

Secure Ecommerce Framework for the .NET Environment

Reuben Sant

Department of Computer Science

University of Malta

reubensant@gmail.com

Abstract

Ecommerce systems are common attack targets due to the financial value of their transactions. Several issues in the software development process contribute towards the introduction of security vulnerabilities in web applications. This dissertation proposes to prevent security vulnerability introduction in ecommerce ASP.NET C# web applications. As a result, the developers can focus more on the functionality of their application rather than the repetitive security management tasks on the developed application.

1 Introduction

Information and communication technologies continue to pervade our lives in various aspects which include health, education, entertainment and ecommerce. People need to be able to trust computer systems as the dependence on them increases. The *Trustworthy Computing* vision (CRA, 2003) refers to computer systems that are intuitive, controllable, reliable and predictable and that ensure availability and security. Secure coding is not trivial and poor code security management may leave the developed web application vulnerable to attack or turn the application into a launch pad for serious attacks.

This paper is organized as follows: Section 2 provides background information to secure coding in web applications. Section 4 outlines the aims and objectives of the project. Section 5 briefly describes the design of the developed secure framework. Section 6 highlights the achievements of the project and Section 7 presents a conclusion.

2 Background

Security vulnerabilities in a system can be at the application, server and network level. Unpatched software, viruses and trojan horses may all expose a system to attack. These security issues should be addressed in any system; however these can easily be bypassed with an attack on a web application. The aim of this project is to focus on the application-level security vulnerabilities which are introduced in the next subsections. (Howard and LeBlanc, 2003)

2.1 Web Application Vulnerabilities

This section presents a brief overview of the most common security vulnerabilities found in ecommerce web applications.

2.1.1 Cross-Site Scripting

Cross-Site Scripting is a form of input validation vulnerability (Howard et al, 2005). Any web application that directly echoes the user input to a webpage without validation has an input trust issue and is vulnerable with Cross-Site Scripting. Echoed input might be a client-side script which is then executed by the browser and runs in the context of the vulnerable domain, having access to the client side cookies.

2.1.2 SQL Injection

SQL injection is a common form of input validation vulnerability (Howard et al, 2005) resulting from the improper string concatenation of SQL commands with non-validated user input. The following code fragment builds a string containing an SQL statement constructed by concatenating SQL commands with user input.

```
string sql = "SELECT user, pass " +
  "FROM client WHERE email='" +
  Email + "'";
```

Input string (me@email.com' OR 'a'='a) in the variable Email results in the following SQL statement:

```
SELECT user, pass FROM client
WHERE email='me@email.com' OR 'a'='a';
```

The WHERE clause of this statement will evaluate to true. Thus, all the records in the client table are returned.

2.1.3 Providing Too Much Information

The role of web applications is to convey information to its users based on the input. However, as explained by (Howard et al, 2005), some information simply cannot be revealed even if it is useful to legitimate users, because it is also useful for an attacker. Information which the attackers look for and should never be leaked includes: version and configuration information; an indication if a username is correct or not in a failed login attempt; host and network information; exception messages, especially SQL errors; absolute file paths, revealing the layout of the server's hard drive; stack trace.

2.1.4 Incorrect Error Handling

Errors or exceptions may arise in any web application. Failure to handle exceptions may result in a denial of service situation as unhandled errors may result in application failure. There are several ways to handle errors incorrectly (Howard et al, 2005). Ignoring errors might bring down the application to an unrecoverable state; misinterpreting errors; functions which return without indicating success or failure may return dangerous values; handling wrong exceptions; handling all exceptions, resulting in the piling up of errors until the application fails.

2.1.5 Poor Password Authentication

As described by (Scambray et al, 2006), bad choice of passwords makes it easier for attackers to guess passwords. (Howard et al, 2005) insist that password-based systems are not the best authentication mechanism possible but we simply cannot just get rid of password systems because other solutions don't seem to be enough on their own. Security experts suggest strong password

while users prefer easy-to-remember weak passwords.

2.1.6 Weak Random Number Generators

Traditional random number generators were not designed with security in mind. Random number generation was designed to be a repeatable process according to the selected random seed. These predictable sequences are useful in repeatable experiments such as Monte Carlo algorithms. A predictable random number generator is dangerous in situations when the generated number should not be guessed by an attacker. (Howard et al, 2005).

2.1.7 Direct Access to Application Memory

It is not a coding defect to call unmanaged code from a managed .NET application. However, the introduced unmanaged code may have several security vulnerabilities. The problem occurs when low level languages allow the mixture of user data and program control instructions for the sake of performance and allow direct access to application memory. A buffer overrun occurs when the user is allowed to write his input beyond the end of the allocated buffer to hijack the execution of the running process. (Howard et al, 2005).

2.2 Vulnerability Trends

Figure 1 shows the Top 4 Vulnerabilities in 2006 compared with the data from 2004 and 2002. The percentages are calculated with respect to the totals from the top 20 vulnerabilities which were distilled from the MITRE data (MITRE, 2007).

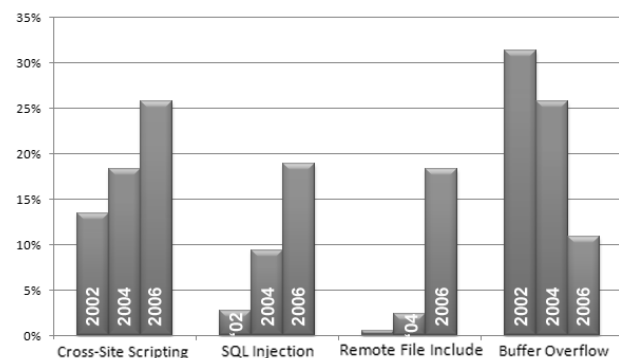


Figure 1: Number of attack incidents.

These results show that the public CVE reports for the top three vulnerabilities for 2006, namely Cross-Site Scripting, SQL Injection and Remote File Inclusion, have been constantly rising throughout the years. The opposite can be said

about the Buffer Overflow Vulnerability. This indicates that attackers are moving from OS vulnerabilities to more fertile ground - web applications. These include ecommerce web applications where the reward, possibly financial, justifies the effort for an attack.

3 .NET Framework Security Assessment

This section aims to analyze various .NET libraries, conventions and mechanisms, which if used correctly, may improve the security level of an ecommerce system. The following subsections identify the security strengths and weaknesses of various .NET components. This study will then lead to the design and implementation of a software artifact which maximizes the use of the identified security strengths and provides alternatives to the security weaknesses.

3.1 Database Access

An SQL injection vulnerability exists when untrusted input data is mixed with the control in the construction of SQL statements. One solution would be to enforce type safety and allow the developer to separate data from control in query generation. LINQ queries are part of the language syntax (C# in our case), hard written in the source code and therefore not strings which may be manipulated by an attacker.

3.2 Session Management

A predictable Session ID generator allows attackers to deduce valid Session IDs and eventually hijack sessions. The MSDN ASP.NET Technical Article (MSDN, a) states that the Session ID is made up of a 120-bit cryptographically random string. However, session hijacking may not only be accomplished by predicting Session IDs but also by stealing issued Session IDs. The ASP.NET Session State Implementation does not seem to provide a way to, at least prevent, against accepting stolen Session IDs in incoming cookies.

3.3 Request Validation

This feature is useful in many cases and should generally be left enabled. However, the design of the Request Validation mechanism unfortunately leaves little room for extensibility and flexibility in the actions the developer can take on the input with un-encoded HTML tags. The thrown exception is difficult to handle because it is thrown before the

page actually starts executing. This mechanism is good at detecting illegal input, but cannot perform filtering. If un-encoded HTML input is required, the developer must disable the Request Validation for the whole page or for the whole application, leaving the system potentially vulnerable to scripting attacks.

3.4 Error Reporting

Error messages which contain useful information for the attacker. Developers may not be aware of the security implications to simply write out the full exception messages.

3.5 Path Canoncalization

.NET file and path handling functions do not discriminate between file paths (supplied as user input) which are relative to the web applications and absolute file paths. An attacker can specify an absolute path and read any file that the ASP.NET user has permission to read if no validation is performed on the input file path. The best way to ensure that a specified file path is a virtual path relative to the web application is to supply the user input to the `Server.MapPath()` method. This method takes a virtual path and converts it to the absolute path on the web application directory.

4 Aims and Objectives

The main objective of this project is to prevent security vulnerability introduction in ecommerce ASP.NET C# web applications. The aim of this project is not to re-invent functionality that is already available in .NET libraries, but to extend their secure usage and to provide a secure framework which prevents the introduction of common security vulnerabilities found in ecommerce systems.

The aim is to develop .NET library to analyze and filter the user HTTP request and application HTTP response in order to detect and prevent a number of identified security vulnerabilities. Security flaws may be introduced with poor design and implementation decisions. This project aims to provide web application developers generic secure implementations of common functionality found in ecommerce systems. The provided secure framework still leaves web application developers free to bypass the provided secure implementations and introduce security vulnerabilities in the developed code. To mitigate this problem,

this project also aims to offer a static code analysis tool which scans the C# source code for common coding flaws that are known to introduce security vulnerabilities.

By reaching these objectives, the burden on web application developers to maintain security throughout the application is alleviated. Security consideration should always be part of the development lifecycle, however, developers can concentrate more on the business logic of the developed application rather than focusing on the constant attention required to ensure that no security vulnerabilities are introduced in the code.

5 Design

Sentry.NET is made up of the following components.

- HTTP Modules;
- Web Controls;
- Common Ecommerce Functionality;
- Static Code Analysis Tool.

The following sections explain the purpose and the design of each of these components.

5.1 HTTP Modules

ASP.NET HTTP Modules are able to act on the user's HTTP request and modify the application's HTTP response at runtime, before and after the ASP.NET page handler executes, respectively. (MSDN, b) This is illustrated in Figure 2.

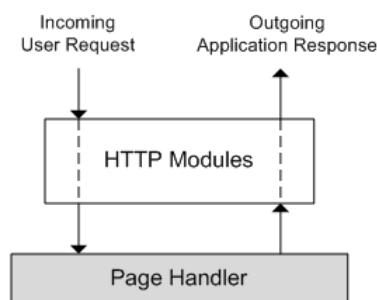


Figure 2: ASP.NET HTTP Modules can execute both before and after the page handler executes

This makes HTTP modules a good choice for sanitizing user input and filtering application output. The developed HTTP modules report any security issues related to the user input to the ASP.NET page handler. The developer can handle

these issues in the code-behind file for a particular page and act appropriately. All the security issues encountered in the page lifecycle are logged for examination.

The following subsections describe the specific architecture for each developed HTTP Module.

5.1.1 XSS Protection Module

The aim of this HTTP Module is to prevent Cross-Site Scripting attacks by identifying any user input that is directly present as application output as illustrated in Figure 3. User input may come from query strings, form fields and cookie values.

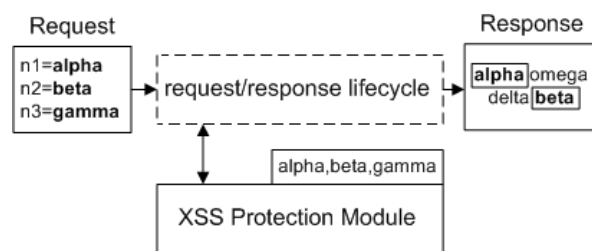


Figure 3: XSS Protection Module

Input values that are directly present in the output are deleted or html-encoded. This approach contrasts with the ASP.NET built-in Request Validation functionality, where the server simply rejects un-encoded HTML with an exception, instead of giving an option for HTML encoding or deletion of the echoed input. In cases where un-encoded HTML input is required by the developer, the built-in Request Validation mechanism cannot be used.

5.1.2 Limit Input Characters Module

The aim of this module is to prevent Input Validation attacks such as SQL Injection and Cross-Site Scripting attacks by limiting the set of characters allowed in the input as illustrated in Figure 4.

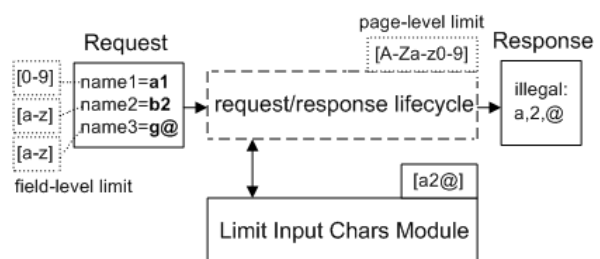


Figure 4: Limit Input Chars Module

When the ASP.NET page handler is initialized, the Limit Input Characters Module, checks for, and keeps a note of disallowed characters present in the input. First, the allowed character set specified at the page-level is checked. All the input fields in the request must satisfy this character set. Then the character sets specified at field-level are optionally checked. If the developer chooses to validate both at the page-level and at the field-level, an input field must satisfy both input character sets for the validation to succeed.

5.1.3 Session Verification Module

The ASP.NET session handler implementation is quite secure (MSDN, a). The ASP.NET session mechanism was investigated and no functionality was found to determine if the Session ID supplied by an incoming cookie originates from the legitimate user who originally requested and possibly authenticated the session, or if it was hijacked by a malicious user. The idea is to create a verification string to be appended to the issued Session ID. The verification string is based on the Session ID issued by ASP.NET and some values specific to the calling machine.

The method of communication between the client browser and web server still consists of standard HTTP requests and responses. This verification process is not a replacement for encrypting network traffic. If network traffic is not encrypted and the verification string is stolen, it is easy for the attacker to steal the request parameters to forge his request. It is important to re-stress that this solution aims to raise the bar for the attackers by preventing Session Hijacking attempts from stolen Session IDs, not to prevent Session ID theft performed by other attacks such as Network Eavesdropping or Cross-Site Scripting.

5.1.4 Safe Cookies Module

A successful Cross-Site Scripting attack attempts to access data on the client machine. This data includes, but is not limited to, cookie values. Luckily, modern web browsers support the HTTPOnly cookie flag which disallows client side scripts from accessing HTTPOnly cookies. The Safe Cookies Module sets the HTTPOnly cookie to the outgoing cookies as illustrated in Figure 5.

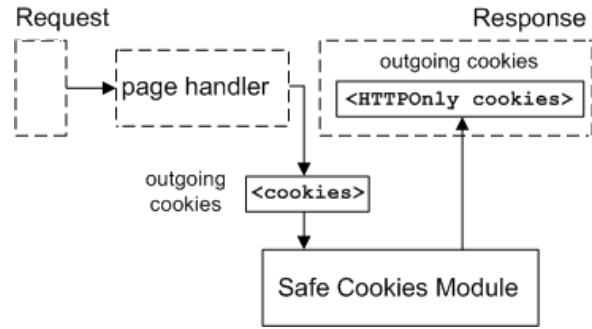


Figure 5: Safe Cookies Module

5.1.5 Filter SQL Errors Module

Information from SQL error messages might contain the information required to perform a database attack. The developer should avoid to write out SQL error messages to application output. This module takes the application response and filters it for any SQL error messages as illustrated in Figure 6. Any SQL errors found in the output will be replaced by a note indicating that the error message was hidden. The list of SQL error messages is stored in an XML file which can be modified by the developer.

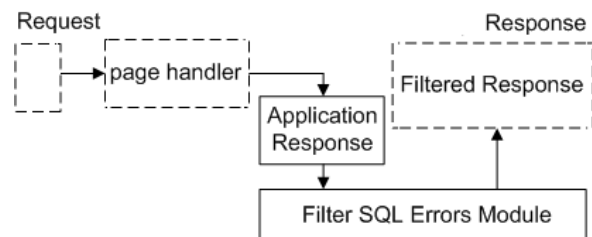


Figure 6: Filter SQL Errors Module

5.2 Web Controls

ASP.NET web server controls are special tags, similar to HTML tags but are understood by the server and rendered into one or more HTML elements to create more complex elements.

5.2.1 One-Time Password

A drawback of password based systems is that the actual password needs to be submitted over a public network in the authentication process. If the password is never transmitted over the network, an attacker could never steal it by eavesdropping. However, some form of ‘token’ based on the original password still needs to be transmitted over the network. This ‘token’ must have the following properties: must be based on the original pass-

word; cannot be the secret password itself; the original secret password cannot be derived from it; must be valid for only one successful authentication, so it would be useless if stolen; cannot be derived from previous ‘tokens’; must not be too short or too long; must be interoperable with existing one-time password authentication mechanisms. This ‘token’ is described by RFC2289 in the form of a One Time Password whose implementation is included in the framework.

5.3 Common Ecommerce Functionality

The security vulnerabilities present in a piece of code may not only be related to input validation or application output but also the implementation details of the code itself. The way the data is stored and accessed, the data structures and algorithms used and the coding techniques used are all important factors to consider when writing secure code.

Common elements found in ecommerce systems were designed with security in mind. A database was designed to store three entities which are important to any ecommerce system. These are *users*, *products* and *shopping cart items*. The properties of each of these entities are not meant to be complete but as general as possible. The web developer is free to store additional properties for each entity in a separate database.

5.4 Static Code Analysis Tool

The framework described until now in this chapter provides the foundations for a web developer to build a secure ecommerce application. However, since ecommerce systems vary in their functionality and developers are free to ignore the security features provided by the secure libraries, the web developer can still introduce security vulnerabilities in the code. The developer is offered a Static Code Analysis Tool which detects common patterns of security vulnerabilities in the static code. The aim of the tool is to warn the user of such code patterns and suggests further investigation or alternative approaches when possible. This tool is not a substitute for testing. It should be used to check the code syntax before it is passed on to testing.

6 Results

The framework was evaluated by building a sample web application that uses Sentry.NET and another web application which does not make use of the secure framework. The results show how easy

it is to introduce security vulnerabilities with a simple mistake, with a lack of security knowledge by simply following online code samples. The framework works well with complex ecommerce web application where the business logic is not trivial. These situations require constant developer attention on the functionality and may cause the developer to overlook security. Sentry.NET is suitable for ecommerce web application and would be an overkill for websites that simply display static information without expecting user input.

The encouraging results show that the identified common security vulnerabilities in ecommerce systems are covered by the provided secure framework. The burden on web application developers to maintain security throughout the application is alleviated. Although security considerations should always be part of the development lifecycle, the developers can focus more on the functionality of their application rather than the repetitive security tasks on the developed application.

7 Conclusion

This dissertation has presented the Sentry.NET framework for developing secure ecommerce web applications. Rapid application development environments may cause web developers to focus on quickly getting the job done without constant attention to security. The solution, apart from providing reusable common core functionality for building secure ecommerce systems, enables the developer to perform important security checks on the actual user HTTP request and application HTTP response, while performing the required remedy actions. Apart from the runtime checks, the solution also provides a tool that scans the static source code for code patterns that are known to introduce security vulnerabilities. This allows the developer to focus on the functionality of the ecommerce web application rather than the security details of each task, drastically easing the task of secure coding.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Mr. Mark Vella, for his help, support and constant supervision throughout the project.

References

- Computing Research Association 2003. *Four Grand Challenges in Trustworthy Computing* Conference II on Grand Research Challenges in Computer Science and Engineering - November 1619.
- M. Howard, D. LeBlanc 2003. *Writing Secure Code* Second Edition Microsoft Press
- M. Howard, D. LeBlanc, J Viega 2005. *19 Deadly Sins of Software Security* McGraw-Hill
- MITRE 2007. *Vulnerability Type Distributions in CVE*
<http://cwe.mitre.org/documents/vuln-trends>
- MSDN *Underpinnings of the Session State Implementation in ASP.NET* Second Edition
<http://msdn.microsoft.com/en-us/library/aa479041.aspx>
- MSDN *ASP.NET HTTP Modules and HTTP Handlers Overview* Microsoft KB article 307985
<http://support.microsoft.com/kb/307985>
- J Scambray, M. Shema, C. Sima 2006. *Hacking Exposed: Web Applications* Second Edition McGraw-Hill