Secure Coding.
Practical steps to defend your web apps.

Copyright SANS Institute
Author Retains Full Rights

This paper is from the SANS Software Security site. Reposting is not permitted without express written permission.

Interested in learning more?
Check out the list of upcoming events offering
"()"
at http://software-security.sans.orghttp://software-security.sans.org/events/
Securing Web Applications
Made Simple and Scalable

September 2013
A SANS Analyst Program Review
Written by Gregory Leonard
Advisor: Stephen Sims

Requirements PAGE 2
Configuration and Automation PAGE 3
Scanning Options PAGE 4
Reporting and Remediation PAGE 8
Reducing False Positives and Negatives PAGE 16
Simulating Advanced Attacks PAGE 18

©2013 SANS™ Institute
The weakest point in modern networking infrastructures is often not the low-level hardware and software running on the servers, but the web applications that are developed and hosted upon them. Older web applications were developed with few security requirements in mind at a time when protecting them against malicious users was of less concern than simply getting them to work. Web apps also tend to stick to the versions of third-party and open source software components they were first developed with, and upgrades that fix vulnerabilities aren’t always implemented. As old technologies remain stubbornly vulnerable, new methods of attack surface daily.

Poor understanding of web security needs on the part of developers (and the failure of security operations groups to properly relay the message in development terms) also limit the effectiveness of application security programs, according to the SANS survey on application security. Although nearly two-thirds of respondents to the 2012 SANS survey on application security have some sort of testing program in place for their applications, only 23 percent of them feel their policies are comprehensive.¹

All of these issues have resulted in the systemic life cycle management problems faced by security operations groups trying to promote better organizational application security. This problem can only be helped with the frequent use of automated testing tools on existing and new applications, according to penetration tester Jim Bird, as he discussed the results of the SANS application security survey.²

In July and August, SANS evaluated HP Fortify WebInspect 10.10 to see how much automated testing can help identify issues in a set of applications with known vulnerabilities for it to discover. WebInspect is a dynamic application security testing (DAST) tool that scans web applications while they run and enables development organizations to perform security tests against their web applications using the same techniques that attackers use to exploit vulnerabilities.

Our review also included the agent-based testing capability of WebInspect, which was enabled by the HP Fortify SecurityScope runtime analysis agent for Java applications and which can be attached to any application container. An agent for .NET applications is also included with the SecurityScope package, but it was not used in our testing.

¹ www.sans.org/reading-room/analysts-program/sans-survey-appsec
² http://software-security.sans.org/blog/2013/01/14/security-testing-less-but-more-often-can-make-a-big-difference
Fortunately, many organizations finally realize the impact of unsecured web applications and are, at last, seeking ways to mitigate the risks posed by these vulnerabilities. They struggle with creating appropriate workflow automation around their testing tools, as indicated in the SANS application security survey.

Effective automated testing tools need to have these characteristics:

- Ease of configuration and automation
- Support for a wide range of scans
- Detailed reporting and remediation guidance
- False positive/negative reduction
- Advanced attack simulation

Organizations should also be prepared to deal with various types of web applications, each of which requires different protective processes:

1. **Legacy Apps.** These web applications tend to be several years old and may be in maintenance-only mode. The applications likely use old or outdated versions of software components, whether these come from a third party, open source software or internal developers. In addition, these applications were unlikely to have been designed with security in mind and could have serious and unsuspected code bugs and operational vulnerabilities that are begging to be exploited.

2. **Current Apps.** These web applications are more modern than the first group, but may be suffering from the same design issues as their legacy siblings. Additionally, newer applications tend to rely heavily on JavaScript and other non-basic client-side components (e.g., Java applets, Flash or VBScript) that may be easily exploited.

3. **Inherited Apps.** Organizations continually acquire new web applications, whether from the purchase of another company or the installation of new software. The provenance of these inherited applications is often unknown, and they can present hitherto-unrecognized vulnerabilities. In many cases, source code for them may not be available, and domain-specific knowledge of them will at first be limited.
Our review of WebInspect began with installation on a Windows client and then moved quickly to the user interface. The installer prompts the user to specify a Microsoft SQL Server instance that will be used to store scan results. If such a database server is not available, the installer can install a copy of SQL Server 2008 Express on the same host as WebInspect. It will then prompt for an installation directory for WebInspect components and, when those bits are in place, hand off to the last installation wizard, which prompts the user to configure an existing application server where the SecurityScope agent will be installed.

The agent installer can update the app server’s configuration files on its own or terminate cleanly if manual configuration is preferred. (The installer does a good job for default app server setups. However, if the server’s startup files have been customized, manual setup is probably a safer choice.) If manual setup is selected, the agent installer will display a message listing the configuration values that need to be added and indicate to which files.

The WebInspect user interface is efficient yet informative, with a Start Page dashboard containing such high-level features as the following:

- A top-level ribbon with the standard Windows menu options (File, Edit, View and Help) above WebInspect’s core functions (New, Open, Report, Schedule and SmartUpdate) as well as its Compliance Manager and Policy Manager tools
- A Start Page with links for starting new scans or viewing scan results, including:
  - A button bar providing access to the Home pane, the Managed Scans pane, and the Manage Schedule pane
  - A left-side pane with links to WebInspect’s primary tools for scanning and reporting
  - A right-side pane with details relating to the selected button

Figure 1 shows WebInspect’s startup screen; the primary tools are in the left pane, while the detailed information appears to the right.
WebInspect provides four primary scanning options: Guided, Basic, Web Service and Enterprise. Guided and Basic scans both lead the user through configuring, initializing and running WebInspect scans against a specific web application.

The Guided scan is a step-by-step walkthrough of scan configuration, while a Basic scan collects the same information, but in a more concise fashion using a four-step workflow. Web Service scans are customized to target web services based on Web Services Description Language (WSDL) rather than conventional web apps. Finally, Enterprise scans can examine multiple websites in one job. The sites to be scanned can be added manually or from a network discovery. Unlike the other scan options, which all execute immediately, Enterprise scans can be scheduled for one-time or for recurring execution.

WebInspect provides several auditing options to choose from, including policies that target the Open Web Application Security Project (OWASP) Top Ten.3 These tests can employ aggressive forms of attacks, such as SQL injection, or even some potentially destructive tests that may write persistent information to the database or create denial-of-service conditions in the application. These audit options are powerful, so users must carefully consider the environment in which tests run to prevent an outage of production applications.

Getting Started

All scans begin with the user following the Scan Wizard and entering the information shown in Table 1.

3   https://www.owasp.org/index.php/Top_10_2013-Top_10

Table 1 continues
Authentication and Connectivity
This section configures WebInspect to connect to the target site and authenticate if required. Options include the use of a network proxy as well as network-based or site-specific authentication information.

For our review, we chose Site Authentication and provided credentials for the application being tested.

Coverage and Thoroughness
This section optimizes scan settings for sites that are built with Oracle ADF Faces or IBM WebSphere Portal by defining scan coverage (how deep into the site should WebInspect crawl) and the Audit policy to enforce.

For our review, we chose Standard audit depth, which will avoid any tests that might degrade production systems.

Detailed Scan Configuration
To make scan results more accurate or useful, the user can include test data for web forms, supply additional or supplementary URLs for examination, include a false-positive list designed to cut down on incorrect findings, or monitor tests through a web proxy.

For our review, we chose to Auto-fill web forms during the crawl, which will use dummy form data during testing. This will allow the crawl to dig deeper into the application and provide more thorough test results.

Table 1. Collected Scan Information
Type of Findings

The findings will vary by the type of scan; for example, a Crawl Only scan reports fewer findings than a Crawl & Audit targeting the same site. All scans are presented in the Scan Dashboard, showing the current scan results with a graphical summary.

Figure 2 shows a Crawl Only scan for one of our testbed’s websites.

Figure 3 shows a Crawl & Audit scan of the same site using identical parameters; key differences between the scan types are marked in red.
Crawl Only simply inspects the URLs found in the application for obvious issues (e.g., sensitive information disclosure or improper security configuration), but the Crawl & Audit attempts to exploit the application through various attack methods. As one would expect, a Crawl & Audit scan generates more findings than the same scan in Crawl Only mode.

In addition, the number of findings grows rapidly as the depth of the crawl and the audit intensity increases. This means filtering false positives and duplicate findings is crucial. We explore this later, under “Reducing False Positives and Negatives.”

**Consider Resource Costs**

Remember that audit processes can significantly increase the time needed to scan an application, so it’s important to first explore the “resource cost” of in-depth scanning. Although users might be tempted to launch the most in-depth scan possible with the most detailed auditing in all situations, the time required for such a scan may not work in large environments. Scanning too many systems at once can provide an overwhelming number of results and make remediation difficult. Limited scanning followed by prompt remediation is often more beneficial than widespread scanning with delayed remediation.

**Scan-and-Fix Scheduling**

Fortunately, a four-prong approach that stages the scan-and-fix effort covers even the largest organization:

1. Crawl Only scans are essential to uncover security issues, but the execution frequency may need to be limited to milestones in an application’s life cycle.

2. For more frequent protection, organizations can set up more frequent scans, with tailored audit rules and limited crawl depth. These will still be successful at finding major issues, but will not burden developers by making them wait unduly for scans to complete.

3. The first two schemes are well suited to development environments, but it is often desirable to test the production environment as well. Again, WebInspect can provide a separate audit configuration that targets specific threats that is easy on a host’s resources.

4. SecurityScope can also provide runtime information that can be stored in a log file and monitored to ensure production systems are not being attacked.
Now that we have looked at the ways to configure and execute a scan, let’s take a closer look at the findings presented by WebInspect’s Scan Dashboard. Figure 4 shows the expanded results from the previous Crawl & Audit example, which display the specific attacks WebInspect used during the scan.

![Scan Dashboard](image)

**Figure 4. Attack Types in a Crawl & Audit Scan**

We see that our scan uncovered two critical Keyword Search vulnerabilities. In this case, the test inserted SQL code into the username and password fields of the target system’s login page and discovered SQL injection vulnerabilities in each case.
The WebInspect interface provides a simple-to-read listing of all of the attacks sent against the target site and the vulnerabilities the scan uncovered. The results include a detailed listing that can be selected for review. Right-clicking on a specific vulnerability presents the user with several options, as indicated by the red outline in Figure 5.

Several Scan Dashboard options are useful for reviewing vulnerabilities and determining the impact they could have on an application; these are detailed in Table 2.

**Edit Vulnerabilities**

This option enables the user to change the severity, category and probability of the vulnerability, if the organization’s policy does not conform to the default values. This provides more accurate reporting, leading to appropriate prioritization of the findings. In this case, we see the details for the categorization of the “Logins Sent Over Unencrypted Connection” vulnerability.
Retest Vulnerabilities
This screen provides detailed request/response and result information for a selected vulnerability, a detailed description of the vulnerability and possible remediation information.

In this case, we see two errors for the testbed site’s userlogin.jsp page. The details of the first one are shown, containing a general-purpose error message indicating when our test ran and the Common Weakness Enumeration (CWE) for the vulnerability.

Mark as False Positive
This enables the user to acknowledge that the finding was reviewed. After further investigation, the finding has been deemed a false positive and can safely be ignored.

In this case, we see a false positive report of a cross-frame scripting attack when none was in progress.

HTTP Editor
This enables the analyst to modify the content and syntax of HTTP attacks. Doing so can help determine the full scope of a vulnerability and lead to the most accurate fix. When selected from the Scan Dashboard, the HTTP Editor is prepopulated with data from the scan being reviewed.

In this case we are inspecting a request that caused us to detect a vulnerability on the homepage of the target application.

Table 2 continues
**SQL Injector**

This enables the user to examine a basic SQL attack in WebInspect’s library and expand it to probe in detail for vulnerabilities. Doing so can help determine the full scope of the vulnerabilities, such as revealing the ability to extract information from the database or compromise database integrity in other ways.

In this case, we are attempting to exploit a SQL injection vulnerability found in the target website’s login page.

**Add Vulnerability Note**

This section allowed us to add detailed notes that are attached to vulnerabilities discovered during a scan. This is helpful for capturing information to be shared between application testers or between review sessions. Being able to take accurate notes during review enables more accurate analysis to be performed later.

In this case, notes for a discovered Cross-Frame Scripting vulnerability can be added to the “Description” section, which appears in the lower pane of the screen.

---

**Table 2. Scan Dashboard Features**

When a scan reveals vulnerabilities, the user can click on a vulnerability’s entry in the Scan Dashboard to see the essential information about a vulnerable web page, including a detailed description of the vulnerability, its implications for the application and suggested fixes. In some cases, reference information is available with further discussion of the vulnerability or possible fixes.
WebInspect reports all flaws found for a URL in a summary containing links to details of individual vulnerabilities. Figure 6 shows a view of a URL with several vulnerabilities.

**Figure 6. Vulnerability Summary View**

By clicking on the third entry (Unencrypted Login Form) in the summary view, we get details of a specific vulnerability, shown in Figure 7.

**Figure 7. Vulnerability Detail View**
In addition to the defect information provided in the Scan Dashboard, WebInspect provides a large array of reports that can provide highly detailed forensic information for the security operations group's use or high-level summaries for nontechnical executives. Figure 8 shows the list of standard reports WebInspect can generate.

The Advanced button in the lower left corner enables the analyst to configure report options such as headers, footers and custom logos. Although no ad hoc reporting features appear to be available, the reports go into sufficient detail for most purposes. Figure 9 shows an example of a high-detail vulnerability report.
Figure 10 shows an executive summary report.

![Sample Executive Summary Report](image)

Each report shows the scan results in a unique way. Review all of the report types and determine which is best for the organization’s needs. Some examples of useful reports include:

- **Vulnerability.** This report is useful for developers. It provides the URL that was called with the entire HTTP request and response that was exposed as a vulnerability, including highlighting the specific content that caused the content to be marked as vulnerable.

- **Compliance.** This report is useful for development organizations that are responsible for conforming to various compliance standards. The report identifies all the rules for the selected standard and summarizes the application's compliance with that standard. Some key compliance standards supported by this report include those defined by the Federal Information Security Management Act (FISMA), HIPAA, National Institute of Standards and Technology (NIST) 800-53, the OWASP Top Ten and PCI.

- **Executive Summary.** This report is useful for high-level presentations. It consists of a one-page summary with key reporting information, including a summary of vulnerabilities by severity, a summary of vulnerabilities by threat class (such as cross-site scripting or SQL injection) and a breakdown of the website by file extensions found (e.g., HTML, ASP or TXT) and site components (comments, cookies, forms, etc.).
Developer Integration

Testing applications under development for security issues is a must, and quickly reporting vulnerabilities to developers as a defect can help reduce the time needed to remediate such flaws. In the case of production web apps, it is critical to close known issues as quickly as possible. The “Send To” option in WebInspect’s Scan Dashboard provides direct integration with HP Quality Center, HP Application Lifecycle Management and IBM Rational ClearQuest, allowing the analyst to begin creating a ticket for the development team with a description of the problem as a software defect. (Note: We did not examine this feature during our review.)

Figure 11 shows an HP Application Lifecycle Management dashboard with security vulnerabilities reported by WebInspect.
Reducing False Positives and Negatives

False positives waste valuable time, taking testers away from projects to review findings that ultimately prove bogus. A high rate of false positives creates doubt around a DAST tool’s capabilities and usefulness. If a large percentage of findings prove to be false positives, could other findings be unconfirmed false positives? Questions such as this reduce an organization’s inclination to use scanning tools.

On the other hand, we have false negatives, in which testing doesn’t pick up an existing vulnerability, leading to a false sense of security. Undiscovered or unsuspected vulnerabilities in a production application make it vulnerable to exploitation by attackers. If an organization relies on a DAST tool and falls victim to such vulnerabilities, developers and testers lose faith in that tool and may quickly abandon it.

**Addressing False Positives**

WebInspect provides the ability to mark findings as false positives or to ignore them. In addition to allowing testers to focus only on relevant vulnerabilities, WebInspect’s handling of false positives enables the tester to send information about incorrect assessments back to HP. (This is designed to help the product developers to fine tune their audit algorithms, and will, presumably, lead to improved false positive performance in future releases.)

Once a false positive is flagged, subsequent scans focus only on new findings. This provides a smaller footprint of findings to review, thus reducing the time it takes to correct potentially damaging problems. Figure 12 shows a second scan against the test site, after several of the original findings were marked as being false positives.

![Figure 12. Results Following Rescan upon False Positive](image-url)
Addressing False Negatives

To help mitigate false negatives, we enabled the Fortify SecurityScope feature of WebInspect to perform two critical functions:

1. **Monitoring all HTTP request/response traffic with the application server.** SecurityScope reports all malicious traffic that is detected, showing what specific lines of code are involved and the flow of events that lead up to the exposure. It reports as much detail as possible about the vulnerability so that problems can be pinpointed as quickly as possible during a forensic review.

2. **Providing real-time feedback about attacks being performed.** This feedback enables WebInspect to launch stronger simulated attacks as well as reduce false positives through confirmation of attack results.

Using SecurityScope as a “man on the inside” agent enables a thorough scan of a site. Let’s return to the site scan we used in previous examples. Figure 13 shows a screenshot of the exact same Crawl & Audit scan that was executed on the target site, only this time SecurityScope has been enabled on the application server.

The audit tests now get two perspectives to review testing data: WebInspect’s report of conditions outside the application, and SecurityScope’s report of targeted scans from the perspective of the application server.

This “both sides of the coin” view provides a much more accurate picture of the site’s vulnerabilities, but it comes with a certain resource cost. Our scan with SecurityScope enabled took much longer to execute than the same scan without it, due in part to the additional detail inherent in its use. In our example, the combination of WebInspect with SecurityScope enabled invoked 780 audit tests versus 46 audit tests in the same scan with just WebInspect. (This provides an excellent demonstration of why such detailed examination is best suited for preproduction environments, rather than being executed against servers relied on for line-of-business use.)
WebInspect provides a series of tools that can be utilized by penetration testing experts to do a deep analysis of a site. WebInspect provides 13 such tools, all of which can be especially useful when generating advanced attacks against a test site. Table 3 lists some of the most useful tools.

### Cookie Cruncher

Some web application cookies, such as session IDs, ought to be random. Sites with poor randomization are vulnerable to guessing attacks, such as some forms of session hijacking. Cookie Cruncher takes a URL, runs a sample set of requests against it, and then analyzes the cookies returned from those requests for randomness. It will indicate if those values are not random enough.

### Encoders/Decoders

Web applications use many forms of encoding and encryption, including:
- Base64 encoding
- Hashing (e.g., MD5 or SHA1)
- Encryption (e.g., 3DES or Blowfish)

The Encoders/Decoders tool assists testers by enabling them to take values and perform the necessary encoding to try and reproduce the values represented in the application, or by trying to decode/decrypt application data. This simulates some of the work intruders may perform while attacking a site.

Table 3 continues
HTTP Editor
As illustrated in Table 2, the HTTP Editor enables a tester to throw arbitrary HTTP request values at a site, similar to what can be done by intercepting a live request using a proxy. This allows the tester time to craft the request data without having to worry about a timeout for a live request.

When selected from the Tools menu, the HTTP Editor brings up a simple template of request parameters, rather than the pre-populated form shown in Table 2.

Regular Expression Editor
Regular expressions are a powerful way to create search strings for parsing and examining text. The Regular Expression Editor can be used to create regular expressions and then filter a body of text through them, as shown in the second column. This can help testers in parsing bulky HTML code to dig out specific fields of interest or to look for specific text formats (such as Social Security numbers).

Table 3 continues
Web Brute
This enables a user to scan all the forms in an application and perform brute force attacks against certain input fields. The tester can throw a large dictionary of test data at an application with a little simple configuration.

Web Fuzzer
“Fuzzing” is the testing of a web application’s input validation routines by taking expected values and inserting unusual characters—for example, providing an input that expects only integer values with the values “3,” or “123 a.” In some cases, fuzzing can expose denial-of-service or privilege escalation vulnerabilities. Common fuzzing mutations include checks for directory traversal, command injection, buffer overflows, SQL injection and other common vulnerabilities.

Table 3 continues
Web Proxy
This is an intermediary for HTTP requests, enabling the user to modify HTTP request/response values before transmitting them to the server/browser. Testers can use this tool to perform parameter manipulation exploits and to attempt bypasses of a web application’s client-side validation.

Table 3. Useful Tools in WebInspect
Web application attacks are one of the primary techniques used by intruders and data thieves. The most effective way to combat such attacks is establishing secure coding practices in the organization’s software development life cycle. Adopting a secure coding mindset from scratch can be daunting, so it helps to begin with a relatively painless starting point and grow from there. Implementing a well-designed dynamic testing tool in the development process can be an effective first step to achieving application security. DAST addresses the software cases outlined in the introduction in the following ways:

1. **Legacy Apps.** Dynamic testing is especially useful for the initial securing of a legacy system because it does not require recompilation of existing code, does not depend on any specific underlying technology (after all, all web applications are ultimately rendered as a combination of HTML, CSS, JavaScript and a few other technologies), and can be run against production systems without causing damage.

2. **Current Apps.** Implementing a DAST tool early in the development life cycle enables quick remediation of defects, can prevent vulnerabilities before they ever go live in production where they can be extremely costly, and can be run in nonproduction environments with end-to-end or destructive tests.

3. **Inherited Apps.** A DAST tool allows an organization that is inheriting a new application to understand which vulnerabilities exist as a starting point for building domain knowledge of such a system. This knowledge can enable the organization to make secondary decisions about the inherited system (e.g., removing the system from production or setting up an application firewall) before making the necessary vulnerability fixes.

HP Fortify WebInspect, especially with SecurityScope enabled, hits all of the key points that are necessary for DAST to be successful and effective.
Gregory Leonard has more than 13 years of experience in software development, with an emphasis on writing large-scale enterprise applications. He has worked with several government agencies, including designing and developing applications for the U.S. Treasury. Greg’s current responsibilities include application architecture and security; performing infrastructure design and implementation, security analysis, code reviews; and evaluating performance diagnostics.

SANS would like to thank its sponsor:
Upcoming SANS App Sec Training