

Best Practices for Application Security:
Prevent the New Breed of Attacks and Comply with Regulatory Security Requirements



Many organizations—including organizations that have already invested considerable resources to safeguard system security—are unprotected against a new breed of security attacks. Security measures such as implementing firewalls, authentication and access control systems, network intrusion detection/prevention systems, anti-virus solutions, anti-spyware solutions, and Secure Sockets Layer are indeed critical. Yet, these safeguards alone are no longer sufficient. Attackers now expect these basic security safeguards to be in place, and attack the system in a new way: via internally-developed custom applications.

Application attacks are responsible for over 75% of recent security attacks. These attacks are not only common, but also severe. If an application has security vulnerabilities, it can allow an attacker to access privileged data, delete critical data, and even break into the system and operate at the same priority level as the application—giving the attacker the power to destroy the entire system. Securing the network, OS, and server but neglecting to secure the application is like building an elaborate fortress, but leaving its main gate open and unguarded.

Even if an application is protected against standard application security attacks (including SQL injection, parameter manipulation, buffer overflows, cross-site scripting, and so on) it may still be vulnerable to a dangerous new breed of specialized application attacks: attacks that take advantage of flawed application logic. For example, consider the recent attack where a retail application storing unnecessary credit card details permitted credit card counterfeiting. Or, the attack where the application gave many user accounts search privileges inappropriate for their roles; attackers managed to access accounts that had these excessive search privileges and use these accounts to potentially access hundreds of thousands of "confidential" personal records.

The most common approach to securing applications is not the optimal way to prevent standard application attacks, and it does little to prevent the application logic attacks that are becoming attackers' tool of choice. The industry's common response to securing applications has been to try to test security into the application at the end of the development process. But this approach fails to address the root cause of the problem: security, like quality, must be built into the application. Testing standard security vulnerabilities out of the application at the end of the development process—the stage when finding and fixing bugs is most difficult, time-consuming, and costly—is both inefficient and largely ineffective. Common "end-of-cycle" security testing techniques can detect some standard application security vulnerabilities. These vulnerabilities must be fixed, but fixing them alone won't ensure security. If the application logic was not designed and built with security in mind, attackers may still exploit the application using a specialized attack scenario.

Building security into an application involves designing and implementing the application in a way that reduces the risk of security attacks, then verifying that the policy is implemented and operating correctly. Essentially, security becomes a specification issue. If the specification does not define how the application should be built to safeguard security, the application will be vulnerable to the types of sophisticated attacks that are starting to emerge now and will become increasingly prevalent in the future, when common application vulnerabilities are protected and the application logic becomes attackers' only real entryway into the system. And if that specification is not a living document that is actually implemented in the code, the organization will be as vulnerable to attacks as it would without a security policy.

Parasoft's security solution delivers an integrated set of services and products that help organizations develop, implement, and verify effective security policies. In doing so, it helps organizations achieve the application security that is required to prevent today's and tomorrow's sophisticated security attacks, as well as to comply with security/privacy requirements for Sarbanes-Oxley, the Payment Card Industry, HIPAA, and other regulations. The core solution provides each organization a custom security policy as well as the technologies and training to ensure that the policy is implemented in the code and operates correctly. Each custom solution is tailored to suit the organization's unique goals, projects, and compliance needs. Listed below are the available solution components, which are described in the following sections:

- Security Policy Development
- Source Code Analysis
- Penetration Testing
- Runtime Analysis
- Automated Regression Testing
- Centralized Reporting
- Training

Security Policy Development — To Ensure that Security is Built into the Application

Security policies are espoused by security experts, such as OWASP, and are mandated for compliance with many regulations, such as Sarbanes-Oxley, that require organizations to demonstrate they have taken "due diligence" in safeguarding application security and information privacy. Yet, although the term is mentioned frequently, it is not often defined. A security policy is a specification document that defines how code needs to be written and tested to protect it from attacks. Security policies typically include custom security requirements, privacy requirements, security coding best practices, security application design rules, and security testing benchmarks. Ideally, security policies should also require that all security-related operations be concentrated in one segment of the application. You can then focus your resources on verifying and maintaining the security of that one critical module. This centralized security acts like a drawbridge for a castle: it isolates the area that attackers can exploit and permits a more focused defense strategy.

Parasoft Professional Services helps organizations develop effective security policies. The security policy is typically developed as a two-step procedure. First, Parasoft reviews the organization's existing application security policy. If no application security policy is currently defined, Parasoft works with the organization to develop one. This policy typically describes what types of resources require privileged access, what kind of actions should be logged, what kind of inputs should be validated, and other security concerns specific to the application. Next, the existing security policy is used to customize and extend the Parasoft security policy, which provides a framework for enforcing security constraints. The security policy consists of core rules based on application security best practices. Application-specific rules that address the organization's specific security concerns are added to extend and customize the core set of rules. This process produces a document that enforces the centralization of security mechanisms, prevents coding problems that can lead to security vulnerabilities, and details custom security requirements.

Source Code Analysis — To Verify that the Security Policy is Implemented in the Code

Having an effective security policy defined on paper will not translate to a secure application unless the security policy is followed during development. Static analysis can be used to automatically verify whether most security policy requirements are actually implemented in the code and identify code that requires rework. Additionally, static analysis can identify common coding issues that make code vulnerable to buffer overflows, SQL injection, and other security breaches. For example, in Java, using `PreparedStatement` is recommended over using plain `Statement` to prevent SQL injections; a static analysis rule that searches for `Statement.executeQuery()` invoked with a dynamic string can pinpoint this statement and provide a first line of defense against SQL injection problems. Additionally, static analysis can also identify suspicious code patterns that might indicate hidden Trojans, easter eggs, or time bombs.

Parasoft Jtest, C++Test, and .TEST can be used to automatically verify that the security policy is implemented in Java, C/C++, or .NET language code, as well as check whether the code follows rules for preventing common security vulnerabilities and identifying malicious code.

The implementation of custom security policy requirements (for instance, for authentication, authorization, logging, and input validation) can be verified by using the products to create and

check custom security rules that verify those requirements. Organizations can create custom rules using the Parasoft RuleWizard rule creation GUI, which allows users to define custom rules graphically (by creating a flow-chart-like representation of the rule) or automatically (by providing code that demonstrates a sample rule violation). Alternatively, Parasoft Professional Services can design and implement the custom rules.

Additionally, Jtest, C++Test, and .TEST can automatically check whether code complies with a set of general security best practices developed for the applicable language. For instance, Jtest checks industry-standard Java security rules such as:

- Use JAAS in a single, centralized authentication mechanism.
- Do not cause deadlocks by calling a synchronized method from a synchronized method.
- Use only strong cryptographic algorithms.
- Validate the `HttpServletRequest` object when extracting data from it.
- Session tokens should expire.

In many cases, the product can automatically correct the code to remove selected security vulnerabilities. Identifying and removing security vulnerabilities in this manner takes just seconds, allowing significant security improvements to be made with minimal resources.

Penetration Testing — To Verify that the Security Policy Operates Correctly

Once you are confident that the security policy is implemented in the code, a smoke test is needed to verify whether the security mechanisms operate correctly. This is done through penetration testing, which involves manually or automatically trying to mimic an attacker's actions and checking if any tested scenarios result in security breaches. When penetration testing is performed in this manner, it can provide reasonable assurance of the application's security after it has verified just several paths through each security-related function.

If the security policy was enforced using static analysis, the penetration testing should reveal only problems related to security policy requirements that cannot be enforced through static analysis (for instance, requirements involving Perl). If problems are identified, either the security policy must be refined, or the code is not functioning correctly and needs to be corrected. In the latter case, locating the source of the problem will be simplified if the code's security operations are centralized (as required by the recommended security policy).

Two levels of penetration testing are recommended. First, to determine whether the application is vulnerable to common attacks, use basic penetration testing, which simulates the types of attacks that could be launched on any application. Next, to determine whether the application is vulnerable to unique attacks (such as attacks on application logic), design and execute penetration tests that attempt to subvert your application's unique security mechanisms (as defined in your security policy). This second level of penetration testing is often overlooked, but critical to verifying that an application can resist the application logic attacks that are now becoming increasingly common.

Parasoft WebKing facilitates both levels of penetration testing for Web applications and Parasoft SOAPtest facilitates both levels of penetration testing for Web services. WebKing's standard (out-of-the-box) penetration testing functionality can be used to automatically identify common Web application vulnerabilities that would violate any Web application security policy. While spidering a site or recording a path, WebKing can automatically create security tests that attempt to exploit the following vulnerabilities:

- **SQL Injections:** When SQL statements are dynamically created as software executes, there is an opportunity for a security breach by passing fixed inputs into the SQL statement, making them a part of the SQL statement. This could allow an attacker to gain access to privileged data, login to password-protected areas without a proper login,

remove database tables, add new entries to the database, or even login to an application with admin privileges.

- **Cross-Site Scripting:** Cross-site scripting problems occur when user-modifiable data is output verbatim to HTML. Subsequently, an attacker can submit script tags with malicious code, which is then executed on the client browser. This allows an attacker to deface a site, steal credentials of legitimate users, and gain access to private data.
- **Parameter Manipulation:** When input parameters to a Web application are not properly validated, it can lead to vulnerabilities in the underlying system. In native applications, buffer overflow attacks can occur when input parameter data sizes go unchecked. These vulnerabilities could cause system crashes or could even lead to unauthorized information being returned to the client application.
- **HTML-Level Vulnerabilities:** The HTML of the Web application's pages can make the application vulnerable to security issues such as interceptable passwords, Web bugs (Web page images designed to monitor who is reading the Web page), Active X controls, caching issues that can allow unauthorized users to access sensitive information, and comments that may divulge excessive or privileged information.

Likewise, SOAPtest's standard (out-of-the-box) penetration testing functionality can be used to automatically identify a number of common Web service security vulnerabilities that would violate any Web service security policy. Given a WSDL file, SOAPtest can automatically generate a suite of security tests that attempt to exploit the following vulnerabilities:

- **XML Bombs:** When using a DTD (Document Type Definition) within an XML Document, a Denial of Service attack can be executed by defining a recursive entity declaration that, when parsed, can quickly explode exponentially to a large number of XML elements. This can consume the XML parser resources, causing a denial of service.
- **SQL Injections:** Described above.
- **Parameter Manipulation:** Described above.

Additionally, both WebKing and SOAPtest can be easily customized to perform penetration tests that verify the operation of custom security policy requirements, including authentication, session management, access control, and more. In fact, the Parasoft Professional Services team has verified and documented how WebKing, along with the open source Nessus and Nikto tools, can be used to perform the various tasks detailed in the OWASP Penetration Testing Checklist. Parasoft Professional Services offers customized training on completing the checklist items.

Runtime Analysis — To Expose Buffer Overflows

When Web application or Web services are built with C/C++, memory corruption (especially memory corruption on the stack) indicates a potential for buffer overflows, which could cause serious security problems. For example, attackers can exploit these weaknesses so that a function returns to a hacker-designated function, or so that the function executes a hacker-designated operation. Memory leaks are also dangerous: they make an application more vulnerable to denial of service attacks.

Parasoft Insure++ automatically exposes memory corruption and memory leaks in C/C++ applications. During integration testing, the application is compiled with Insure++. As the application is being stimulated and exercised by the test suites, Insure++ identifies memory corruption—every chunk of memory is observed and every memory access is checked to verify whether it is legal and to determine whether it can overwrite the range, resulting in a buffer overflow. Insure++ also identifies memory leaks and pinpoints the line of source code that causes each leak. If the test suite thoroughly exercises the application, this process should expose most possibilities for memory corruption and leaks.

Automated Regression Testing — To Ensure Continued Security

To ensure that code remains secure as the application evolves, all security-related tests (including penetration tests, static analysis tests, runtime error-detection tests, and so on) should be added to a regression test suite, and this test suite should be run nightly during the automated build. Tests are then performed consistently, without disrupting the existing development process. If no problems are identified, no team intervention is required. If tests find that code modifications reintroduce previously-corrected security vulnerabilities or introduce new ones, the team is alerted immediately. This automated testing ensures that applications remain secure over time and also provides documented proof (such as that required for Sarbanes-Oxley compliance) that the application security policy has been enforced.

Additionally, each time a new security problem is identified, a test that detects or prevents that type of problem should be added to the regression test suite. Over time, this will result in significantly fewer security vulnerabilities slipping into the code.

All Parasoft products support regression testing. Parasoft Professional Services team can configure an automated regression testing infrastructure that runs automatically each night, as part of the automated build. Moreover, all Parasoft products can be customized to prevent the same types of errors from recurring.

Centralized Reporting — For Accessible, Documented Results

Security data gathered from the automated regression tests can be collected, analyzed, and organized by the Parasoft Group Reporting System (GRS). GRS serves as the brain of the organization's security testing efforts, and provides the organization with a user-friendly, non-disruptive method of fulfilling security reporting requirements, such as those for Sarbanes-Oxley. By automating security record keeping, statistical analysis, and reporting, it provides a centralized system for collecting and organizing all of the organization's security-related data—including data from third-party or internally-developed security verification technologies. Data from additional quality control efforts can also be sent to GRS and incorporated into the available reports.

The GRS relational database, which is typically MySQL on Linux but can be any relational database (Oracle, for example), collects data from all available security analyses. Next, the GRS reporting engine analyzes and correlates all available security error data to produce reports. Team members can then use a Web interface to access reports tailored to their role and responsibilities. For instance, developer dashboards provide each developer a customized report that helps him identify and correct the security problems that he has introduced into the code. The tester dashboard helps the testing team capture manual security tests as well as monitor the security bug fixes and new security features that require testing. The architect dashboard helps the team architect monitor project-wide technical security details. The manager dashboard provides the manager a concise, high-level summary of the project's security status.

With GRS, team members and managers always have a fast and easy way to assess the project's security status, identify problems as soon as they emerge, and respond to them immediately. This increased control improves productivity, reduces costs, and helps ensure that internally-developed applications are secure and reliable. Moreover, GRS's archived reports provide documented proof (such as that required for Sarbanes-Oxley compliance) that the application security policy has been enforced.

Training — To Teach the Skills and Strategies Needed to Implement and Verify Security

Parasoft Professional Services can design and conduct customized training courses designed specifically for each client's needs. Training can cover product usage as well as the "think like a hacker" strategies required to effectively verify whether unique security policy requirements are truly safe from attack. Commonly-requested courses include Product Introductory Training, Product Security Training, and Penetration Testing Bootcamp.

Contacting Parasoft

USA

101 E. Huntington Drive, 2nd Floor
Monrovia, CA 91016
Toll Free: (888) 305-0041
Tel: (626) 305-0041
Fax: (626) 305-3036
Email: info@parasoft.com
URL: www.parasoft.com

Europe

France: Tel: +33 (1) 64 89 26 00
UK: Tel: +44 (0)1923 858005
Germany: Tel: +49 7805 956 960
Email: info-europe@parasoft.com

Asia

Tel: +886 2 6636-8090
Email: info-psa@parasoft.com

© 2005 Parasoft Corporation

All rights reserved. Parasoft and all Parasoft products and services listed within are trademarks or registered trademarks of Parasoft Corporation. All other products, services, and companies are trademarks, registered trademarks, or servicemarks of their respective holders in the US and/or other countries.