Symantec Connect

Login

Register

Help

English

Select Language

English

简体中文

Français

Deutsch

日本語

Español

Browse By Product Name

Partners

Forums

Blogs

Groups

Common Security Vulnerabilities in e-commerce Systems

By:

Created 25 Apr 2004 2 Comments

0

by K. K. Mookhey

1. Introduction

The tremendous increase in online transactions has been accompanied by an equal rise in the number and type of attacks against the security of online payment systems. Some of these attacks have utilized vulnerabilities that have been published in reusable third-party components utilized by websites, such as shopping cart software. Other attacks have used vulnerabilities that are common in any web application, such as SQL injection or cross-site scripting. This article discusses these vulnerabilities with examples, either from the set of known vulnerabilities, or those discovered during the author's penetration testing assignments. The different types of vulnerabilities discussed here are SQL injection, cross-site scripting, information disclosure, path disclosure, price manipulation, and buffer overflows.

Successful exploitation of these vulnerabilities can lead to a wide range of results. Information and path disclosure vulnerabilities will typically act as initial stages leading to further exploitation. SQL injection or price manipulation attacks could cripple the website, compromise confidentiality, and in worst cases cause the e-commerce business to shut down completely.

Wherever examples of such vulnerabilities are given in advisories published by Bugtraq, we have given the Bugtraq ID in square brackets. Details of the vulnerability may be viewed by navigating to http://www.securityfocus.com/bid/<bid\_number> .

2. Vulnerabilities

2.1 Background

There are a number of reasons why security vulnerabilities arise in shopping cart and online payment systems. The reasons are not exclusive to these systems, but their impact becomes much greater simply because of the wide exposure that an online website has, and because of the financial nature of the transactions.

One of the main reasons for such vulnerabilities is the fact that web application developers are often not very well versed with secure programming techniques. As a result, security of the application is not necessarily one of the design goals. This is exacerbated by the rush to meet deadlines in the fast-moving e-commerce world. Even one day's delay in publishing a brand new feature on your website could allow a competitor to steal a march over you. We've typically found this in cases where e-commerce sites need to add functionality rapidly to deal with a sudden change in the business environment or simply to stay ahead of the competition. In such a scenario, the attitude is to get the functionality online; security can always be taken care of later. Another reason why security vulnerabilities appear is because of the inherent complexity in most online systems. Nowadays, users are placing very demanding requirements on their e-commerce providers, and this requires complex designs and programming logic.

In a number of cases, we've found that e-commerce sites tout their 128-bit SSL certificates as proof that their sites are well secured. The gullibility of customers to believe in this has reduced over the past few years, but even now there are thousands of web sites displaying Verisign or Thawte certificate icons as proof of their security.

The following sections look at common security vulnerabilities that have been discovered in shopping cart and online payment systems.

2.2 SQL Injection

SQL injection refers to the insertion of SQL meta-characters in user input, such that the attacker's queries are executed by the back-end database. Typically, attackers will first determine if a site is vulnerable to such an attack by sending in the single-quote (') character. The results from an SQL injection attack on a vulnerable site may range from a detailed error message, which discloses the back-end technology being used, or allowing the attacker to access restricted areas of the site because he manipulated the query to an always-true Boolean value, or it may even allow the execution of operating system commands.

SQL injection techniques differ depending on the type of database being used. For instance, SQL injection on an Oracle database is done primarily using the UNION keyword [ref 1] and is much more difficult than on the MS SQL Server, where multiple queries can be executed by separating them with the semi-colon [ref 2]. In its default configuration, MS SQL server runs with Local System privileges and has the 'xp\_cmdshell' extended procedure, which allows execution of operating system commands.

The most publicized occurrences of this vulnerability were on the e-commerce sites of Guess.com [ref 3] and PetCo.com [ref 4]. A 20-year old programmer in Orange County, California, Jeremiah Jacks discovered that it was possible to ferret out highly sensitive data such as credit card numbers, transaction details, etc. from these and a number of other sites using specially crafted URLs containing SQL meta-characters.

SQL injection vulnerabilities have also been discovered in shopping cart software such as the VP-ASP Shopping Cart [bid 9967], IGeneric Free Shopping Cart [bid 9771], Web Merchant Services Storefront Shopping Cart [bid 9301], etc. Of these, the vulnerability in the Storefront Shopping Cart occurred in its login.asp page, and could potentially allow the attacker to execute malicious database queries, without needing to authenticate to the web site.

2.3 Price Manipulation

This is a vulnerability that is almost completely unique to online shopping carts and payment gateways. In the most common occurrence of this vulnerability, the total payable price of the purchased goods is stored in a hidden HTML field of a dynamically generated web page. An attacker can use a web application proxy such as Achilles [ref 5] to simply modify the amount that is payable, when this information flows from the user's browser to the web server. Shown below is a snapshot of just such a vulnerability that was discovered in one of the author's penetration testing assignments.

Figure 1: Achilles web proxy

The final payable price (currency=Rs&amount=879.00) can be manipulated by the attacker to a value of his choice. This information is eventually sent to the payment gateway with whom the online merchant has partnered. If the volume of transactions is very high, the price manipulation may go completely unnoticed, or may be discovered too late. Repeated attacks of this nature could potentially cripple the viability of the online merchant.

Similar vulnerabilities have also been found in third-party software such as in the 3D3 ShopFactory Shopping Cart [bid 6296], where price and item-related information was stored in client-side cookies, which could easily be manipulated by an attacker. Similarly, Smartwin Technology's CyberOffice Shopping Cart 2.0 could be attacked by downloading the order form locally, and resubmitting it to the target server with the hidden form fields modified to arbitrary values [bid 1733].

2.4 Buffer overflows

Buffer overflow vulnerabilities are not very common in shopping cart or other web applications using Perl, PHP, ASP, etc. However, sending in a large number of bytes to web applications that are not geared to deal with them can have unexpected consequences. In one of the author's penetration testing assignments, it was possible to disclose the path of the PHP functions being used by sending in a very large value in the input fields. As the sanitized snapshot below shows, when 6000 or more bytes were fed into a particular field, the back-end PHP script was unable to process them and the error that was displayed revealed the location of these PHP functions.

Figure 2: PHP timeout error

Using this error information it was possible to access the restricted 'admin' folder. From the structure of the web site and the visible hyperlinks there would have been no way to determine that there existed the 'admin' directory within the 'func' sub-directory below the main $DocumentRoot.

Multiple buffer overflows were also discovered in the PDGSoft Shopping Cart [bid 1256], which potentially allowed the attacker to execute code of his choice by over-writing the saved return address.

Error pages can serve as a valuable source for critical information. These errors can be induced in web applications that do not follow strict input validation principles. For instance, the application may expect numeric values and would fail when alphabets or punctuation characters are supplied to it. This is exactly what has happened in the case below. Here, the e-commerce website used numbers for its various pages. Users would navigate using a link such as http://www.vulnerablesite.com/jsp/Navigate.jsp?pageid=123. By manipulating the URL and supplying the value 'AA' for the pageid, the following error was induced:

Figure 3: Discovering information through navigation errors

If you observe carefully, the highlighted information reveals the Oracle Application Server version as Oracle 9iAS 9.0.3.0.0 as well as certain third-party components being used by the web application, such as Orion Application Server. It also reveals the path where other (possibly vulnerable) .jsp scripts exist - /scripts/menu.jsp .

2.5 Cross-site scripting

The Cross-site Scripting (XSS) [ref 6] attack is primarily targeted against the end user and leverages two factors:

The lack of input and output validation being done by the web application

The trust placed by the end-user in a URL that carries the vulnerable web site's name.

The XSS attack requires a web form that takes in user input, processes it, and prints out the results on a web page, which also contains the user's original input. It is most commonly found in 'search' features, where the search logic will print out the results along with a line such as 'Results for <user\_supplied\_input>'. In this case, if the user input is printed out without being parsed, then an attacker can embed JavaScript by supplying it as part of the input. By crafting a URL, which contains this JavaScript, a victim can be social engineered into clicking on it, and the script executes on the victim's system. A typical XSS attack URL would look like this: http://www.vulnerablesite.com/cgi-bin/search.php?keywords=&lt;script>alert("OK")&lt;script>. In this case, when the victim clicks on this link, a message box with the text "OK" will open up on his system.

In most cases, the attacker would craft the URL in order to try and steal the user's cookie, which would probably contain the session ID and other sensitive information. The JavaScript could also be coded to redirect the user to the attacker's website where malicious code could be launched using ActiveX controls or by utilizing browser vulnerabilities such as those in Internet Explorer or Netscape Navigator.

However, the JavaScript can also be used to redirect the user to a site that looks similar to the original web site and requests the user to enter sensitive information such as his authentication details for that web site, or his credit card number or social security number. A related attack is shown below:

Figure 4: Phishing scam (Source: Article on SecurityFocus http://www.securityfocus.com/infocus/1745)

In this case, the attacker has opened up two windows on the victim's system. The one in the background is the original Citibank web site, whereas the pop up window in front of it requests for the user's debit card number, PIN, and card expiration date. On hitting the submit button, this information is sent to the attacker's server. Called a 'phishing' attack [ref 7], it was done by sending a spoofed email that claimed to originate from Citibank and asked users to verify their details. The link in the spoofed email looked something like this http://www.citibank.com:ac=piUq3027qcHw003nfuJ2@sd96V.pIsEm.NeT/3/?3X6CMW2I2uPOVQW Most users would not be aware that as per HTTP rules, this link would actually go to sd96v.pisem.net (highlighted above), and not www.citibank.com

Similar attacks can be carried out if the web application has scripts that redirect users to other parts of the site, or to other related sites. For instance, in one of our assignments, the web application had a script that was used to send the user to dynamically created parts of the web site: http://www.vulnerablesite.com/cgi-bin/redirect.php?url=some\_dynamic\_value Due to the lack of security awareness of the web developers, they did not realize that an attacker could craft a URL such as http://www.vulnerablesite.com/cgi-bin/redirect.php?url=www.attackersite.com and send it to a victim. This URL can be trivially obfuscated by hex-encoding the part that follows 'url=' or by converting the attacker's IP address into hexadecimal, octal or double-word values. For instance if the attacker's IP address is 192.168.0.1, the URL could be crafted as follows: http://www.vulnerablesite.com/cgi-bin/redirect.php?url=http://7934518627/ .

2.6 Remote command execution

The most devastating web application vulnerabilities occur when the CGI script allows an attacker to execute operating system commands due to inadequate input validation. This is most common with the use of the 'system' call in Perl and PHP scripts. Using a command separator and other shell metacharacters, it is possible for the attacker to execute commands with the privileges of the web server. For instance, Hassan Consulting's Shopping Cart allowed remote command execution [bid 3308], because shell metacharacters such as |;& were not rejected by the software. However, directory traversal was not possible in this software.

In another case, Pacific Software's Carello Shopping Cart [bid 5192] had a vulnerable DLL that allowed the execution of remote commands due to directory traversal attacks that could be carried out using a specially crafted URL.

2.7 Weak Authentication and Authorization

Authentication mechanisms that do not prohibit multiple failed logins can be attacked using tools such as Brutus [ref 8]. Similarly, if the web site uses HTTP Basic Authentication or does not pass session IDs over SSL (Secure Sockets Layer), an attacker can sniff the traffic to discover user's authentication and/or authorization credentials.

Since HTTP is a stateless protocol, web applications commonly maintain state using session IDs or transaction IDs stored in a cookie on the user's system. Thus this session ID becomes the only way that the web application can determine the online identity of the user. If the session ID is stolen (say through XSS), or it can be predicted, then an attacker can take over a genuine user's online identity vis-à-vis the vulnerable web site. Where the algorithm used to generate the session ID is weak, it is trivial to write a Perl script to enumerate through the possible session ID space and break the application's authentication and authorization schemes.

This was illustrated in a paper by David Endler [ref 9], "Brute-Force Exploitation of Web Application Session IDs", where he explains how session IDs of sites like www.123greetings.com, www.register.com, and others could be trivially brute-forced. Similarly, in one such instance, we discovered that the order ID for the user's transactions was not generated randomly, and it was possible to access the orders placed by other users simply by writing a Perl script that enumerated all possible order IDs within a given range. The most pertinent point here is that although web application may have mechanisms to prevent a user from multiple password guessing attempts during authentication, they do not usually prevent a user from trying to brute-force sessions IDs by resubmitting the URLs as described in Endler's paper.

3. Countermeasures

The most important point is to build security into the web application at the design stage itself. In fact, one of the key activities during the design phase should be a detailed risk assessment exercise. Here, the team must identify the key information assets that the web application will be dealing with. These could include configuration information, user transaction details, session IDs, credit card numbers, etc. Each of these information assets needs to be classified in terms of sensitivity. Depending upon the tentative architecture chosen, the developers along with security experts must analyze the threats, impact, vulnerabilities and threat probabilities for the system. Once these risks are listed out, system countermeasures must be designed and if necessary the architecture itself may be modified. Countermeasures should also include strict input validation routines, a 3-tier modular architecture, use of open-source cryptographic standards, and other secure coding practices.

Some excellent resources on secure coding are David Wheeler's book "Security Linux Programming HOWTO" [ref 10], Michael Howard's "Writing Secure Code", and John Viega's "Secure Programming Cookbook for C and C++". The Open Web Application Security Project's Guide [ref 11] is also a highly useful document on web application security issues.

4. Conclusion

The vulnerabilities discussed in this article are not necessarily exclusive to shopping carts or online payment systems. They could easily be present in other types of web applications as well. However, in the case of e-commerce systems, the vulnerabilities acquire a graver dimension due to the financial nature of transactions. What is at stake is not only a direct loss of revenues, but companies may face a serious loss to their reputations as well. In some cases, they may be faced with legal penalties for violating customer privacy or trust, as in the case of Guess.com and PetCo.com. It is of paramount importance for designers and developers of web applications to consider security as a primary design goal and to follow secure coding guidelines in order to provide the highest possible degree of assurance to their customers.

References

1. SQL injection and Oracle, Pete Finnigan http://www.securityfocus.com/infocus/1644

2. Advanced SQL injection, Chris Anley http://www.nextgenss.com/papers/advanced\_sql\_injection.pdf

3. News article on SQL Injection vulnerability at Guess.com http://www.securityfocus.com/news/346

4. Jeremiah Jacks at work again, this time at PetCo.com http://www.securityfocus.com/news/7581

5. Achilles can be downloaded from http://achilles.mavensecurity.com/

6. CERT Advisory Malicious HTML HTML Tags Embedded in Client Web Requests http://www.cert.org/advisories/CA-2000-02.html

7. Definition of 'phishing' http://www.webopedia.com/TERM/p/phishing.html

8. Brutus can be downloaded from http://www.hoobie.net/brutus/

9. Brute-Force Exploitation of Web Application Session IDs, David Endler http://www.idefense.com/application/poi/researchreports/display

10. Secure Programming for Linux and Unix HOWTO, David Wheeler, http://www.dwheeler.com/secure-programs/

11. OWASP Guide http://www.owasp.org/

About the author

View more articles by K. K. Mookhey on SecurityFocus.

Comments and/or reprint requests can be sent to the editor.

This article originally appeared on SecurityFocus.com -- reproduction in whole or in part is not allowed without expressed written consent.

Tags: Products, Endpoint Protection

Subscriptions (0)

View Profile

Jump to latest comment

2 Comments

Login or Register to post comments.

Contact UsPrivacy PolicyEarn RewardsRewards Terms and Conditions

© 2019 Symantec Corporation