully testing an application for security vulnerabilities requires thinking "outside of the box". Normal use cases will test the normal behavior of the application when a user is using it in the manner that you expect. Good security testing requires going beyond what is expected and thinking like an attacker who is trying to break the application. Creative thinking can help to determine what unexpected data may cause an application to fail in an insecure manner. It can also help find what assumptions made by web developers are not always true and how can they be subverted. This is one of the reasons why automated tools are actually bad at automatically testing for vulnerabilities, this creative thinking must be done in a case by case basis and most of the web applications are being developed in a unique way (even if using common frameworks)

Understanding the Subject

One of the first major initiatives in any good security program should be to require accurate documentation of the application. The architecture, data flow diagrams, use cases, and more should be written in formal documents and available for review. The technical specification and application documents should include information that lists not only the desired use cases, but also any specifically disallowed use cases. Finally, it is good to have at least a basic security infrastructure that allows monitoring and trending of any attacks against your applications & network (e.g. IDS systems).

Use the Right Tools

While we have already stated that there is no tool silver bullet, tools do play a critical role in the overall security program. There is a range of open source and commercial tools that can assist in automation of many routine security tasks. These tools can simplify and speed the security process by assisting security personnel in their tasks. It is important to understand exactly what these tools can and cannot do, however, so that they are not oversold or used incorrectly.

The Devil is in the Details

It is critical not to perform a superficial security review of an application and consider it complete. This will instill a false sense of confidence that can be as dangerous as not having done a security review in the first place. It is vital to carefully review the findings and weed out any false positives that may remain in the report. Reporting an incorrect security finding can

[4] http://europa.eu.int/comm/internal_market/privacy/law_en.htm

often undermine the valid message of the rest of a security report. Care should be taken to verify that every possible section of application logic has been tested, and that every use case scenario was explored for possible vulnerabilities.

Use Source Code When Available

While black box penetration test results can be impressive and useful to demonstrate how vulnerabilities are exposed in production, they are not the most effective way to secure an application. If the source code for the application is available, it should be given to the security staff to assist them while performing their review. It is possible to discover vulnerabilities within the application source that would be missed during a black box engagement.

Develop Metrics

An important part of a good security program is the ability to determine if things are getting better. It is important to track the results of testing engagements, and develop metrics that will reveal the application security trends within the organization. These metrics can show if more education and training is required, if there is a particular security mechanism that is not clearly understood by development, and if the total number of security related problems being found each month is going down.

Consistent metrics that can be generated in an automated way from available source code will also help the organization in assessing the effectiveness of mechanisms introduced to reduce security bugs in software development. Metrics are not easily developed so using standard metrics like those provided by the OWASP Metrics project and other organizations might be a good head start.

Chapter 3: Testing Techniques Explained

This section presents a high-level overview of various testing techniques that can be employed when building a testing program. It does not present specific methodologies for these techniques, although Part 2 of the OWASP Testing project will address this information. This section is included to provide context for the framework presented in Chapter 4 and to highlight the advantages and disadvantages of some of the techniques that can be considered.

- Manual Inspections & Reviews
- Threat Modeling
- Code Review
- Penetration Testing

Manual Inspections & Reviews

Manual inspections are human-driven reviews that typically test the security implications of the people, policies, and processes, but can include inspection of technology decisions such as architectural designs. They are usually conducted by analyzing documentation or using interviews with the designers or system owners. While the concept of manual inspections and human reviews is simple, they can be among the most powerful and effective techniques available. By asking someone how something works and why it was implemented in a specific way, it allows the tester to quickly determine if any security concerns are likely to be evident.

Manual inspections and reviews are one of the few ways to test the software development lifecycle process itself and to ensure that there is an adequate policy or skill set in place.

As with many things in life, when conducting manual inspections and reviews we suggest you adopt a *trust but verify* model. Not everything everyone tells you or shows you will be accurate. Manual reviews are particularly good for testing whether people understand the security process, have been made aware of policy, and have the appropriate skills to design and/or implement a secure application. Other activities, including manually reviewing the documentation, secure coding policies, security requirements, and architectural designs, should all be accomplished using manual inspections.

Advantages and Disadvantages

Advantages

- Requires no supporting technology
- Can be applied to a variety of situations
- Flexible
- Promotes team work
- Early in the SDLC

Disadvantages

- Can be time consuming
- Supporting material not always available
- Requires significant human thought and skill to be effective!

Threat Modeling

Overview

In the context of the technical scope, threat modeling has become a popular technique to help system designers think about the security threats that their systems will face. It enables them to develop mitigation strategies for potential vulnerabilities. Threat modeling helps people focus their inevitably limited resources and attention on the parts of the system that most require it.

Threat models should be created as early as possible in the software development life cycle, and should be revisited as the application evolves and development progresses. Threat modeling is essentially risk assessment for applications. It is recommended that all applications have a threat model developed and documented.

To develop a threat model, we recommend taking a simple approach that follows the NIST 800-30 [5] standard for risk assessment. This approach involves:

- Decomposing the application – through a process of manual inspection understanding how the application works, its assets, functionality and connectivity.
- Defining and classifying the assets – classify the assets into tangible and intangible assets and rank them according to business criticality.
- Exploring potential vulnerabilities (technical, operational, and management)
- Exploring potential threats – through a process of developing threat scenarios or attacks trees and develops a realistic view of potential attack vectors from an attacker's perspective.
- Creating mitigation strategies – develop mitigating controls for each of the threats deemed to be realistic.

The output from a threat model itself can vary but is typically a collection of lists and diagrams.

Part 2 of the OWASP Testing Guide (the detailed "How To" text) will outline a specific Threat Modeling methodology. There is no right or wrong way to develop threat models, and several techniques have evolved. The OCTAVE model from Carnegie Mellon (http://www.cert.org/octave/) is worth exploring.

Advantages and Disadvantages

Advantages

- Practical attackers view of the system
- Flexible
- Early in the SDLC

Disadvantage

- Relatively new technique
- Good threat models don't automatically mean good software

[5] Stoneburner, G., Goguen, A., & Feringa, A. (2001, October). *Risk management guide for information technology systems*. Retrieved May 7, 2004, from http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf

Source Code Review

Overview

Source code review is the process of manually checking a web applications source code for security issues. Many serious security vulnerabilities cannot be detected with any other form of analysis or testing. As the popular saying goes “if you want to know what’s really going on, go straight to the source”. Almost all security experts agree that there is no substitute for actually looking at the code. All the information for identifying security problems is there in the code somewhere. Unlike testing third party closed software such as operating systems, when testing web applications (especially if they have been developed in-house) the source code is and should be almost always available.

Many unintentional but significant security problems are also extremely difficult to discover with other forms of analysis or testing such as penetration testing making source code analysis the technique of choice for technical testing. With the source code a tester can accurately determine what is happening (or is supposed to be happening) and remove the guess work of black box testing (such as penetration testing). Examples of issues that are particularly conducive to being found through source code reviews include concurrency problems, flawed business logic, access control problems and cryptographic weaknesses as well as backdoors, Trojans, Easter eggs, time bombs, logic bombs, and other forms of malicious code. These issues often manifest themselves as the most harmful vulnerabilities in web sites. Source code analysis can also be extremely efficient to find implementation issues such as places where input validation was not performed or when fail open control procedures maybe present. But keep in mind that operational procedures need to be reviewed also since the source code being deployed might not be the same as the one being analyzed.[6]

Advantages and Disadvantages

Advantages

- Completeness and effectiveness
- Accuracy
- Fast (for competent reviewers)

Disadvantages

- Requires highly skilled security developers
- Can miss calls to issues in compiled libraries
- Can not detect run-time errors easily
- The source code actually deployed might differ from the one being analyzed.

[6] See "Reflections on Trusting Trust" by Ken Thompson (http://cm.bell-labs.com/who/ken/trust.html)

Penetration Testing

Overview

Penetration testing has become a common technique used to test network security for many years. It is also commonly known as black box testing or ethical hacking. Penetration testing is essentially the "art" of testing a running application remotely, without knowing the inner workings of the application itself to find security vulnerabilities. Typically, the penetration test team would have access to an application as if they were users. The tester acts like a attacker and attempts to find and exploit vulnerabilities. In many cases the tester will be given a valid account on the system.

While penetration testing has proven to be effective in network security, the technique does not naturally translate to applications. When penetration testing is performed on networks and operating systems, the majority of the work is involved in finding and then exploiting known vulnerabilities in specific technologies. As web applications are almost exclusively bespoke, penetration testing in the web application arena is more akin to pure research. Penetration testing tools have been developed that automated the process but again with the nature of web applications their effectiveness is usually poor.

Many people today use web application penetration testing as their primary security testing technique. Whilst it certainly has its place in a testing program, we do not believe it should be considered as the primary or only testing technique. Gary McGraw summed up penetration testing well when he said, "If you fail a penetration test you know you have a very bad problem indeed. If you pass a penetration test you do not know that you don't have a very bad problem".

However, *focused* penetration testing (i.e. testing that attempts to exploit known vulnerabilities detected in previous reviews) can be useful in detecting if some specific vulnerabilities are actually fixed in the source code deployed at the web site.

Advantages and Disadvantages

Advantages

- Can be fast (and therefore cheap)
- Requires a relatively lower skill-set than source code review
- Lots of web application pen testers available
- Familiar
- Tests the code that is actually being exposed

Disadvantages

- Inefficient
- Too late in the SDLC
- Front impact testing only!

The Need for a Balanced Approach

With so many techniques and so many approaches to testing the security of your web applications, it can be difficult to understand which techniques to use and when to use them.

Experience shows that there is no right or wrong answer to exactly what techniques should be used to build a testing framework. The fact remains that all techniques should probably be used to ensure that all areas that need to be tested are tested. What is clear, however, is that there is no single technique that effectively covers all security testing that must be performed to ensure that all issues have been addressed. Many companies adopt one approach, which has historically been penetration testing. Penetration testing, while useful, cannot effectively address many of the issues that need to be tested, and is simply "too little too late" in the software development life cycle (SDLC).

The correct approach is a balanced one that includes several techniques, from manual interviews to technical testing. The balanced approach is sure to cover testing in all phases in the SDLC. This approach leverages the most appropriate techniques available depending on the current SDLC phase.

Of course there are times and circumstances where only one technique is possible; for example, a test on a web application that has already been created, and where the testing party does not have access to the source code. In this case, penetration testing is clearly better than no testing at all. However, we encourage the testing parties to challenge assumptions, such as no access to source code, and to explore the possibility of complete testing.

A balanced approach varies depending on many factors, such as the maturity of the testing process and corporate culture. However, it is recommended that a balanced testing framework look something like the representations shown in Figure 3 and Figure 4.

The following figure shows a typical proportional representation overlaid onto the software development life cycle. In keeping with research and experience, it is essential that companies place a higher emphasis on the early stages of development.

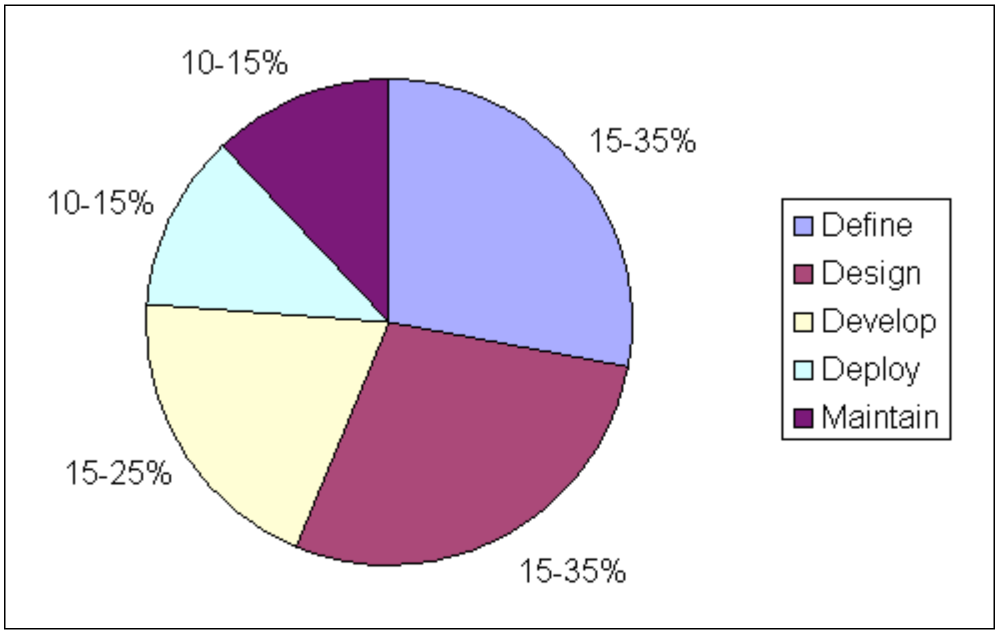

Figure 3: Proportion of Test Effort in SDLC.

The following figure shows a typical proportional representation overlaid onto testing techniques.

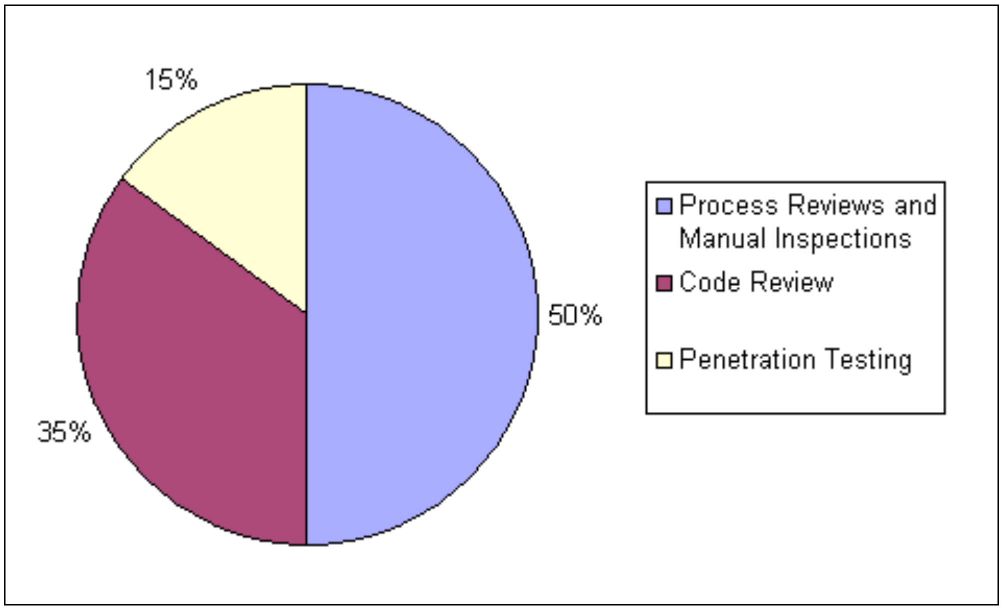

Figure 4: Proportion of Test Effort According to Test Technique.

A Note about Web Application Scanners

Many organizations have started to use web application scanners. While they undoubtedly have a place in a testing program, we want to highlight some fundamental issues about why we do not believe that automating black box testing is (or will ever be) effective. By highlighting these issues, we are not discouraging web application scanner use. Rather, we are saying that their limitations should be understood, and testing frameworks should be planned appropriately.

***NB:** OWASP is currently working to develop a web application scanner-benchmarking platform.*

The following examples indicate why automated black box testing is not effective.

Example 1: Magic Parameters

Imagine a simple web application that accepts a name-value pair of "magic" and then the value. For simplicity, the GET request may be:

http://www.host/application?magic=value

To further simplify the example, the values in this case can only be ASCII characters a – z (upper or lowercase) and integers 0 – 9. The designers of this application created an administrative backdoor during testing, but obfuscated it to prevent the casual observer from discovering it. By submitting the value sf8g7sfjdsurtsdieerwqredsgnfg8d (30 characters), the user will then be logged in and presented with an administrative screen with total control of the application.

The HTTP request is now:

http://www.host/application?magic= sf8g7sfjdsurtsdieerwqredsgnfg8d

Given that all of the other parameters were simple two- and three-characters fields, it is not possible to start guessing combinations at approximately 28 characters. A web application scanner will need to brute force (or guess) the entire key space of 30 characters. That is up to 30^{28} permutations, or trillions of HTTP requests! That is an electron in a digital haystack!

The code for this may look like the following:

```
public void doPost( HttpServletRequest request, HttpServletResponse response)
{
        String magic = "sf8g7sfjdsurtsdieerwqredsgnfg8d";
        boolean admin = magic.equals( request.getParameter("magic"));
        if (admin) doAdmin( request, response);
        else …. // normal processing
}
```

By looking in the code, the vulnerability practically leaps off the page as a potential problem.

Example 2: Bad Cryptography

Cryptography is widely used in web applications. Imagine that a developer decided to write a simple cryptography algorithm to sign a user in from site A to site B automatically. In his/her wisdom, the developer decides that if a user is logged into site A, then he/she will generate a key using an MD5 hash function that comprises:

Hash { username : date }

When a user is passed to site B, he/she will send the key on the query string to site B in an HTTP re-direct. Site B independently computes the hash, and compares it to the hash passed on the request. If they match, site B signs the user in as the user they claim to be.

Clearly, as we explain the scheme, the inadequacies can be worked out, and it can be seen how anyone that figures it out (or is told how it works, or downloads the information from Bugtraq) can login as any user. Manual inspection, such as an interview, would have uncovered this security issue quickly, as would inspection of the code.

A black-box web application scanner would have seen a 128-bit hash that changed with each user, and by the nature of hash functions, did not change in any predicable way.

A Note about Static Source Code Review Tools

Many organizations have started to use static source code scanners. While they undoubtedly have a place in a comprehensive testing program, we want to highlight some fundamental issues about why we do not believe this approach is effective when used alone. Static source code analysis alone cannot understand the context of semantic constructs in code, and therefore is prone to a significant number of false positives. This is particularly true with C and C++. The technology is useful in determining interesting places in the code, however significant manual effort is required to validate the findings. For example:

```
char szTarget[12];
char *s = "Hello, World";

size_t cSource = strlen_s(s,20);
strncpy_s(temp,sizeof(szTarget),s,cSource);
strncat_s(temp,sizeof(szTarget),s,cSource);
```

Chapter 4: The OWASP Testing Framework

Overview

This section describes a typical testing framework that can be developed within an organization. It can be seen as a reference framework that comprises techniques and tasks that are appropriate at various phases of the software development life cycle (SDLC). Companies and project teams can use this model to develop their own testing framework and to scope testing services from vendors. This framework should not be seen as prescriptive, but as a flexible approach that can be extended and molded to fit an organization's development process and culture.

This section aims to help organizations build a complete strategic testing process, and is not aimed at consultants or contractors who tend to be engaged in more tactical, specific areas of testing.

It is critical to understand why building an end-to-end testing framework is crucial to assessing and improving software security. Howard and LeBlanc note in *Writing Secure Code* that issuing a security bulletin costs Microsoft at least $100,000, and it costs their customers collectively far more than that to implement the security patches. They also note that the US government's CyberCrime web site (http://www.cybercrime.gov/cccases.html) details recent criminal cases and the loss to organizations. Typical losses far exceed USD $100,000.

With economics like this, it is little wonder why software vendors move from solely performing black box security testing, which can only be performed on applications that have already been developed, to concentrate on the early cycles of application development such as definition, design, and development.

Many security practitioners still see security testing in the realm of penetration testing. As shown in Chapter 3: Testing Techniques Explained, and by the framework, while penetration testing has a role to play, it is generally inefficient at finding bugs and relies excessively on the skill of the tester. It should only be considered as an implementation technique, or to raise awareness of production issues. To improve the security of applications, the security quality of the software must be improved. That means testing the security at the definition, design, develop, deploy, and maintenance stages, and not relying on the costly strategy of waiting until code is completely built.

As discussed in the introduction of this document, there are many development methodologies such as the Rational Unified Process, eXtreme and Agile development, and traditional waterfall methodologies. The intent of this guide is to suggest neither a particular development methodology nor provide specific guidance that adheres to any particular methodology. Instead, we are presenting a generic development model, and the reader should follow it according to their company process.

This testing framework consists of the following activities that should take place:

- Before Development Begins
- During Definition and Design
- During Development
- During Deployment
- Maintenance and Operations

Phase 1 — Before Development Begins

Before application development has started:

- Test to ensure that there is an adequate SDLC where security is inherent.
- Test to ensure that the appropriate policy and standards are in place for the development team.
- Develop the metrics and measurement criteria.

Phase 1A: Policies and Standards Review

Ensure that there are appropriate policies, standards, and documentation in place. Documentation is extremely important as it gives development teams guidelines and policies that they can follow.

People can only do the right thing, if they know what the right thing is.

If the application is to be developed in Java, it is essential that there is a Java secure coding standard. If the application is to use cryptography, it is essential that there is a cryptography standard. No policies or standards can cover every situation that the development team will face. By documenting the common and predictable issues, there will be fewer decisions that need to be made during the development process.

Phase 1B: Develop Measurement and Metrics Criteria (Ensure Traceability)

Before development begins, plan the measurement program. By defining criteria that needs to be measured, it provides visibility into defects in both the process and product. It is essential to define the metrics before development begins, as there may be a need to modify the process in order to capture the data.

Phase 2: During Definition and Design

Phase 2A: Security Requirements Review

Security requirements define how an application works from a security perspective. It is essential that the security requirements be tested. Testing in this case means testing the assumptions that are made in the requirements, and testing to see if there are gaps in the requirements definitions.

For example, if there is a security requirement that states that users must be registered before they can get access to the whitepapers section of a website, does this mean that the user must be registered with the system, or should the user be authenticated? Ensure that requirements are as unambiguous as possible.

When looking for requirements gaps, consider looking at security mechanisms such as:

- User Management (password reset etc.)
- Authentication
- Authorization
- Data Confidentiality
- Integrity
- Accountability
- Session Management
- Transport Security
- Privacy

Phase 2B: Design an Architecture Review

Applications should have a documented design and architecture. By documented we mean models, textual documents, and other similar artifacts. It is essential to test these artifacts to ensure that the design and architecture enforce the appropriate level of security as defined in the requirements.

Identifying security flaws in the design phase is not only one of the most cost efficient places to identify flaws, but can be one of the most effective places to make changes. For example, being able to identify that the design calls for authorization decisions to be made in multiple places; it may be appropriate to consider a central authorization component. If the application is performing data validation at multiple places, it may be appropriate to develop a central validation framework (fixing input validation in one place, rather than hundreds of places, is far cheaper).

If weaknesses are discovered, they should be given to the system architect for alternative approaches.

Phase 2C: Create and Review UML Models

Once the design and architecture is complete, build UML models that describe how the application works. In some cases, these may already be available. Use these models to confirm with the systems designers an exact understanding of how the application works. If weaknesses are discovered, they should be given to the system architect for alternative approaches.

Phase 2D: Create and Review Threat Models

Armed with design and architecture reviews, and the UML models explaining exactly how the system works, undertake a threat modeling exercise. Develop realistic threat scenarios (see Manual Inspections & Reviews on page 9). Analyze the design and architecture to ensure that these threats have been mitigated, accepted by the business, or assigned to a third party, such as an insurance firm. When identified threats have no mitigation strategies, revisit the design and architecture with the systems architect to modify the design.

Phase 3: During Development

Theoretically, development is the implementation of a design. However, in the real world, many design decisions are made during code development. These are often smaller decisions that were either too detailed to be described in the design, or in other cases, issues where no policy or standards guidance was offered. If the design and architecture was not adequate, the developer will be faced with many decisions. If there were insufficient policies and standards, the developer will be faced with even more decisions.

Phase 3A: Code Walkthroughs

The security team should perform a code walkthrough with the developers, and in some cases, the system architects. A code walkthrough is a high-level walkthrough of the code where the developers can explain the logic and flow. It allows the code review team to obtain a general understanding of the code, and allows the developers to explain why certain things were developed the way they were.

The purpose is not to perform a code review, but to understand the flow at a high-level, the layout, and the structure of the code that makes up the application.

Phase 3B: Code Reviews

Armed with a good understanding of how the code is structured and why certain things were coded the way they were, the tester can now examine the actual code for security defects.

Static code reviews validate the code against a set of checklists, including:

- Business requirements for availability, confidentiality, and integrity
- OWASP Guide or Top 10 Checklists (depending on the depth of the review) for technical exposures
- Specific issues relating to the language or framework in use, such as the Scarlet paper for PHP or Microsoft Secure Coding checklists for ASP.NET
- Any industry specific requirements, such as Sarbanes-Oxley 404, COPPA, ISO 17799, APRA, HIPAA, Visa Merchant guidelines or other regulatory regimes

In terms of return on resources invested (mostly time), static code reviews produce far higher quality returns than any other security review method, and rely least on the skill of the reviewer, within reason. However, they are not a silver bullet, and need to be considered carefully within a full-spectrum testing regime.

For more details on OWASP checklists, please refer to OWASP Guide for Secure Web Applications, or the latest edition of the OWASP Top 10.

Phase 4: During Deployment

Phase 4A: Application Penetration Testing

Having tested the requirements, analyzed the design, and performed code review, it might be assumed that all issues have been caught. Hopefully, this is the case, but penetration testing the application after it has been deployed provides a last check to ensure that nothing has been missed.

Phase 4B: Configuration Management Testing

The application penetration test should include the checking of how the infrastructure was deployed and secured. While the application may be secure, a small aspect of the configuration could still be at a default install stage and vulnerable to exploitation.

Phase 5: Maintenance and Operations

Phase 5A: Conduct Operational Management Reviews

There needs to be a process in place which details how the operational side, of the application and infrastructure, is managed.

Phase 5B: Conduct Periodic Health Checks

Monthly or quarterly health checks should be performed on both the application and infrastructure to ensure no new security risks have been introduced and that the level of security is still intact.

Phase 5C: Ensure Change Verification

After every change has been approved and tested in the QA environment and deployed into the production environment, it is vital that as part of the change management process, the change is checked to ensure that the level of security hasn't been affected by the change.

A Typical SDLC Testing Workflow

The following figure shows a typical SDLC Testing Workflow.

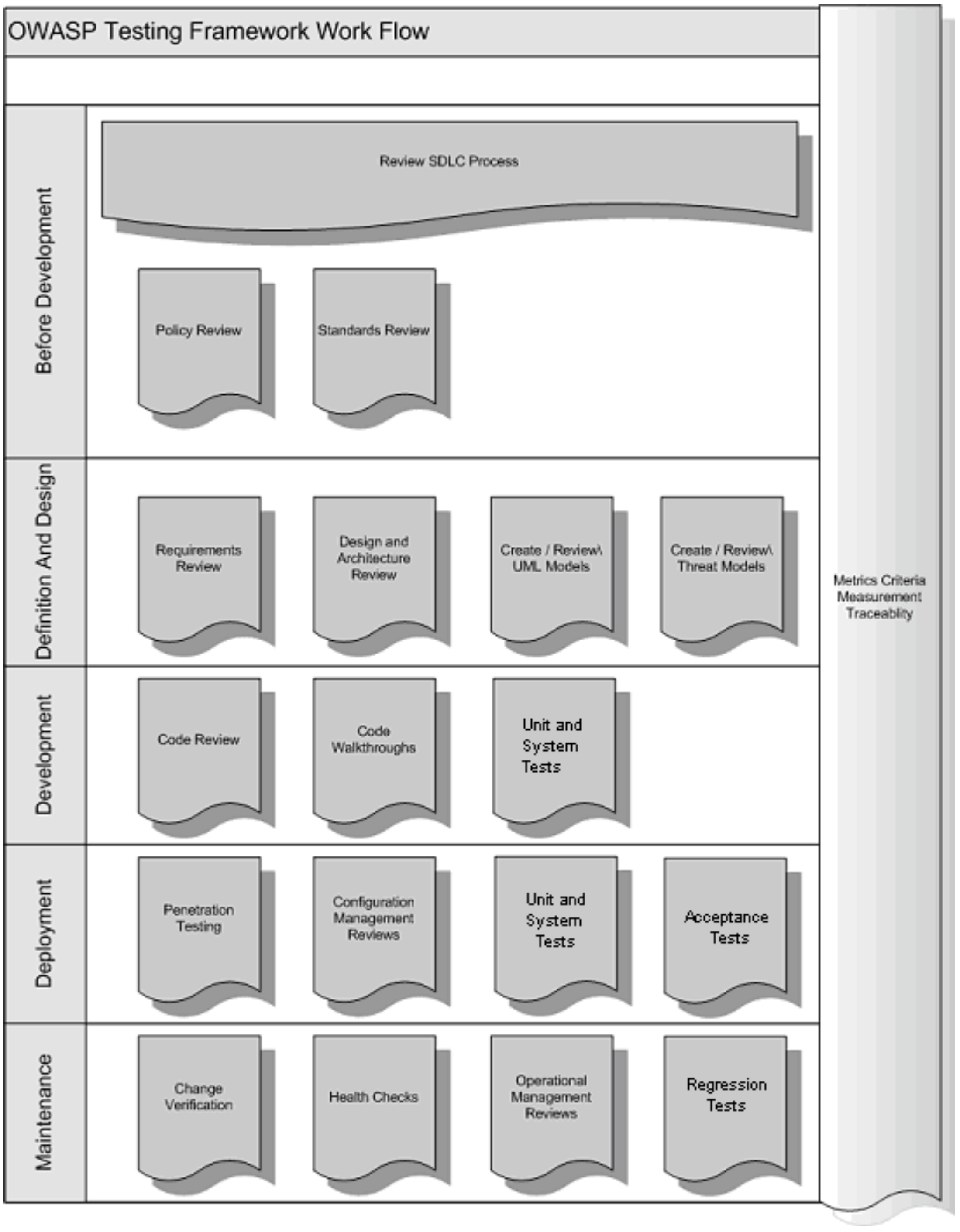

Figure 5: Typical SDLC Testing Workflow.

Appendix A: Testing Tools

Source Code Analyzers

Open Source / Freeware

Analyzer	URL
RATS	http://www.securesoftware.com
FlawFinder	http://www.dwheeler.com/flawfinder
Microsoft's FXCop	http://www.gotdotnet.com/team/fxcop
Split	http://splint.org/
Boon	http://www.cs.berkeley.edu/~daw/boon/
Pscan	http://www.striker.ottawa.on.ca/~aland/pscan/

Commercial

Analyzer	URL
Fortify	http://www.fortifysoftware.com
Ounce labs Prexis	http://www.ouncelabs.com
GrammaTech	http://www.grammatech.com
ParaSoft	http://www.parasoft.com
ITS4	http://www.cigital.com/its4/
CodeWizard	http://www.parasoft.com/products/wizard/

Black Box Scanners

Open Source

Scanner	URL
SPIKE	http://www.immunitysec.com
WebScarab	http://www.owasp.org
Paros	http://www.proofsecure.com

Commercial

Scanner	URL
ScanDo	http://www.kavado.com
WebSleuth	http://www.sandsprite.com
SPI Dynamics	http://www.spidynamics.com

Other Tools

Runtime Analysis

Analyzer	URL
Rational PurifyPlus	http://www-306.ibm.com/software/awdtools

Binary Analysis

Analyzer	URL
BugScam	http://sourceforge.net/projects/bugscam
BugScan	http://www.hbgary.com

Requirements Management

Manager	URL
Rational Requisite Pro	http://www-306.ibm.com/software/awdtools/reqpro

Appendix B: Suggested Reading

Whitepapers

- *Security in the SDLC (NIST)* http://csrc.nist.gov/publications/nistpubs/800-64/NIST-SP800-64.pdf Note: Need to change to official link.

- *The OWASP Guide to Building Secure Web Applications (Version 1.0)* http://www.owasp.org/documentation/guide

- *The OWASP Guide to Building Secure Web Applications (Working Draft Version 2.0)* http://www.owasp.org/documentation/guide current

- *The Economic Impacts of Inadequate Infrastructure for Software Testing* http://www.nist.gov/director/prog-ofc/report02-3.pdf

- *Threats and Countermeasures – Improving Web Application Security* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/threatcounter.asp

- *The Security of Applications: Not All Are Created Equal* http://www.atstake.com/research/reports/acrobat/atstake_app_unequal.pdf

- *The Security of Applications Reloaded* http://www.atstake.com/research/reports/acrobat/atstake_app_reloaded.pdf

- *Use Cases: Just the FAQs and Answers* http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/jan03/UseCaseFAQS_TheRationalEdge_Jan2003.pdf

Books

- Beizer, Boris, *Software Testing Techniques*, 2nd Edition, © 1990 International Thomson Computer Press, ISBN 0442206720
- *Secure Coding,* by Mark Graff and Ken Van Wyk, published by O'Reilly, ISBN 0596002424*(2003)* http://www.securecoding.org

- *Building Secure Software: How to Avoid Security Problems the Right Way*, by Gary McGraw and John Viega, published by Addison-Wesley Pub Co, ISBN 020172152X *(2002)* http://www.buildingsecuresoftware.com

- *Writing Secure Code,* by Mike Howard and David LeBlanc, published by Microsoft Press, ISBN 0735617228 (2003)
  http://www.microsoft.com/mspress/books/5957.asp

- *Innocent Code: A Security Wake-Up Call for Web Programmers,* by Sverre Huseby, published by John Wiley & Sons, ISBN 0470857447(2004)
  http://innocentcode.thathost.com

- *Exploiting Software: How to Break Code,* by Gary McGraw and Greg Hoglund, published by Addison-Wesley Pub Co, ISBN 0201786958 (2004)
  http://www.exploitingsoftware.com

- *Secure Programming for Linux and Unix HOWTO, David Wheeler (2004)*
  http://www.dwheeler.com/secure-programs/

- *Mastering the Requirements Process,* by Suzanne Robertson and James Robertsonn, published by Addison-Wesley Professional, ISBN 0201360462
  http://www.systemsguild.com/GuildSite/Robs/RMPBookPage.html

- *The Unified Modeling Language – A User Guide*
  http://www.awprofessional.com/catalog/product.asp?product_id=%7B9A2EC551-6B8D-4EBC-A67E-84B883C6119F%7D

- *Web Applications (Hacking Exposed)* by Joel Scambray and Mike Shema, published by McGraw-Hill Osborne Media, ISBN 007222438X

- *Software Testing In The Real World (Acm Press Books)*
  by Edward Kit, published by Addison-Wesley Professional, ISBN 0201877562 (1995)

- *Securing Java,* by Gary McGraw, Edward W. Felten, published by Wiley, ISBN 047131952X (1999)
  http://www.securingjava.com/

Articles

- *Web Application Security is Not an Oxy-Moron, by Mark Curphey*
  http://www.sbq.com/sbq/app_security/index.html

  Software Security Testing – Back to Basics (The OWASP Testing Framework) – Mark Curphey
  http://softwaremag.com

Useful Websites

- OWASP — http://www.owasp.org

- Secure Coding — http://www.securecoding.org

- Secure Coding Guidelines for the .NET Framework
  http://msdn.microsoft.com/security/securecode/bestpractices/default.aspx?pull=/library/en-us/dnnetsec/html/seccodeguide.asp

- Security in the Java platform — http://java.sun.com/security/

- Sardonix — http://www.sardonix.org
- OASIS WAS XML — http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=was

Additional resources are available at http://www.securecoding.org/companion/links.php

Index