Web Application Security - Layers of Protection

by
William Fredholm
January 26, 2003

GIAC Security Essentials Certification (GSEC)
Practical Assignment – Version 1.4b
Option 1 – Research on Topics in Information Security

Abstract

This paper reviews some of the large number of resources available for creating secure Web applications. These resources ranges from the security features of the development and database environments used, to automated tools evaluating an existing Web application, to Web sites dedicated to all facets of Web application security.

Web application security is an extremely complex topic and by making one single mistake an otherwise secure application may be opened up to uninvited guests. By using the different resources available the risk borne by applications can be reduced to an acceptable level. In addition, some risk can be avoided at the very beginning of the project life cycle when the requirements for the system are defined.

Resources on the Internet - OWASP

The Open Web Application Security Project (OWASP) is a Web site devoted to Web application security [19]. According to the *About Us* section, the OWASP site strives to be a location to learn and share information about Web application security. The site has several different sub sections.

Web application security now also has its own top-10 list. The OWASP *The Ten Most Critical Web Application Security Vulnerabilities* paper [12] details not only a name and a short description of 10 common vulnerabilities, but it also provides more extensive explanations on what the vulnerability is, what type of systems that are affected, how to test for it and remedies. Also included are references to other material in relation to each vulnerability.

A second document published at the site is titled *A Guide to Building Secure Web Applications* [4]. This is a longer document explores topics ranging from the definition of a Web application to descriptions and security implications of the many technical elements involved in Web applications and the HTTP protocol.

One section of the site is focused on testing. While no compete document has been published on the site at this point, a draft table of contents indicates that a comprehensive document is in the works.

The project is publishing a Web application vulnerability scanner named Web Scarab. The tool is described as a spider that will crawl a site to find potential vulnerabilities and generate tests to evaluate the potential vulnerabilities. The tool can also work as a proxy in manual testing mode. A "pre-beta 3" version of the tool was released in December of 2002, and the source code for the tool can be downloaded from SourceForge.net [20].

WebGoat is an application intended to be used as a learning tool for Web application security concepts. WebGoat includes lessons for vulnerabilities and it allows a user to try to exploit the vulnerabilities after reading about them. The OWASP site also states that future releases will include a benchmarking feature for Web application vulnerability scanners. WebGoat can be downloaded from SourceForge.net [20].

Other projects at different stages of completion includes:

> *VulnXML*, a project to standardize the descriptions of Web application vulnerabilities so that checks can be developed in any Web application vulnerability scanner tool that understands this common format.
> *Application Security Attack Components* (ASAC), a project focusing on creating a common set of definitions that can be used to describe vulnerabilities. This would remove ambiguity when describing and modeling more complex vulnerabilities.
> *Filters*, which is a project to assist Web developers in using secure practices.

CodeSeeker, a firewall and/or IDS at the application level. Using rules, it examines HTTP requests to block and/or report on malicious traffic. *CodeSeeker* is installed on the Web server and examines traffic after any SSL traffic has been decrypted,

As seen above, the OWASP site has numerous projects in different realms of Web application security. Many of the projects are still in development stages, but a lot of information is provided already from the two papers published at the site.

Security Provided by Development Environments

The development environment in use provides many different security features. One problem for developers is that different environments provide different features. The security features are not automatic, if they are not properly used and coded into the application they will not provide any protection. What security features in the development environment do provide is abstraction, the developer does not need to implement all security features from scratch and the implementation details are hidden. Below are highlights of some security features provided by the Java and .NET Web development environments and that are commonly required by any Web application.

In Java, several security features are available for the developer for example functions for cryptography and access control (permissions). Java (JDK 1.2) defines a security model where Java code is subject to a security policy that dictates what permissions a caller executing some code has on different system resources [5].

Java Web Services also has many built in security features [3]. Security can be defined as an application is deployed (called declarative security) or more granular in the Java program (called programmatic security).

The access control mechanism includes several different concepts. Users are individuals that have been authenticated by the run time environment. Groups are sets of users, and Roles (security roles) are permissions to access some resource. Further, security constraints are specified to protect parts of the Web application.

The .NET environment also includes many security features [9] starting with IIS (the Web server used in the .NET environment) using regular Windows file permissions. IIS must have the proper permissions to access the files a user requests via their browser.

Access is either granted based on specific login credentials provided by the user, but will more often for Web applications be based on the access right of a standard account used by IIS for all anonymous access (access without specific user Windows credentials). Because of the need for anonymous access (otherwise, separate Widows accounts has to be created for each user) the usefulness of the built in Windows file permissions will be limited for Web applications.

.NET provides other security features to secure a Web application. Access can be granted or denied for URLs based on the user, role or verb (type of access being requested, GET or POST). This type of authorization can be defined for the complete application or on individual sub directories and it is set up using configuration files. This is still a rather coarse method, as it defines access based on the applications directory structure and not based on the data in the application (i.e. if you can access the "ViewBankAccount.aspx" page, you can view any bank account). Role based security functions allow a .NET application fine granular access control based on the user and the user's role. This type of access control must be coded into the application by the developer. .NET also provides a set of encryption routines.

Both the Java and the .NET environments provide a rich set of comparable security features [6]. The security features are extensive and flexible but complex. It would be rather easy to make mistakes, either locking out sections of the application that should be accessible or opening up parts that should be locked down. The first case will soon be noted as users start to point this out (hopefully before production), while the second case may not be discovered at all until it is exploited by a malicious user.

Security Provided by the DBMS

The database system itself can provide a high level of security, if used properly. The DBMS can for example provide granular access control for and auditing, but the DBMS must also be maintained with patches and audit logs must be reviewed in order for the information to be of use [18].

While the DBMS can limit access to data in several different ways on a user-by-user basis, one common problem in a Web application is that the end user does not actually access the database using his or her own account. Instead, the Web application often uses a service or functional account that all database access is conducted through. The Web application acts as a proxy between the user and the DBMS. In this case, many of the more granular access control features of the DBMS are bypassed and the access control now becomes the burden of the Web application.

The worst-case scenario is when a Web application uses a high power account, such as 'sa' for a SQL Server system. By default, 'sa' can perform basically any function within SQL Server and the account can access any data. This in turn means that if the Web application has a vulnerability, maybe a SQL injection problem, the potential damage is enormous. Anything from data being stolen and tables being dropped to compromise of the DBMS host computer.

Another interesting scenario is when an older legacy application is "Web enabled" by deploying a Web application front-end to the system. In this case, a system that was perfectly secure when accessed via terminal or custom client software may be compromised because the Web application now uses an account that has access to more of the data than the individual user originally had. If the Web application is not secure, the legacy application is no longer secure.

Using an account like 'sa' is obviously not a good idea. Instead one or more specific accounts should be created for the Web application to use in connecting to the DBMS. Then, follow the recommendations in limiting access control only to data that is required by the accounts [18]. In this case all data and each stored procedure in use by the Web application must be available to the account or accounts, but nothing else. The setup of this scenario is a lot more complex compared to just allowing the Web application to access anything, but the level of risk is also reduced substantially.

Tools to Test Web Application Security

One method to ensure that a Web application is safe is to test it from a Web browser's perspective. Testing for functionality is usually a standard phase of the development life cycle, but testing can also be focused on security issues. This testing is conducted by sending HTTP requests from a client to the application and observing the responses.

Tools are now available to assist in this type of testing. A recent article in the Information Security magazine [22] evaluated two of these tools, AppScan by Sanctum [1] and WebInspect by SPI Dynamics [21]. The conclusion of the evaluation seems to be that the tools pinpoint many types of vulnerabilities, but that there is still room for improvement with both performance, false positives and false negatives.

In general, Web application vulnerability scanner works in two main stages. First, the tool crawls the Web site (automatic and/or manual) following URLs and submitting forms. The second stage is the assessment stage, where the tool generates a large number of specially crafted HTTP requests. The requests are either based on manipulating information gathered during the crawl stage (testing for issues like cross site scripting and SQL injection) or they are standard requests known to exploit vulnerabilities in Web server software (such as appending a special character at the end of a URL forcing the Web server to return source code instead of HTML). The first type of assessment requests is usually referred to as testing for "unknown vulnerabilities" while the second type is referred to as testing for "known vulnerabilities".

Tools can be an excellent companion to the experienced auditor. However, in the wrong hands, results from any type of testing tool can also give a false sense of security. In addition to running the tool, a good evaluation must include the use of a pair of critical eyes. What does it mean that the report from the tool came back "clean"? Does it mean we are secure or does it mean that the tool was not used properly?

Some issues may never be completely identified by a tool and total automation in this area may never be obtained, unless the tools can start making use of additional business related information. Consider the following example:

- A user provides a login ID and password to log on to a Web application. The users login session is maintained using cookies while it is the initial login ID that identifies what resources the user can access. Perhaps a list of specific bank accounts.
- The application includes the bank account number currently being accessed on the Web page as a parameter in some way (a hidden form filed, a URL parameter, a cookie etc.).
- In this case, if the user changes the bank account number by manipulating the parameter, the application displays the information for that bank account, no matter if it is one of the accounts on the user's list or not.
- The application is not verifying that the user should have access to the selected bank account.

This brief example does not reveal anything new when it comes to Web application security (trivial parameter manipulation), however the question here is *how would an automated tool assessing the application strictly from a Web browsers perspective ever be able to tell that the second account is not owned by the user?*

The information sent back to the browser (or the tool testing the application) looks just like perfect HTML code. No errors were produced and nothing unusual was returned. The problem here is that the application violates a business rule. As long as the tool does not know about business rules, it cannot determine that this was an error.

There are of course many options that can be suggested to allow the tool to actually determine the problem. For example, if the owner of the account is in some way included as a hidden field on the Web page the test tool could verify that the owner field does not change, but this again requires some knowledge about business rules. If the tool always flags a change in the owner field as a vulnerability, but a user does have valid access to the account (perhaps a shared account owed by the users spouse) the tool would produce a false positive. This extends to any other field, when is a change in a filed an error and when is it part of a well functioning application?

Testing in Production

Although conducting testing in production is not usually a good idea, when it comes to assessing Web application security it may be a necessity for at least a couple of reasons. First, even if thorough testing is done on development and test systems, there is still a potential for production systems to be configured differently. The actual application code could also be different, for example if files that were supposed to be deployed from test to production never got deployed, or of files has been manipulated on the production servers. Second, there are many Web applications that are already in production but that were never tested for security to begin with. There may be an option to set up a parallel test environment for such applications but that is not always possible.

Special care must obviously be taken when testing production applications as applications are many times not designed to be tested in a live production environment. This means for example that data submitted to a production database may not easily be identified as test data and cleanup may be difficult. If the application contains vulnerabilities, testing may inadvertently ruin large quantities of data that the test user should not have had access to. Testing should, if possible be conducted during maintenance windows so that any problems or DoS conditions can be cleaned up before the application goes live again.

Testing of a production application should be done in a controlled fashion, so using automated features of testing tools must be carefully considered. It would be best to know exactly what HTTP requests the testing tool will use in the assessment stage. One option for a production application testing is a 3rd party penetration type testing.

Application Level Firewalls

Once the application is up and running, there are further measures one can take to monitor how well the application is doing from a security stand point or to prevent malicious requests from ever reaching our application [2].

This type of devices or software often labeled *application level firewalls* promises to monitor HTTP traffic and automatically make determinations on what constitutes a valid request and what is to be considered a malicious request. Responses include denying a users request, logging and sending administrative alerts [10][8][7][11]. One must take care when using automated responses (denying access) as the firewall may have been responding to a false positive. Another issue with automated responses is if malicious users are able to trick the firewall in denying legitimate traffic.

At this time, searches on the Internet did not provide any major evaluations of the effectiveness of this type of products. What level of protection do they provide? What type of attack do they stop? Do they live up to their promises? While evaluations are scares, an overview of products [17] shows that there are tradeoffs that have to be made when making the decision on what to deploy.

Using Requirements to Avoid Risk

New vulnerabilities and exploits are discovered continuously, therefore, it is impossible to be 100% certain that all weaknesses in a system has been addressed. As is often pointed out within the security discipline, what we need is defense in depth [13] and each layer added must mean a stronger defense.

Because of this, security cannot be considered simply a technical issue that it is up to the development teams to "code" into the application. Nor can security be completely the responsibility of code reviewers, testers and security auditors.

Security is the responsibility of all parties involved in the application project. While the previous sections have all discussed layers of protection provided by several of these parties, this final layer is actually provided by stakeholders involved from the beginning of a development project, when the requirements for the system are defined.

The method is simple:

> *When defining requirements for a system, always consider the implications of the data within the system ending up in the wrong hands. If some data element identified carries a high risk, consider ways to do without it. If the business and the system do not absolutely need some specific data element, do not include it!*

Risk can be addressed in 4 different ways: avoid risk; transfer risk; mitigate risk or accept risk [16]. While most other methods that address Web application security intends to mitigate risk, this method instead avoids risk all together. No matter what, *unauthorized access to data that is not present will never happen*.

The cost of data being exposed to un-authorized persons is getting higher every day. For example, states are passing laws *requiring* companies to inform customers of such events [15]. In this case, the bill [14] addresses un-encrypted exposure of a persons name in combination with some specific other data elements. By applying the method above, for example by ensuring that the specific data elements are not present in the system, the risk of not complying with this law would be avoided. Of course, as this bill discusses un-encrypted data one way to mitigate the risk would be to store the data encrypted. However, that does not address the problem where the data is displayed or exposed somewhere within the system un-encrypted. If the data is not there we *know* it will not be exposed.

At times data elements are being included in requirements because they seem like something that can be nice to have at some point in time or simply because "this is the way we always done it". Data elements like social security numbers may have been freely used in many older system and they are just kept on being included by habit. This is one reason why a reminder that each data element included carries a risk is needed.

If it is realized that a data element, or a combination of data elements within a system exposes an organization to unacceptable high levels of risk then there is a good chance that alternative ways will be explored to deal with the business problem at hand. However, unless the requirements analysis is conducted with this explicitly in mind less risky alternatives may not be used, as they are not always obvious.

Conclusions

During development of an application, security features at many levels must be used. It all starts when the requirements are defined, through design, coding, and testing of the application. But it does not end there, as the wonderful world of Web applications is always changing, both the application and new vulnerabilities being discovered, continuous education and review of production system are also required.

Defense in depth must be employed using all avenues available and this paper looked at a few of those avenues. Start by learning (for example using the OWASP Web site). Then, create low-risk requirements, and make sure you utilize available security features in your development environment. Test your application, both in development and during periodic reviews in production.
Finally, application level firewalls can provide a layer of ongoing monitoring and protection of your Web application.

References

[1] AppScan, Sanctum. URL: http://www.sanctuminc.com (January 26, 2003)

[2] Ben-Itzhak, Yuval. "Web Application Protection" Web Application Security--The Next Evolution. URL: http://www.devx.com/security/Article/10236/0/page/4 (January 26, 2003)

[3] Carson, Debbie. "Web Application Security" The Java Web Services Tutorial. August 7, 2002. URL: http://java.sun.com/webservices/docs/1.0/tutorial/doc/WebAppSecurity.html (January 26, 2003)

[4] Curphery M. and others. "A Guide to Building Secure Web Applications." The Open Web Application Security Project. September 11, 2002. URL: http://prdownloads.sourceforge.net/owasp/OWASPGuideV1.1.1.pdf?download (January 26, 2003)

[5] Dageforde, Mary. "Trail: Security in Java 2 SDK 1.2" The Java Tutorial. November 25, 2002. URL: http://java.sun.com/docs/books/tutorial/security1.2/index.html (January 26, 2003)

[6] Kunene, Glen. "Software Engineers Put .NET and Enterprise Java Security to the Test." URL: http://archive.devx.com/enterprise/articles/dotnetvsjava/GK0202-1.asp (January 26, 2003)

[7] No author sited. "AppShield." Sanctum. URL: http://www.sanctuminc.com/solutions/appshield/index.html (January 26, 2003)

[8] No author sited. "APS 100." Stratum8. URL: http://www.stratum8.com/docs/aps_datasheet.pdf (January 26, 2003)

[9] No author sited. "Introduction to Web Application Security." Visual Basic and Visual C# Concepts. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vbconintroductiontowebformssecurity.asp (January 26, 2003)

[10] No author sited. "NC-1000 Product Information." NetContinuum. URL: http://www.netcontinuum.com/products/nc1000.html#prevent (January 26, 2003)

[11] No author sited. "SecureIIS Web Server Protection". eEye Digital Security. URL: http://www.eeye.com/html/Products/SecureIIS/Features.html (January 26, 2003)

[12] No author sited. "The Ten Most Critical Web Application Security Vulnerabilities." The Open Web Application Security Project. January 13, 2003. URL: http://prdownloads.sourceforge.net/owasp/OWASPWebApplicationSecurityTopTen-Version1.pdf?download

[13] Northcutt, S. and others. Threat and the Need for Defense in Dept - Security Essentials Day 2 - SANS Track One. The SANS Institute

[14] Peace and Simitian. BILL NUMBER: SB 1386. February 12, 2002. URL: http://info.sen.ca.gov/pub/01-02/bill/sen/sb_1351-1400/sb_1386_bill_20020926_chaptered.html (January 26, 2003)

[15] Poulsen, Kevin. "California disclosure law has national reach." SecurityFocus. January 6, 2003. URL: http://online.securityfocus.com/news/1984 (January 26, 2003)

[16] Pritchard, Carl L. Risk Management: Concepts and Guidance. Arlington: ESI International, 2001.

[17] Sapiro, Benjamin. "Application Level Content Scrubbers." August 22, 2001. URL: http://www.sans.org/rr/firewall/scrubbers.php (January 26, 2003)

[18] Suddeth, S. Brian. "Database - The Final Firewall". January 28, 2002. URL: http://www.sans.org/rr/appsec/final.php (January 26, 2003)

[19] The Open Web Application Security Project (OWASP). URL: http://www.owasp.org/

[20] The Open Web Application Security Project (OWASP) software and documentation repository. URL: http://sourceforge.net/projects/owasp (January 26, 2003)

[21] WebInspect, SPI Dynamics. URL: http://www.spidynamics.com (January 26, 2003)

[22] White, Kelley and Chon, Yong-Gon. "Wide Open on Port 80." Information Security January 2003: 32 - 41