

Web Application Checklist

Prepared by Krishni Naidu

References:

Web application and database security, Darrel E. Landrum, April 2001
Java's evolving security model: beyond the sandbox for better assurance or a murkier brew? Matthew J. Herholtz, March 2001
Basics of CGI security: Common Gateway Interface, CGI, at a glance, Jeffrey McKay, April 2001
CERT: Understanding malicious content mitigation for web developers
Secure a web application, Java style, Michael Cymerman
CERT: Malicious HTML tags embedded in client web requests
Best practices for web development, Razvan Peteanu
Security Code guidelines (<http://java.sun.com/security/seccodeguide.html>)
Web application security considerations
(<http://www.4.ibm.com/software/webservers/appserver/doc/v35>)
Perl Security: (<http://www.perl.com/pub/doc/manual/html/pod/perlsec.html>)
Extensible Security architecture for Java, Dan Wallach, Dirk Balfanz, Drew Dean, Edward Felten

Introduction:

This checklist is to be used to audit a web application. It is essential that the web application not be evaluated on its own in an e-commerce implementation. The other elements like the operating system, IIS/Apache, the database, router configuration and firewall configuration needs to be evaluated to ensure that appropriate steps have been taken to address the risks posed by numerous vulnerabilities that may be present.

The procedural elements must also not be forgotten e.g. physical security and the enforcement of the security policy elements such as development standards.

Prior to using this checklist the following should be considered:

- Privacy: It is not possible to highlight all the considerations to be taken into account when taking into account privacy regulations in the application due to the numerous country laws that would have to be considered. It is thus the responsibility of the auditor to identify the relevant law that is applicable in the specific country where the review is occurring and ensure that the application takes this into account.
- Web application as part of ERP package: In some instances the web application may be an add on module of an ERP e.g. SAP, Navision, etc. In such instances it may be important to ascertain the security implications with the requisite vendor as well as with the in house development team to ascertain the security implications of the modification. The web application authentication may be a part of the ERP thus it is important to perform the review together with the security review of the ERP. In such instances it is also important to ensure that no web user has ERP administrator access e.g. SAP ALL access in a SAP environment. ERP security reviews are a comprehensive subject on their own and thus no attempt has been made in this checklist to audit the web application part of a ERP. This checklist with some modification can be used in conjunction with a security review of the ERP.

- Database and other elements security; This checklist does not include database security or security considerations for any of the other elements like the operating system as these are exhaustive topics that need their own checklists.
- Programming languages: It is beyond the scope of this paper to provide a comprehensive listing of all security considerations for all languages used to create web applications. A listing of security considerations for languages that are commonly used is however provided like Java, Perl and CGI. Other languages not catered for are XML, ActiveX, etc.
- Securing the program/web application: This checklist does not address the aspect of securing the program files at the operating system folder/directory level. This is an important element to ensure that only the authorised developer has access to this directory in the development environment. When the program is moved to the production environment, there should be adequate controls such as movement by a change control group and the program files must again be secured such that they may not be modified by unauthorised persons.

Prior to reviewing the security of the application it is important to ascertain what the application will be doing e.g. will the application be used to provide clients access to account information or will the application be used to sell consumer goods via the Internet.

Checklist:

No.	Control Item
1.	Requirements phase Ensure that the user requirements specifications include the following items: <ul style="list-style-type: none"> • Whether the assets have been identified • How the application will be used • Identifying the users, their roles and rights – authorization and authentication • Legal and business issues – support for non-repudiation, audit trail, digital signatures, strong encryption
2.	Java Ensure that the Java application is sandboxed. Ensure that the java.security, java.security.acl and java.security.interfaces packages are run. Run the application with the following: <ul style="list-style-type: none"> • java -Djava.security.debug=help This will output the results of Checkpermission calls, loading and granting policies, dumps of relevant domains and other information. Review the output to ensure that it is appropriate in terms of security. Ensure that the SecureRandom class is used to create random numbers. Ascertain if form based authentication or basic authentication is being used. If it is form based authentication, ensure that sensitive forms are protected via userids and passwords. Since userids and passwords are passed in the clear ensure that SSL is used. Ensure that the code includes a line that does not permit access if the authentication fails and returns the user to the login form again e.g.: <pre data-bbox="402 1751 885 1850"> // if the user is not authenticated if (!isAuthenticated .booleanValue()) { </pre>

```
// process the unauthenticated request
unauthenticatedUser (response, requested Page) ;
}
```

Ensure that code exists to store the user authentication information inside a session variable e.g.

```
// create a session
session + request.getSession ( True);

// convert the Boolean to a Boolean
Boolean booleanIsAuthenticated + new Boolean( isAuthenticated) ;

// store the Boolean value to the session
session.putValue(
    Constants.AUTHENTICATION,
    BooleanIsAuthenticated) ;
```

Ensure that the classes available to the virtual machine are limited. Review the classpath to ensure that unnecessary entries are removed.

Ensure that the code does not have access to third party tools or extraneous code.

Review the various sensitive beans to ensure that the EJB's deployment descriptor has the following code:

```
(accessControlEntries
    DEFAULT [administrators basicUsers
    TheRestrictedMethod [administrators]
) ; end accessControlEntries
```

This ensures that only the administrators have access to the restricted method. Review the weblogic.properties file to ensure that only authorised administrators are listed in the administrators group.

Ensure that the java.security.acl package is used to grant permissions and add new users.

Review all user permissions using the following:

```
Boolean isReadFileAuthorized = accessList.checkPermission (*User,
    ReadFile) ;
```

Ensure that user permissions are appropriate e.g. only the designer has access to the owner object.

Ensure that the security manager has been enabled.

Ensure that non final public static variables are not used since there is no way to check whether the code that changes such variables has the appropriate permissions.

Ensure that the scope of methods and fields are reduced as much as possible.

Ensure that developers have refrained from using public methods/fields.

Ensure that any public method that has access to and/ modifies sensitive states includes a security check.

Ensure that adequate steps have been taken to prevent against package insertion e.g.:

- add line to java.security properties file
package.definition=Package#1 [, Package#2,....., Package#n]
- Place the package's class in a sealed JAR file

Ensure that the following line has been added to the java.security properties file to protect package accesses:

```
Package.access=Package#1 [,Package#2,.....,Package#n]
```

Ensure that objects are made immutable.

Ensure that there is no return of a reference to an internal array that contains sensitive data.

	<p>Ensure that user given array of objects is not stored directly. If serialisation is used ensure the following precautions are taken:</p> <ul style="list-style-type: none"> • Ensure that the transient keyword is used for fields that contain direct handles to system resources and that contain information relative to an address space. • Ensure that a class defines its own deserialising method and that the ObjectInputValidation interface is used to validate invariants. • If a class defines its own serialising method, ensure that it does not pass an internal array to an DataInput/DataOutput method that takes an array. • Ensure that byte streams are encrypted. • If untrusted code has a restriction in creating an object, ensure that the untrusted code has the same restriction when it deserialises the object. <p>Ensure that native methods are examined for the following:</p> <ul style="list-style-type: none"> • What they return • What they take as parameters • Whether they bypass security checks • Whether they are public or private • Whether they contain method calls which bypass package boundaries, thus bypassing package protection <p>Ensure that sensitive information such as credentials is kept in mutable data types. Ensure message digests or digital signatures are used to protect the integrity of sensitive data.</p>
3.	<p>Privileged code Ensure that privileged code is as short as possible. Privileged code when run can access any resource within the code that it does not have permissions to access. Ensure that tainted variables are not used within the privileged code when used with public methods. Ensure that private methods are used and can not be called from outside the class. Ensure code is wrapped in a privileged block when the code performs tasks that would not normally be allowed by an applet or untrusted code.</p>
4.	<p>Perl Ensure that the script is run in tainted mode. This is done by using the <code>-T</code> command line flag. Run the following script to check whether a variable contains tainted data:</p> <pre> Sub is_tainted { Return ! eval { Join (' ', @_), kill 0; }; } </pre> <p>Any presence of tainted data anywhere within an expression renders the whole expression tainted. If data has to be untainted ensure that the <code>./+ </code> command is not used as this command lets everything through. Ensure that for “Insecure \$ENV{PATH}” messages, the <code>\$ENV{‘PATH’}</code> is set to a known value. Each directory in the path must be only writable to the owner and the group. Ensure that the variables such as IFS, CDPATH, ENV and BASH_ENV are deleted since these are run untainted. The script is as follows:</p> <pre> Delete @ENV{qw(IFS CDPATH ENV BASH_ENV)}; # Make %ENV </pre>

	<p style="text-align: center;">safer</p> <p>Ensure that file tests for taintedness are performed for user supplied filenames. Ensure that a child has been forked using the open syntax that connects the parent and the child via a pipe. The child has less privilege compared to the parent and thus is safer to use. It is thus safer to open or pipe a file from setuid/setgid.</p> <p>Since backticks are also vulnerable to call the shell, ensure that the shell is never called. The script to ensure that backticks are performed safely are as follows:</p> <pre> Use English; Die "Can't fork: \$!" unless defined \$pid = open (KID, "- "); If (\$pid) { #parent While (<KID>) { # do something } close KID; } else { my @temp = (\$EUID, \$EGID); \$EUID = \$UID; \$EGID = \$GID; # initgroups () also called; #Make sure privs are really gone (\$EUID, \$EGID) = @temp die "Can't drop privileges" unless \$UID == \$EUID && \$GID eq \$EGID; \$env{PATH} = "/bin:/usr/bin"; exec 'myprog', 'arg1', 'arg2' or die "can't exec myprog: \$!"; } </pre> <p>Ensure that a similar strategy as above is used for glob.</p>
5.	<p>Cgi</p> <p>Ascertain how often CGI hacking tools are run to determine vulnerabilities and whether there is a process to fix the vulnerabilities identified by the hacking tools. Tools such as Whisker (Rain Forrest Puppy) or wCGIchk can be used to ascertain vulnerabilities.</p> <p>Ensure that there is a process to keep up to date with new vulnerabilities/patches and updates and fix them as appropriate.</p> <p>If the CGI programs are used to create or open files, ensure that the following is observed:</p> <ul style="list-style-type: none"> • Error handling code is included to warn if the file isn't actually a file, cannot be created or opened, already exists, requires different permissions, etc. • Ensure that files are not written to world writeable or world readable directories. • Ensure that the files UMASK are explicitly set. • Ensure that the file permissions are set as restrictively as possible. • Ensure that the file's name does not have metacharacters in it. If the file is created on the fly ensure that there is a screening process to filter out metacharacters. • Ensure that scripts not in use are deleted. • Ensure that CGIWrap is utilized to allow general users access to CGI scripts and HTML forms without compromising the security of the web server. • Ensure that scripts are run using the permission of the user who

	<p>owns the script and not the userid of the httpd process.</p> <p>If CGI scripts are downloaded from the web ensure that the following checks are made:</p> <ul style="list-style-type: none"> • Complexity of the script – more problems if more complex. • Whether it reads or writes files on the host system. Programs with read files may violate access restrictions or pass sensitive information to hackers. Programs that write files may modify or damage documents or introduce Trojans. • Interactions with other programs on the system e.g. with sendmail. Is the interaction secure • Whether it runs with suid privileges. This should not be permitted. • Whether the author validates user input from forms. This is an indication that security is being considered. • Whether explicit path names are used when invoking external programs. The PATH environment variable is insecure if used to resolve partial path names. <p>If coding in C ensure that the developer has taken into account buffer overflows.</p> <p>Ensure that unchecked remote user input is not passed to a shell command. Risky C commands are the popen(), exec(), and in Perl the system(), exec(), piped open and eval() function.</p> <p>Ensure that backtick quotes are avoided.</p>
6.	<p>Malicious HTML tags embedded in client web requests</p> <p>Ensure that there is a process whereby web developers ensure that dynamically generated pages do not contain undesired tags.</p> <p>Ensure there is a process for developers to restrict variables to those characters that are explicitly allowed and to check those variables during the generation of the output page.</p> <p>Ensure that the developers follow the two processes above.</p>
7.	<p>Malicious content mitigation</p> <p>Ensure that the character set encoding for each page generated by the web server is explicitly set.</p> <p>Ensure that the developers have a defined process to identify special characters and filter them out. List of special characters are as follows:</p> <ul style="list-style-type: none"> • < • > • & • “ ” • “ • space and tab • new line • % • semicolon, parenthesis, curly braces • ! • ampersand' <p>Ensure that dynamic output elements are encoded.</p> <p>Ensure that dynamic content filtering is implemented on the output side.</p> <p>Ensure that there is a process to carefully examine cookies that are accepted and that filtering techniques are used to verify that they are not storing malicious content.</p> <p>Ensure that encoding is also applied to URL's and HTML pages.</p>

8.	<p>Testing of Application</p> <p>Ensure that the application is tested using application scanners like AppScan from Sanctum, Retina from eEye, and Web Inspect from SPI Dynamics.</p>
9.	<p>Privacy</p> <p>Ensure that the application deals with the application and handling of private data as defined by the country's specific laws and regulations.</p>
10.	<p>Ensure that there is a process whereby application developers are keeping abreast of new vulnerabilities by for e.g. subscribing to mailing lists like CERT.</p>
11.	<p>Documentation</p> <p>Ensure that the system is adequately documented. The documentation should include:</p> <ul style="list-style-type: none"> • server and application settings • resource permissions • what the sensitive resources are • how to perform operations or changes the right way
12.	<p>Anonymous access</p> <p>Ensure that all pieces of functionality use proper authentication rather than anonymous authentication.</p>
13.	<p>Testing</p> <p>Ensure at the testing stage there is adequate testing of the authentication and ACL's.</p>
14.	<p>Application logins</p> <p>Ensure that the code is not run using the suid root/administrator account. Also the application must not be run using the database administrator account e.g. SQL sa account.</p>
15.	<p>GET, POST & Encryption</p> <p>Ensure that GET is not used to send sensitive data as the information is logged in clear text even if SSL is used. SSL only encrypts data in transit – not at the destination point.</p> <p>If POST is used the HTTP body is not logged. However the POST method still sends data in clear text, thus encryption is vital.</p> <p>Ensure that encryption is used for sensitive data at the application level.</p>
16.	<p>Incoming data</p> <p>Ensure that the developer has fully considered the implications of incoming data in terms of the URL, method, cookie, HTTP Headers and data fields.</p> <p>Ensure such scenarios have been appropriately tested in terms of the following:</p> <ul style="list-style-type: none"> • if the URL is changed can the client access another users session <p>Ensure that the application has been tested to ensure that incoming data fields can not overflow buffers or append to an SQL statement (execute code on the SQL server).</p>
17.	<p>Client keeping important data</p> <p>Ensure that the application does not rely on the client keeping information such as :</p> <ul style="list-style-type: none"> • Hidden form fields • Parameters <p>Ensure that any information, which is capable of being changed by the client, is stored on the server side.</p>
18.	<p>Logs</p> <p>Ensure that logs are created and that the information provided by the logs are useful i.e. provide sufficient detail.</p>

19.	<p>ASP/JSP</p> <p>Ensure that sensitive credential information such as username/password combinations for accessing the following:</p> <ul style="list-style-type: none"> • membership directories • database connection strings <p>are not hardcoded in the page.</p>
20.	<p>Extensions</p> <p>Ensure that file extensions are not available on the server side. If the hacker asks for the file specifically instead of the including page, will be served possible sensitive information.</p>
21.	<p>HTML Comments left in production code</p> <p>Ensure that no sensitive information is included in the HTML comments, which are embedded in the HTML or client script. E.g.</p> <ul style="list-style-type: none"> • Connection string that was once part of a server side script and commented out. Through editing this can reach the client script and thus be transmitted to the browser.
22.	<p>Error Messages</p> <p>Ensure that error messages do not reveal sensitive information, which can be used to facilitate an attack against the organisation. For e.g.:</p> <ul style="list-style-type: none"> • physical paths • platform architecture <p>Review the error related configuration of the server and how errors are handled by the application. Under IIS, ensure that the generic error option is chosen instead of send detailed ASP error message to client (default).</p>
23.	<p>Relationship with QA and code reviews</p> <p>Ensure that there is close continuous working relationship between QA and the development team, such that any security related issues discovered by QA are forwarded timeously to the development team to fix.</p> <p>Ensure that code reviews are performed and that issues raised during the code review are adequately fixed.</p>
24.	<p>Wizard generated or sample code</p> <p>Ensure that there is a process to review wizard generated or sample code to ascertain whether they include hardcoded credentials to access resources e.g. databases.</p> <p>Alternatively review wizard generated or sample code to ensure that there is no hardcoded credentials.</p>
25.	<p>C/C++</p> <p>Since C/C++ does not deal with buffer overflows, the programmer is left to implement this. Another problem is the format string attacks. Ensure that there are proper code reviews to identify insecure practices and that the issues raised have been fixed.</p> <p>Test for unsafe constructs using tools like L0pht's SLINT.</p> <p>Ensure that all input arguments are checked for validity.</p> <p>Ensure that the system() call, shell(), popen and exec*p is not used.</p> <p>Ensure that scanf is not used to read anything as its behaviour when given a string that does not match the format expected is undefined.</p> <p>Ensure that environment variables are actively checked for validity.</p> <p>Ensure that all functions are checked for valid returns.</p> <p>Ensure that binaries are stripped.</p>
26.	<p>SSL</p> <p>Ensure SSL is used to provide encryption for in-transit elements.</p>