A Security Checklist for Web Application Design

Gail Bayse

A Security Checklist for Web Application Design

Gail Zemanek Bayse

GIAC Security Essential Certification (GSEC)
Practical Assignment, Version 1.4b

Abstract

Web applications are very enticing to corporations. They provide quick access to corporate resources; user-friendly interfaces, and deployment to remote users is effortless. For the very same reasons web applications can be a serious security risk to the corporation. Unauthorized users can find the same benefits: “quick access,” “user-friendly,” and “effortless” access to corporate data.

This paper is written for Information Technology professionals who are not programmers and may not be aware of the specific problems presented when using an externally facing web application to attach to a mission critical database. The content provides a description of the security challenges introduced by externally facing web applications. It provides the knowledge necessary to articulate to developers the security requirements for a specific web application, to make contractual the obligation of the developer to build an application that is secure, and to assure that appropriate testing is completed prior to moving to a production environment. The document is structured as a checklist of challenges. For each challenge there are specific checkpoints that delineate the security concern. The checklist provides a basis for securing web applications and the databases they connect to from malicious and unintentional abuse.

Checklist

Risk Assessment

Authentication

Authorization and Access Control

Session Management

Data and InPut Validation

Cross Site Scripting (XSS)

Command Injection Flaws

Buffer Overflows

Error Handling

Logging

Remote Administration

Web Application and Server Configuration

Risk Assessment

Challenge:

Not all applications used in a secured Local Area Network (LAN) present additional security risk. It is important to match the security requirements with the risk imposed by the new application. An application that is used by employees solely from within the LAN and streamlines tasks already part of their functional role may require no additional security. However, an externally facing web application used by remote employees, consultants or vendors, attaching to a mission critical database poses a very different set of concerns. Every data asset must be examined and its confidentiality, criticality and vulnerability assessed. It is crucial to develop security procedures that are appropriate to each asset's criticality and vulnerability. "Security is almost always an overhead, either in cost or performance."[1] Therefore, the goal is to match the level of security with the assessed risk to assure that latency caused by security and the dollar amount spent securing an application are realistic and acceptable.

An application determined to be of risk to mission critical data will require a thorough security component during its' design phase, in development and implementation, and into maintenance. Use the following questions as checkpoints to determine the level of risk posed and the requisite security layers to be added.

Checkpoints:

- Which applications are affected by the requested change?
- Who are the users? Where are the users physically located?
- Will the application attach to mission critical applications? Will it modify any confidential or critical data?
- Where should additional user authentication be built into the application?
- Where will the application be physically located in the network? In the DMZ, the internal network? Will it be installed on new equipment or share an existing server? Will it coexist well with existing applications?
- Will any data considered sensitive or confidential be transmitted over external communication links?
- If the system was compromised would it result in financial loss or the loss of reputation? Can you place a dollar amount on any loss?
- What is the history of the OS platform with respect to security?
- What would motivate someone to break into the application?
- Will the application have high external visibility, making it an obvious target to attackers?

Authentication

Challenge:

Authentication is a first line of defense. The application must determine if the user is who he/she claims to be or if the entity, a server or program, is what it claims to be. This is the “I recognize who you are.” stage. The most common form of authentication is the user id and password. Authentication policies, processes, and logging must be designed, developed and documented to assure that the application keeps unauthorized users from accessing the site. It must correctly identify the true owner of a user id and password.

Checkpoints:

To prevent a user id and/or password from being hacked, failed logins should trigger a lock-out after a determined number of attempts. The account lock-out should be maintained for a number of hours to prevent and discourage the attacker from reissuing the attack. The activity should be logged.[2] All authentication attempts should be logged – log in, log outs, failed logins, password change requests. In addition notification or alerts should be sent to an administrator when the account is locked due to failed logins.

An attacker should not be able to deduce the user id. “Bill.Johnson” would not be considered a strong user id, while “bijohn” would be less obvious. Likewise, strong password rules should be applied. A strong password has a minimum of seven characters and it uses three of the following: numbers, upper case letters, lower case letters, and symbols. It will have a symbol character in the second – sixth position. A strong password will not use repeated or sequenced characters. It will look random.[3] Finally, the password should not be found in any dictionary.

Implement a password expiry time for all passwords. The more critical an application is deemed, the more often the password should change. For applications requiring a highly secure system, consider two-factor authentication. This is a token or fob with a code that automatically changes every 60 seconds. It is something you have (the token’s changing code), and something you know (a pin or passcode), and your user id.[4]

When a password is changed, require the existing password to be entered prior to accepting a new password. It is important to verify that the owner of the user id is the person requesting the password change. When passwords are successfully changed the program should forward a message to the email address of the owner of the user id, and the user should be forced to re-authenticate.

When a user forgets a password, the password must be changed rather than “recovered.” Passwords should not be stored in a manner that would allow a recovery. On form based password resets, the use of “secret” questions and answers is recommended. Again, the application should force a new authentication following the password reset.[5]

Passwords and user ids must be transmitted and stored in a secure manner. Do not send user ids and passwords in a clear-text email message. Should this be a necessity, any passwords sent in clear text must be encrypted using Secure Socket Layer (SSL).

Any list of passwords should be hashed (one-way hash) to assure that an attacker is unable to read authentication information. For applications that require intense security, consider combining a randomly generated salt value with the password hash. Salt is "random data …included as part of a session key. Salt values are added to increase the work required to mount a brute-force (dictionary) attack" [6] against authentication credentials. If user accounts are required to be exposed, i.e., in a drop down box, then an alias must be used to protect the user id.

Best practice recommends encrypting the entire logon transaction with SSL. Forms-based authentication must use a POST request to assure that the authentication credentials are not cached to browser history.[7] Using SSL on all login pages will accomplish this. Make use of "no cache" tags to further prevent someone from backing up to a login page and resubmitting a logon. Do not allow the application to cache both user id and password. "Remember me" functionality is not recommended, but if used should allow users to automatically sign in only to non-critical portions of the site.

Note on SSL:
SSL can provide authentication, confidentiality, and integrity for data as it is transported to and from web services. It is important to recognize that SSL doesn't protect the web application. It protects the "transport" of the data as it moves between the web application server and a browser. SSL is a transport layer protocol that operates between the TCP/IP layer and the Application layer where HTTP operates. Being aware of where SSL encryption is implemented will reduce any false sense of security developers may have when using SSL to secure an application.

Authorization and Access Control

Challenge:

Authentication tells a user "I recognize you as a user." Authorization says "Now that I know who you are I also know what you are allowed to do; what data you are allowed to see and modify." Access control determines where a user can connect from; what time they can connect, and the type of encryption required. The goal is to develop a security strategy to protect back-end and front-end data and systems. This can be accomplished through the use of roles, credentials, and sensitivity labels.[8]

Checkpoints:

During the design phase, user roles should be defined based on a "least privilege" model. If a user role will not be modifying data, then the role should not be given any opportunity to edit, delete, or add data to the critical database. Document the user roles during development and determine who will hold the responsibility for assigning users to specific roles.

The designers of the web application should have available to them complete documentation of the mission critical database at the conception of the project's design. It should include a description of all fields and tables, data length and expected values for a field, and any permissions assigned to the field.

Assure that users cannot "browse" past their user role rights. The user should not be able to access an unauthorized page by entering the location into the URL. Similarly, A user should not be able to enter a file path (\\servername\winnt\drivers) into a URL that would allow a user to access and potentially modify a system file.

Assure that users' activity is not cached when handling sensitive information. If multiple employees share a workstation, clicking the back arrow should not take a user to the URL of the last users' login or their last pages visited. This may result in elevated rights.

Use file system access rights only as a last defense.

Testing of an application prior to moving it into a production environment will include a review of user role documentation and a review of the code implementing access controls. Penetration testing will be necessary to assure that every access control has been tested and prevents unauthorized access.[9]

Session Management

Challenge:

A common vulnerability of web applications is caused by not protecting account credentials and session tokens.[10] There are four types of session id attacks: interception, prediction, brute-force, and fixation. In each attack, an unauthorized user can hijack a session and assume the valid user's identity. Encrypting sessions is effective against interception; randomly assigned session ids protect against prediction; long keyspaces render brute-force attack less successful, and forcing assignment and frequent regeneration of session ids make fixation less problematic.[11]

HTTP is a stateless protocol. It will answer any http request. HTTP does not, by itself, keep conversations in any specific order. It is important to use a state mechanism to separate and maintain an individual user's activities within a session. A cookie is a common vehicle used to maintain state in HTTP sessions. The cookie allows a user to make numerous HTTP transactions in one conversation. The session id keeps the various requests together in one conversation. [12]

Checkpoints:

Because cookies are transmitted in clear text, the content of the cookie must not contain or be used to obtain sensitive information. State mechanisms were not designed to manage sensitive information. Therefore, state mechanisms should not be used to authenticate users. The user must be made aware of and agree with the use the application will make of cookie sessions. The user must be able to immediately delete the cookie and the state associated with it. Any information stored within the cookie must not be disseminated to third parties without the users' consent.[13]

Session ids should be unique to users, and issued after successful authentication. They should be randomly generated using a respected randomization source. The session id should never contain personal information. Session ids are always assigned, never chosen by end user. The keyspace of the token must as large as possible to combat guessing and other attacks. A 12-digit keyspace has 1 trillion $(10^{12)}$ possible different codewords.[14] As bandwidth increases, the size of the keyspace must increase to keep hackers out. .[15]

Session ids must be protected throughout their life cycle to prevent hijacking.[16] They should have a time-out set for inactive sessions. Active sessions should also have a set time to expire and regenerate a new session token. This reduces the time window that a hacker would have to break into a session.

Session ids should be protected with SSL. Session ids should change routinely and always during major transitions, i.e., when moving to and from an SSL and when authenticating. For highly secure transactions re-authentication and a new session id should be issued prior to processing the requested transaction.

On log out, the session id should be over-written.

Data and InPut Validation

Challenge:

Cross-Site Scripting and Command Injection take advantage of a "violation of trust"[17] between a user accessing a known and trusted site and an attacker. The attacker bypasses security mechanisms by adding malicious code to open parameters in an application. An open parameter could be a URL, QueryString, Header, Cookie, Form Field, or a Hidden Field. It is any parameter that does not assure that the data entered is data that would normally be expected. For example, if the parameter is a date field, and the input "injected" into it is a script file, then the attacker has been successful in finding and using an open parameter. Well-written code would discard the script. The importance of knowing and documenting what is known as valid data cannot be stressed enough.

Checkpoints:

The strongest defense against these attacks is Input Validation. If the server validates all data entering the web application against known good criteria, the chances of successful attack are greatly reduced. The burden of security validation must fall on the server, and hence the application developer, rather than the client. Client-side validation is often used as a primary validation to “reduce round trips to the server,” but should not be used as a security defense. [18]

The use of a common library of field validations can be used to more efficiently and accurately confirm the integrity of the entry data.

- Constrain input – decide what is allowed in the field
- Validate data – type, length, format, and range
- Reject “known bad” input – do not rely only on this as it assumes the programmer knows everything that could possibly be malicious.
- Sanitize Input – This can include stripping a null from the end of a user-supplied string; escaping out values so they are treated as literals, and HTML or URL encoding to wrap data and treat it as a literal.[19]

Make strict use of canonicalization. Know what the server is expecting in every field. All data input must be reduced to a pure format, the format that the server and database expects. Input validation assures that all data is appropriate for its meaningful purpose. It may be necessary to establish character sets on the server to establish the canonical form that input must take.[20]

As noted in the Authentication section, an SSL connection is established in the transport layer, after the malicious code is introduced. It is important to recognize that SSL does not protect against invalid data. The SSL connection sees a valid conversation between server and user and transports the malicious code.

Cross Site Scripting (XSS)

Challenge:

When a web application creates output from user input without validating the data, the output can include malicious code. An attacker looks for instances in code where there is no validation and inserts the attack at that point. The output that the user receives may well carry malicious code. The user may receive a “click here” message, and trusting the “known site,” abide by the hackers desire. The result is directed at the end user rather than the application’s infrastructure. This could transmit corporate confidential data to an outside site. It can result in program installations or disclosure of end user files.[21]

Checkpoints:

All code must be reviewed for input variables that result in output and have no validation included. All headers, cookies, query strings, form fields, and hidden fields accepting input are validated against acceptable data lists. Every field must have a list of acceptable values.[22]

Replacing the following characters will also defeat XSS.

Replace	With
<	&lt
<	&gt
(	&#40
)	&#41
#	&#35
&	&#38 [23]

Command Injection Flaws

Challenge:

Command injection flaws allow attackers to relay malicious code through the web application to another system. The malicious code can include whole scripts. SQL injection is the most prevalent. SQL injection attaches specifically to a parameter that passes through to an SQL database allowing an attacker to modify, erase, copy, or corrupt an entire database. SQL Injections can take the following forms: Authorization (Authentication), Select Statement, Insert, and SQL Server Stored Procedures.[24]

Checkpoints:

Review for SQL injection is time consuming. All parameters must be examined for calls to external sources. Review the code for any instance where input from an HTTP request could be written into any of these external calls.

Build filters that verify that only expected data is included. If symbols are required, assure that they are converted to HTML.

Prepend and append a quote to all user input.

SQL server comes with a variety of stored procedure calls. Many are not used in specific applications. Give users access to only the SQL stored procedures that are required. All others should be stored away from the web application.[25]

Wherever possible avoid shell commands and system calls. In many cases there are language libraries that perform the same functions without using a system shell interpreter.[26] Where shell commands cannot be avoided, the code must validate the input against a valid input list to ensure that it does not include malicious code. Consider all supplied input as data, reducing, though not eliminating external calls.

In the event of data that is not acceptable, there should be a mechanism in place to block and time out the session.

Buffer Overflows

Challenge:
"The buffer overflow attack involves sending large amounts of data that exceed the quantities expected by the application within a given field.[27] Such attacks cause the application to abandon its normal behavior and begin executing commands on behalf of the attacker."

Attackers find buffer overflow vulnerabilities by searching for system calls and functions that do not restrict the length and type of input. This can be done manually or electronically with a code inspection tool. The attacker can also run a brute force attack against the program in the hope of finding vulnerabilities in the code. Once the attacker finds a vulnerability, custom code is inserted that does not crash the system, rather instructs it to execute other commands or programs of the attackers desire.

Checkpoints:
All code that accepts input from users via an HTTP request must be reviewed to ensure that it can identify large input. Once inappropriate data is identified the activity must be logged and the data dropped.

All data input fields must have reasonable field lengths and specific data types. Limit the amount of text allowed in free form fields.

Routinely check the code of web applications during their development phase to assure that the design is secured as built.[28]

Insecure Use of Cryptography

Challenge:

Apply encryption to any part of the program that affects critical or confidential data. Assure that all elements of encryption are securely stored. Encryption schemas should be developed by a commercial company rather than developed internally.

Encryption is fairly easy to add to an application, but it is often not done correctly. Some common mistakes are:

- Insecure storage of keys, certificates, and passwords
- Improper storage of secrets in memory

- Poor source of randomness
- Poor choice of algorithm or internally developed algorithms
- Failure to encrypt critical data
- Attempt to invent a new encryption algorithm
- Failure to include support for encryption key changes and maintenance procedures. [29]

Checkpoints:
Determine what data is critical or vulnerable and develop encryption schemes to sheild the data from unauthroized users and use. Encryption adds latency to the application, therefore it may be prudent to apply encryption to specific parts of the site, for example, the authentication pages.

Review the code to learn how critical data is secured. The review should also identify how keys, passwords, and other secrets are stored, loaded, processed, and cleared from memory.

Assure the developer's choice of randomness and algorithm is of high quality. The programmer should not build the algorithm or the randomness. There are numerous professional sources available for both.

Error Handling

Challenge:

Errors are inevitable. Errors can be caused by user, programs, or perhaps they are errors between two systems. During development and testing an effort is made to identify all potential errors and appropriate error messages are developed for the end user. There will also be errors that are unanticipated. The application must have protocols for these errors as well. Left "unhandled," the administrator has no idea that an error has occurred. The procedure for handling the unanticipated needs to include what the error was, when it occurred, and where it occurred.[30]

Checkpoints:

During development write a policy for handling errors. Determine which errors should trigger a response to the end user. Carefully write error pages with appropriate information. The error page reported to the end user must be carefully crafted to give the user some information. However, an attacker can learn a tremendous amount of information about a website from default error messages. The messages "file not found" or "access denied" give hackers information about the file system structure and its permissions. Determine which errors should be logged.

Log unhandled errors to an event log. Include time and date, user id, error code, if possible the code line. This log should be encrypted as it is critical information.

Thoroughly test the application to determine the possible errors. Decide what the programmatic response will be to known errors. Write error pages that reflect enough information to the end user without giving the user information about the code, the file system, or permissions.

When an error occurs that causes the program or a part of the program to fail, it is vital that the system will “fail closed,” blocking an unauthorized user from reaching the operating system or the site. The action that caused the error should be logged and then blocked.

Logging

Challenge:
Logging is crucial to an organization’s ability to track unauthorized access and to determine if any access attempt was successful. Logs provide individual accountability. They are vital to reconstruction of events leading to a program failure. Log as much as possible. Logs are often required in any legal proceedings.[31]

Checkpoints:
Begin by synchronizing your servers and syslog server to a time server. Time and date stamps must be accurate.

Preserve a baseline of your network to be used as a comparison point in the event of system failure.

The following items will make any log entry meaningful:

- Date and Time
- Initiating Process
- Process Owner
- Description

Log all Authentication and Authorization Events – logging in, logging out, failed logins. These should include date/time, success/failure, resources being authorized, and the user requesting the authorization, if appropriate an IP address or location of the Authentication Attempt.

Log all Administrator activity. All of it.

Log the deletion of any data.

Log any modification to data characteristics: permissions, location, field type.

Log files are critical data. They should be encrypted. If your environment is highly secure consider WORM technology to protect the log files from deletion or modification.

Develop a procedure for archiving log files. Consider encrypting this critical data.

Remote Administration Flaws

Challenge:
In a most secure scenario, remote administration is not allowed. As this is not often possible, it is necessary to design a secure system for remote connections to the server.

Checkpoints:
Determine how the site is to be administered. Document who has the rights to make changes, and when they can be made. Determine an effective vehicle for remote management such as a VPN solution, strong authentication with tokens, or certificates.

Assure that user roles and administration roles are clearly defined and that the program holds the roles to their intended use. You may also bind administrator functions to specific IP addresses using IP Filtering.

Web Application and Server Configuration

Challenge:
Out of the box, servers are laden with vulnerabilities. They must be patched before web services are installed. All default settings should be reviewed; and unnecessary services deleted or disabled.

Checkpoints:
Configure server disks to allow for the separation of the operating system and the web server. This will allow the restriction of directory traversal to inappropriate locations.

Verify that assigned file and directory permissions are correctly applied using “least privilege” mode.

Disable any services that are not used by the web server or applications.

Delete default accounts and their default passwords

Rename the default Administrator account or make it inaccessible. Delete all guest accounts.

Disable debugging functions.

Edit error messages to provide as little information as possible to a hacker.

Do not use self-signed SSL certificates, or default certificates. Assure that SSL certificates and encryption settings are properly configured.

Scan from the outside network to assure that all unnecessary ports are closed. Run port scans monthly to assure that nothing has been changed.

Assign security maintenance to an individual or team to be responsible for: monitoring latest security vulnerabilities; testing and applying the latest patches; updating security configuration guidelines; regular vulnerability scanning; regular status reports to upper management; and documenting the overall security practice or posture.

Conclusion

Each of the challenges discussed in this paper are a part of “Defense in Depth.” Should any one piece of this defense fail or be compromised, another layer of defense should stop an intruder from taking complete control of the site.

The checklist provides a foundation for selecting and contracting with an application developer. The corporation’s requirements for security are clearly available to the developer as part of the contract. The developer will have a concrete understanding of the risk mitigation requirements throughout the design and development process. Prior to moving into a production, environment security analysis can verify that all known security gaps are closed.

Security requirements will change in response to the external environment. What secured an application today may need to be changed tomorrow. Constant review and attention to the current security threat environment is necessary to maintain the application’s security.

Externally facing applications have provided corporations great flexibility and greater efficiencies. As evidenced by the challenges of this checklist, they can also provide a serious security risk if they attach to mission critical systems. It is imperative to secure and maintain the state of security throughout the life-cycle of the application.

Bibliography

Aspect Security, “Common Vulnerabilities for Web Applications,” http://www.aspectsecurity.com/comvuln.html, Accessed January 20, 2004.

Carnegie Mellon, Software Engineering Institute, February 3, 2000, Malicious HTML Tags Embedded in Client Requests, www.cert.org/advisories/CA-2000-02.html

Cook, S., January 11, 2003, “A Web Developer’s Guide to Cross-Site Scripting”, SANS Institute GIAC Practical Repository, Accessed, January 22, 2004.

Curphey, M, Endler, D, Hau, W, Taylor, S, Smith, T, Russell, A, McKenna, G, Parke, R, and Nigel, K. September 22, 2002. “A Guide To Building Secure Web Applications,” The Open Web Applications Security Project, http://www.cgisecurity.com/lib/OWASPBuildingSecureWebApplicationsAndWebServices-V1.1.pdf, Accessed February 18, 2004.

Janowski, D, Sarrel, M., “IPSec and SSL: The Nitty-Gritty, http://pcmag.com/print_article/0,3048,a=45204,00.asp, Accessed August 19, 2003.

Klein, Amit, Sanctum, Inc, 2002,”Hacking Web Applications Using Cookie Poisoning”, http://www.cgisecurity.com/lib/CookiePoisoningByline.pdf, Accessed February 24, 2004.

Kolsek, M. December 2002, “Session Fixation Vulnerability In Web-Based Applications”, ACROS, http://www.acros.si/papers/session_fixation.pdf, Accessed February 4, 2004.

Legary, M. July, 30, 2003, “Understand Technical Vulnerabilities: Buffer Overflow Attacks,” http://www.seccuris.com/documents/features/Securris - Understanding Technical Vulnerabilities - Buffer Overflows.pdf, Accessed February 23, 2004.

Microsoft Corporation, “Platform SDK: Security,” http://msdn.microsoft.com/library/default.asp?url=/library/en-us/security/security/s_gly.asp, Accessed February 18, 2004.

Microsoft Corporation, November 24, 2003, “How to Create Stronger Passwords,” http://www.microsoft.com/security/articles/password.asp, Accessed February 20, 2004.

Microsoft Corporation. Design Guidelines for Secure Web Applications, MSDN

Miller, C, October 20, 2002, “Password Recovery,” http://fishbowl.pastiche.org/archives/docs/PasswordRecovery.pdf, Accessed February 18, 2004.

Moore, K., October 2000, “RFC 2964 - Use of HTTP State Mechanisms”, http://www.faqs.org/rfcs/rfc2964.html, Accessed February 8, 2004

O’Gorman, Lawrence, Avaya Labs Research, Basking Ridge, NJ, “Securing Business’s Front Door—Password, Token, and Biometric Authentication, http://www.research.avayalabs.com/user/logorman/HA_BusinessChapter.doc, Accessed February 23, 2004.

Obviex.com, January 14, 2004, “How to: Hash with Salt”, http://www.obviex.com/samples/hash/aspx

O'Gorman , L, Securing Business’s Front Door – Password, Token, and Biometric Authentication, Avaya Labs Research, Basking Ridge, NJ, http://www.research.avayalabs.com/user/logorman/HA_BusinessChapter.doc, Access, January 23, 2004.

Open Web Application Security Project (OWASP), January 27, 2004, The Ten Most Critical Web Application Security Vulnerabilities – 2004 Update, http://prdownloads.sourceforge.net/owasp/OWASPTopTen2004.pdf?download, accessed February 24, 2004.

Open Web Applications Security Project (OWASP), January 11, 2003, The Ten Most Critical Web Application Security Vulnerabilities, no longer available on line.

RSA Security, "RSA SecurID® Two-Factor Authentication," http://www.rsasecurity.com/products/securid, Accessed February 18, 2004.

Spett, Kevin, 2002, "Blind SQL Injection: Are Your Web Applications Vulnerable?", http://www.spidynamics.com/whitepapers.html

Spett, Kevin, 2002, "SQL Injection: Are Your Web Applications Vulnerable?", http://www.spidynamics.com/whitepapers.html

SPI Dynamics, 2002, "SQL injection, are your web applications vulnerable", http://www.spidynamics.com/whitepapers/Blind_SQLInjection.pdf, Accessed February 11, 2004.

Tuliper, Adam, January 2, 2003, "Web Application Error Handling in ASP.NET," http://www.15seconds.com/issue/030102.htm, Accessed February 23, 2004.

MultiNet, Inc, (2001),Web Vulnerabilities and Security Solutions, http://elitesecureweb.com/dta/solutions/wvuln1.html, Accessed February 11, 2004

[1] Curphey, Mark, Endler, David, Hau, William, Taylor, Steve, Smith, Tim, Russell, Alex, McKenna, Gene, Parke, Richard, and Nigel, Kevin, (September 22, 2002). "A Guide To Building Secure Web Applications," The Open Web Applications Security Project, http://www.cgisecurity.com/owasp/OWASPBuildingSecureWebApplicationsAndWebServices-V1.1.pdf _page 15, February 18, 2004.

[2] Curphey, M. et.al. page 30.

[3] Microsoft Corporation, (November 24, 2003), "How to Create Stronger Passwords," http://www.microsoft.com/security/articles/password.asp, February 20, 2004.

[4] RSA Security, "RSA SecurID® Two-Factor Authentication," http://www.rsasecurity.com/products/securid, February 18, 2004.

[5] Miller, Charles, "Password Recovery," (October 20, 2002), http://fishbowl.pastiche.org/archives/docs/PasswordRecovery.pdf, February 18, 2004.

[6] Microsoft Corporation, "Platform SDK: Security," http://msdn.microsoft.com/library/default.asp?url=/library/en-us/security/security/s_gly.asp, (February 18, 2004)

[7] Curphey, M., et. al., p.30.

[8] Curphey, M. page 49\.

[9] The Open Web Application Security Project (OWASP), (January 11, 2003 Since superceded by January 27,2004), "The Top Ten Most Critical Web Application Security Vulnerabilities – 2003 and 2004 Update, February 24, 2004, http://prdownloads.sourceforge.net/owasp/OWASPTopTen2004.pdf?download, February 25, 2004.

[10] Aspect Security, "Common Vulnerabilities for Web Applications," http://www.aspectsecurity.com/comvuln.html, February 18, 2004.

[11] Kolsek, M. (December 2002), Session Fixation Vulnerability In Web-Based Applications, ACROS, http://www.acros.si/papers/session_fixation.pdf, February 4, 2004.

[12] Moore, K., (October 2000), "RFC 2964 - Use of HTTP State Mechanisms", http://www.faqs.org/rfcs/rfc2964.html, February 8, 2004.

[13]Moore, K. section 2.1.

[14] O'Gorman, L., Avaya Labs Research, Basking Ridge, NJ, "Securing Business's Front Door – Password, Token, and Biometric Authentication", http://www.research.avayalabs.com/user/logorman/HA_BusinessChapter.doc, page 7, February 24, 2004

[15] Kolsek, p.11

[16] OWASP 2003, page 9.

[17] CERT Coordination Center, DoD-CERT, the DoD Joint Task Force for Computer Network Defense (JTF-CND), the Federal Computer Incident Response Capability (FedCIRC), and the National Infrastructure Protection Center (NIPC), (February 3, 2000), “Malicious HTML Tags Embedded in Client Requests”, http://www.cert.org/advisories/CA-2000-02.html, p.2.

[18] Meier, J. D., Mackman, Alex, Dunner, Michael, Vasireddy, Srinath, Escamilla, Ray, and Murukan, Anandha (MSDN), “Chapter 4: Design Guidelines for Secure Web Applications”, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh04.asp, p.75-77, February 24, 2004.

[19] Meier, J.D., et. al., page 77

[20] Meier, J.D., et. al., page 76

[21] Cook, Stephen, (January 11, 2003), “A Web Developer’s Guide to Cross-Site Scripting”, http://www.giac.org/practical/GSEC/Steve_Cook_GSEC.pdf , February 24, 2004.

[22] OWASP, 2003, page 11.

[23] CGISecurity.com, (August, 2003),Cross Site Scripting Questions and Answers”, http://www.cgisecurity.com/articles/xss-faq.shtml, February 24, 2004.

[24] Spett, Kevin, SPI Dynamics, (2002), “SQL Injection, Are Your Web Applications Vulnerable?”, http://www.spidynamics.com/whitepapers.html, February 24, 2004.

[25] Spett, Kevin, SPI Dynamics, (2002), “Blind SQL Injection: Are Your Web Applications Vulnerable?”, http://www.spidynamics.com/whitepapers.html, February 24, 2004.

[26] OWASP, 2003, page 13.

[27] MultiNet, Inc, (2001), Web Vulnerabilities and Security Solutions, http://elitesecureweb.com/dta/solutions/wvuln1.html, February 11, 2004

[28] Legary, Michael, (July, 30, 2003), “Understand Technical Vulnerabilities: Buffer Overflow Attacks,” http://www.seccuris.com/documents/features/Seccuris-Understanding%20Technical%20Vulnerabilities%20-%20Buffer%20Overflow.pdf, February 23, 2004.

[29] OWASP, 2003, page 17.

[30] Tuliper, Adam, (January 2, 2003), “Web Application Error Handling in ASP.NET,” http://www.15seconds.com/issue/030102.htm, February 23, 2004.

[31] Curphey, M. page 53.

Last Updated: September 17th, 2020

Upcoming SANS Training

Click here to view a list of all SANS Courses

SANS Australia Spring 2020	, AU	Sep 21, 2020 - Oct 03, 2020	Live Event
SANS FOR500 Milan 2020 (In Italian)	Milan, IT	Oct 05, 2020 - Oct 10, 2020	Live Event
SANS October Singapore 2020	Singapore, SG	Oct 12, 2020 - Oct 24, 2020	Live Event
SANS SEC504 Rennes 2020 (In French)	Rennes, FR	Oct 19, 2020 - Oct 24, 2020	Live Event
SANS SEC560 Lille 2020 (In French)	Lille, FR	Oct 26, 2020 - Oct 31, 2020	Live Event
SANS Tel Aviv November 2020	Tel Aviv, IL	Nov 01, 2020 - Nov 05, 2020	Live Event
SANS Sydney 2020	Sydney, AU	Nov 02, 2020 - Nov 14, 2020	Live Event
SANS Secure Thailand	Bangkok, TH	Nov 09, 2020 - Nov 14, 2020	Live Event
APAC ICS Summit & Training 2020	Singapore, SG	Nov 13, 2020 - Nov 21, 2020	Live Event
SANS FOR508 Rome 2020 (in Italian)	Rome, IT	Nov 16, 2020 - Nov 21, 2020	Live Event
SANS Oslo Local November 2020	Oslo, NO	Nov 23, 2020 - Nov 28, 2020	Live Event
SANS Wellington 2020	Wellington, NZ	Nov 30, 2020 - Dec 12, 2020	Live Event
SANS OnDemand	OnlineUS	Anytime	Self Paced
SANS SelfStudy	Books & MP3s OnlyUS	Anytime	Self Paced