

Web-based Application Security: Common Problems to Know and Avoid

By Scott Redilla, *General Manager, Technical Services*

With so much reliance on the Internet for day-to-day activities and business processes—from online banking and shopping to email—it's hard to imagine a world without web-based applications. While these applications support convenience, efficiency and even cost savings, they also present significant risks for data compromise. In this article, Allied InfoSecurity's Scott Redilla outlines three of the vulnerabilities most frequently encountered during web application security assessments — and explains how you can safeguard your organization's data.



Your security is our business.

555 E. North Lane • Conshohocken, PA 19428
phone: 866.240.0094
email: ask@alliedinfosecurity.com
web: www.alliedinfosecurity.com

© 2008 Allied InfoSecurity, Inc.

The following material is presented as general information only and does not constitute legal advice or a legal opinion. You should seek the advice of legal counsel with respect to your particular circumstances.



Letter from the President

Today's business e-lobby is the ubiquitous Web application. Try to get through any given day without using a web-based application. If you're like most people, doing so would be difficult, if not impossible. We've come to rely on the Internet as a vital—and often only—channel for interactions and transactions. From banking and shopping to renewing insurance registrations and even paying taxes, web-based applications are now at the heart of most people's day-to-day lives.

It's quite likely your organization is equally dependent on such applications. In fact, many companies now rely on web-based systems not only for efficiency enhancements and cost savings, but also as core revenue generators. That means the security of web-based applications is more important than ever. Even so, many organizations have online applications that are dangerously insecure—creating opportunities for potentially disastrous breaches. And executing such breaches may be easier than you think.

Allied InfoSecurity routinely scrutinizes web-based applications for our clients, using both “black-box” and “white-box” security testing. If you aren't certain of the difference between the two—or why organizations often need both—you'll definitely want to read this compelling article by Allied InfoSecurity's Scott Redilla. In this edition of Allied in Touch, Scott outlines three of the most common vulnerabilities we encounter in our clients' web-based applications. Just as important, he explains why thorough, ongoing assessments are vitally important to the security of every business that runs web-based applications.

Sincerely,

A handwritten signature in black ink, appearing to read 'C. Herberger'.

Carl Herberger
President
Allied InfoSecurity

From online banking and shopping to email and even access to private medical information, Web-based applications are everywhere. Supporting unprecedented levels of convenience and access to services and information, these applications are so widely used that it's hard to imagine how we got along without them. And yet, web-based applications can also present serious risks of data compromise. These applications are often riddled with vulnerabilities that provide attackers with direct access to databases or servers. Even a single, seemingly minor vulnerability can serve as an open door through a corporate firewall and directly into a back-end network.

Although there are many types of vulnerabilities that can compromise the confidentiality, integrity and availability of a web application or its data, this article outlines the top three vulnerabilities that Allied InfoSecurity has discovered during the web application security assessments and penetration testing we routinely conduct for our customers.

“ The overall number of instances of SQL injection vulnerabilities found in web applications is generally diminishing due to increased awareness of the problem. ”

Vulnerability #1: SQL Injection

SQL injection vulnerabilities allow specially crafted code to be inserted into the SQL database queries that a website typically performs in response to user input. This injected input can allow attackers to retrieve data

from the back-end database, interfere with application logic, or even execute commands directly on the database server itself.

The overall number of instances of SQL injection vulnerabilities found in web applications is generally diminishing due to increased awareness of the problem. Even so, it's not unusual to discover this issue during application security testing — and that's true even within applications that were designed with SQL injection vulnerabilities in mind.

During a recent penetration test of a customer's web application, we found that the majority of the user input fields were not susceptible to SQL injection attempts. However, there was a “Forgot Password” page that would send users their forgotten passwords via the email addresses on file in the database. Since this user input field did contain a SQL injection vulnerability, it was quite easy for us to insert our own email address in place of the application administrator's email address that was stored in the database. We then used the “Forgot Password” link to have the application send the administrator's password to our email address — giving us full access to the application, as well as confidential company and customer data.

In this case, the application developers were fully aware of the dangers of SQL injection and understood how to write their code and database calls to avoid the problem. As it turned out, the relatively simple “Forgot Password” page had been written many years before and had been “forgotten” as the application evolved.

The bottom line: A single vulnerability in an otherwise secure application can lead to the complete compromise of the application and data.

Vulnerability #2: Parameter Manipulation

Another huge area of concern in web applications is the fact that user input is often “trusted.” In other words, simple data validation is performed by the client’s web browser using client-side code. Since the early days of software development, developers have understood that user input must be properly validated before being processed by an application. With the advent of web browsers and web-based applications, the easiest and most efficient method of validating user input was via client-side code.

Common examples of client-side data validation include checking that dates are in the proper MM/DD/YYYY format or that a telephone number field contains only numeric data that is 10 characters long. This client-side validation works very well as long as the “trusted” user is playing by the rules — that is, using the web browser to interact with the application as the developer intended.

Unfortunately, not all users are trustworthy, and they may be interested in interacting with the application in ways the developer never intended. Thus, the problem with client-side input validation is that there’s no control over the data after it leaves the web browser. If the data is intercepted before it reaches the server, it can be manipulated in a way that is contrary to what the application is expecting to receive.

There are several freely available web proxy tools designed to intercept the data being sent from the browser — allowing attackers to manipulate it in any way they choose before sending it on to the application. Specialized tools known as “fuzzers” have also been created to automate the process of data manipulation to speed up an input validation attack. An attacker can use any of these applications to manipulate data in violation of the client-side validation requirements. The ultimate goal: forcing the application to crash or otherwise perform improperly.

“ Unfortunately, not all users are trustworthy, and they may be interested in interacting with the application in ways the developer never intended. ”



For example, an attacker may replace the validated 10-digit telephone number in a contact form with an alphanumeric string that is 50 characters long. When the application attempts to store this 50-character string into the 10-character telephone number field in the database, the application may crash and, in some cases, provide the attacker with direct access to the database or application server.

Vulnerability #3: Access Control Problems

Broken access controls are one of the most common vulnerabilities found in web applications today. Access controls provide critical security to a web application by allowing users to access only the content for which they’ve been given access rights.

For example, web applications often provide different functionality for normal users versus those with privileges to administer the web application. A user with administrative privileges can perform high-level application functions, such as adding users or resetting passwords. Unfortunately, improperly implemented or “broken” access controls are often easily bypassed.

The simplest form of broken access control allows any regular user of a web application to access high-level administrative functions simply by knowing — or guessing — the URL that provides this high-level access. For example, the URL <https://www.company.com> provides normal users with access to a hypothetical website. In the case of a broken access control, guessing the URL <https://www.company.com/admin/> allows a malicious user to bypass any access control in place and jump directly to the administrative functions page.

While that example is very simplistic, we commonly find similar access control problems when testing web applications for our customers. An illustration of this would be an administrative URL of <https://www.company.com/link/j98d768h213/56ff98726/AddNewUser.jsp> that allows an administrative user to add a new user to the web application. One might argue that this link would be difficult to guess and is therefore relatively secure even if the access control was broken. While security through obscurity is not necessarily a best practice, this URL would, in fact, be very difficult to guess due to the random numbers and letters in the URL.

“ The simplest form of broken access control allows any regular user of a web application to access high-level administrative functions simply by knowing or guessing — the URL that provides this high-level access. ”

Often, however, there is no need to guess the URL because it's provided within the application code. The code snippet below is an example of a method that might be employed to add a “Create a new user” link to an administrator's user interface within the application while hiding the link from a non-administrative user.

As you can see, the URL that was initially very difficult to guess is now easily viewed within the dynamic client side JavaScript code — and the malicious user can navigate directly to the “Create a new user” page.

```
var usrAdmin = false;
-----
if (usrAdmin)
{
    admin Links.addItem("/link/j98d768h213/56ff98726/AddNewUser.jsp",
    Create a new user");
}
```

Access to static resources is another area of concern. Requests for protected web resources are frequently sent directly to the static link for the resource.

This type of functionality is common in software, music, and e-book download sites. For example, a user of an e-book web application may purchase a newly released book, following all of the normal checkout and payment procedures. Once payment is provided, the user is presented with the following link to download the book:

<https://books.company.com/download/0131817590>. The number at the end of the URL appears to be a standard ISBN number used to identify books. It is now a simple matter of replacing the ISBN number with any valid number to download the book of a malicious user's choosing.

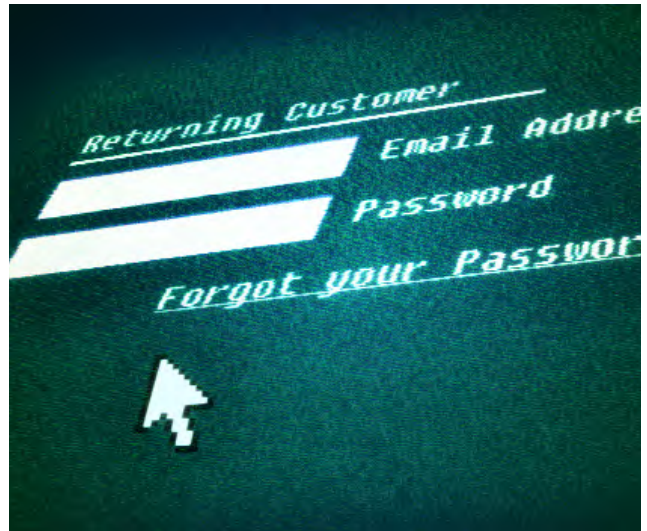
Securing Web Applications

The vulnerability examples highlighted in this article represent only the proverbial tip of the iceberg. New vulnerabilities and attack vectors are being discovered nearly every day—making it critically important to stay up-to-date on the latest vulnerability trends. It's also imperative to regularly test and review your web applications to ensure they are as secure as they can possibly be.

By far, the most effective way to secure a web-based application is through the use of secure coding techniques, thus eliminating any possible vulnerabilities in the application as it is being developed. Unfortunately, this is easier said than done. Web-based applications can have many thousands of lines of code written by many different developers, each of whom often works independently on his or her own component of the application. This process can lead to inconsistencies in the way the code is written, which often negatively affects the security of the application.

In addition to performing secure coding practices during application development, an independent security review and test should be performed before placing a new web application into production or after any major changes to the application. There are two main types of web application security testing: Black Box and White Box.

Black Box testing simulates the methodology used by an attacker to compromise a web application. This type of testing is performed from the “outside” with no prior knowledge of the inner workings of the application. The inputs and outputs of the application are observed while being manipulated by the tester in an attempt to find any flaws in the functionality of the application. This type of testing is often the fastest and most efficient method of uncovering high-level vulnerabilities within an application, especially when performed in conjunction with automated tools like fuzzers.



In contrast, White Box testing involves taking a detailed look under the hood of the application. The web application tester is provided with complete access to the application — including source code, design documentation, and other materials that allow a full review of the application code and functionality.

With access to the source code, it's often possible to find vulnerabilities that would be difficult or impossible to find using a Black Box approach. For example, a back-door administrator login function that was inserted by a developer may be found only by performing a White Box code review.

In most cases, Black Box and White Box techniques can complement each other to provide the most comprehensive web application testing possible. A suspected vulnerability found in the code during a White Box test can be confirmed using Black Box techniques. Conversely, a probable flaw discovered during a Black Box test can be traced to the specific area of concern in the code using White Box techniques.

Regardless of the methods you use, the most important thing to remember is that Web applications must be tested prior to being placed into production and on a regular basis thereafter.

About the Author:

Scott Redilla



About Allied InfoSecurity, Inc.

An independent company focused only on security and staffed by certified security professionals, Allied InfoSecurity is a consulting and outsourcing provider that helps businesses improve and manage their information security programs, mitigate risk, and respond to regulatory and marketplace demands more quickly and effectively than they could on their own.

Web-based Application Security • 7

Scott Redilla is a General Manager, Technical Services for Allied InfoSecurity. In this role, he applies two decades of experience in managing and securing complex IT infrastructures in the telecommunications, chemical, retail and e-commerce industries

Prior to joining Allied InfoSecurity, Mr. Redilla founded and led HighPoint IT, a provider of IT managed services and information security consulting. As the company's senior consultant, he performed security assessments, penetration testing, and social engineering engagements for numerous clients for numerous clients and worked with diverse organizations to optimize their infrastructures and security programs, with emphasis on protected health information and credit card/financial data, to meet HIPAA, PCI & GLBA requirements.

From 2000 to 2005 Mr. Redilla was instrumental in the design, deployment, and ongoing operational security of the chemical industry B2B e-commerce hub, Elemica. In this role he worked with customers to ensure the end to end security of their electronic transactions. In addition, Mr. Redilla created, tested, and maintained an enterprise wide business continuity and disaster recovery plan and performed security assessments on third party partners and vendors. He was also responsible for developing and implementing security policies based on the ISO 17799 framework, and providing security awareness training and education to employees.

Mr. Redilla's previous experience includes hands -on technical responsibility with AT&T Network Systems, Rohm and Haas, and Rotan, Inc., a retail holding company. In those organizations, he was responsible for supporting diverse computing systems including architecting and building secure network infrastructures. He also has extensive experience evaluating vendor solutions, managing budgets, and collaborating with cross-functional project teams.