

Card payment implementation guide for ASP.NET and PHP websites

Alexander Hutchinson

Thesis
Business Information Technology
August 2012



Business Information Technology

<p>Author Alexander Hutchinson</p>	<p>Year of entry 2009</p>
<p>Title of report Card payment implementation guide for ASP.NET and PHP websites</p>	<p>Number of pages and appendices 83 + 8</p>
<p>Supervisor Juhani Välimäki</p> <p>E-commerce is growing by around 13% year on year, but more than 50% of orders on e-commerce sites are still abandoned prior to payment. This is due to a number of factors, such as consumer trust, and issues of site usability. This study tackles both factors by providing a guide for web developers to be able to implement effective card payment solutions.</p> <p>The guide firstly explains about the process of card payments, and explains about the importance of the Payment Gateway (a third party handling card payments on behalf of online merchants). Payment gateway integrations generally fall into two categories, hosted solutions (requiring the e-commerce site user to be transferred to the payment gateway site) and API's (requiring the handling of credit card details by the e-commerce site) This study focuses on the later due its complexity and important security considerations.</p> <p>The guide gives ASP.NET and PHP developers usable code examples, showing how to implement a basic payment page, and explains in depth about the importance of validating user input, to reduce security threats, and about aspects of checkout page usability. It also proposes the importance of informing consumers about SSL to increase awareness and develop trust, further preventing security problems, and ultimately loss of sales. Concerns about loss of personal data, and card fraud are the main concerns for consumers. The guide explains that SSL is the main technology used to secure the transfer of personal information across the internet, and is routinely used in e-commerce, in accordance with security guidelines.</p> <p>This guide then explains one aspect of maximising online sales while keeping consumers safe through explaining the technical implementation for accepting card payments for ASP.NET and PHP websites hosted on IIS for Windows Server, and Apache Web Server.</p>	
<p>Keywords e-commerce, ASP.NET, PHP, API, SSL</p>	

Table of contents

Glossary.....	3
1 Introduction.....	6
2 The big picture of the e-commerce environment.....	11
2.1 The payment process.....	12
2.1.1 The parties involved.....	12
2.1.2 Stages of an online payment.....	14
2.2 Security threats and other risks to online payments.....	17
2.2.1 Process complexity.....	17
2.2.2 Sniffer Programs.....	17
2.2.3 Backdoors.....	18
2.2.4 Spoofing and DNS cache poisoning.....	21
2.3 Security methods employed.....	22
2.3.1 Asymmetric cryptography.....	22
2.3.2 Secure Sockets Layer (SSL).....	23
2.3.3 Digital certificates.....	25
2.3.4 Certificate Authorities and requesting a certificate.....	25
2.3.5 PCI Standards.....	27
2.3.6 Payment Application Best Practice (PABP).....	28
2.4 Payment gateway integrations.....	28
2.4.1 Factors influencing payment gateway choice.....	30
2.4.2 Transaction types.....	31
2.4.3 Google Checkout and Amazon Payments.....	32
2.4.4 PayPal.....	33
2.4.5 Authorize.net.....	35
2.5 Chapter Summary.....	36
3 Front end implementation.....	38
3.1 HTML Forms.....	38
3.1.1 Form attributes.....	39
3.1.2 Payment page HTML.....	40
3.2 User input validation.....	46

3.2.1	The composition of a credit/debit card number	46
3.2.2	Regular expression validators.....	47
3.2.3	JavaScript validation and form submission script.....	48
3.3	Payment page usability.....	49
3.4	SSL Certificate logos and informing users.....	52
3.5	Chapter Summary.....	53
4	Back end implementation	55
4.1	Payment gateway user accounts	55
4.1.1	Paypal manager account	56
4.1.2	Authorize.net test account	56
4.2	Server side code implementation	57
4.2.1	PayPal Payflow Pro	57
4.2.2	Authorize.net Advanced Integration Method (AIM)	61
4.3	Enabling SSL for Apache and Windows Server	68
4.3.1	IIS 7 for Windows Server 2008	68
4.3.2	Apache	70
4.4	Chapter Summary.....	72
5	Evaluation	73
6	Further development	75
7	Summary.....	77
8	References	81
9	Appendices.....	87
9.1	Appendix 1 - Diagram of the payment authorization process.....	87
9.2	Appendix 2 – Recognisable Certificate Authority brand logos	88
9.3	Appendix 3 - Current browser interface SSL connection and certificate notifications	89
9.4	Appendix 4 – Payment Application Best Practices (PABP).....	90
9.5	Appendix 5 – Rendered HTML payment buttons	91
9.6	Appendix 6 – Nordea Bank’s SSL information to users.....	92
9.7	Appendix 7 – Effective payment page design example	93
9.8	Appendix 8 – Implementing SSL in IIS 7.0 Windows Server 2008	94

Glossary

Algorithm	An encryption algorithm is the often multi stage process of converting plain text into encrypted text, the more encryption stages the stronger the encryption.
API	Application Programming Interface is a library of data structures, object classes, routines and other resources made available over a network to be used by calling applications.
Attribute (HTML)	Is a modifier of an HTML element, included within the HTML element tag.
Back end	Server side code and related architecture, including the server itself.
E-commerce	Electronic commerce is the process of selling goods and services online through a web application.
Encryption	A mathematical system whereby plain text is converted into a code, where only the holder of the encryption key can convert the code back to plain understandable text.
Front end	User interface code and related implementation.
IIS	Internet Information Services is the web server included in Windows Server instances. Included with IIS are commonly a number of other configurable modules.
Implementation	The code writing, compiling, installation and configuration of a program.
Library	A collection of code files often packaged into a non editable .DLL or .LIB file, which enables the reuse of the code in many applications.

Merchant	A company selling goods and/or services. The term is used in the card payment industry, and is commonly used in e-commerce.
NVP	Name and Value Pairs describes a type of data set involving a collection of names, or keys, and a values relating to those names or keys.
Payment gateway	A company providing a collection of card payment services to online merchants such as access to many payment processors, fraud prevention and batch settlement. See section 2.1.1
PHP	Hypertext pre-processor. A popular open source server scripting language for building web applications.
Regular Expression	Commonly abbreviated to RegEx, these expressions provide a way to check whether a text string or number matches the that specified in the expression. One expression can also check for a large number of variances efficiently.
SDK	Software Development Kit is a package of resources and configurations often associated with an IDE (Integrated Development Environment) to assist in the development of certain types of applications.
SEO	Search Engine Optimization is the process of undertaking tasks to improve the rating of a site with search engines. The hope is that the site then appears nearer to the top of the search results, making it easier to find.
SSL	Secure Socket Layer. An Application layer TCP-IP protocol encrypting the contents of HTTP packets. See section 2.3.2.
SQL	Structured Query Language, developed originally by IBM, is the standard language for interacting with a database.

Validation

The checking of user input fields against defined allowable values, prior to further processing.

1 Introduction

Background

During the late 1990's electronic commerce, later termed e-commerce, began to grow rapidly around the world. Between 1997 and 2000 in the United States alone as many as 12.000 e-commerce companies were started, involving a total of \$100 billion of investment (Schneider, 2011, p. 9). Investors were eager to invest in e-commerce companies seeking to capitalise on the rapid period of growth, leading the media to coin the term 'dotcom boom'. Unfortunately at the time it was seemingly enough just to be an online business to make money. It of course transpired that a successful business needs more than a good idea and a website, consequently in the US more than 3000 e-commerce companies went bust shortly after year 2000. Again the media were quick to report on the 'dotcom bust' later referring to the whole period as the 'dotcom bubble'. In reality the period between 2000 and 2003 saw investment double to \$200 billion in the US (Schneider, 2011, p. 9) as acquisition companies merged failing online businesses and implemented proper business models.

The current state of e-commerce is more stable. Many of the most successful, exclusively online retailers (Amazon, EBay etc.), are still very successful while most large traditional retailers also have a e-commerce arm to their business. E-commerce for many businesses accounts for an important part of their sales. According to survey by Cisco Research and Economics online sales will increase 13,5 % each year until 2015 reaching a global total of \$1,4 trillion (Enright, 2011). This rate of growth is especially impressive at a time when global economic growth (real GDP growth) is barely forecast to reach 3,3% by 2014 (World Bank, 2011).

The research question

In the light of these figures it is reasonable to suggest that a web developer will at some stage work on building e-commerce applications. However there are alarming statistics relating to factors which greatly reduce the amount of sales that are generated through e-commerce sites. One of the biggest factors contributing to this loss of sales is poor e-commerce site implementation. There are a number of a number of research sources analysing the concept of 'shopping cart abandonment' and the possible causes. Ac-

cording to research by the Baymard Institute, an independent web research organisation, 60% of orders initiated on e-commerce sites get abandoned before payment is processed (Baymard Institute, 2011). In the case of an exclusively online company, more than half of the company's revenue is therefore lost due to, amongst other reasons, poor e-commerce site design. This may sound a bold statement, but consider that a customer in most cases has an intention or wish to make a purchase, evidenced by the time spent logging in, finding the item, and placing the item in the shopping cart. After which time, a percentage no doubt simply change their minds and abandon their purchase, but the vast majority give up for a host of other reasons, which are caused by poor site usability. A proportion of these reasons will be due specifically to poor payment page, or checkout implementation.

On this basis this study seeks to address the knowledge gap many web application developers clearly experience in developing e-commerce websites. This study focuses on the more advanced types of card payment integrations, as opposed to readymade solutions that are more popular, but in using them, much of the understanding of the process, security, and responsibility involved is taken away from the developer. This study therefore focuses specifically on the fundamental components of how to accept card payments through an e-commerce website.

Addressing the research question

The subject of card payment page implementation can be logically broken down into three distinct areas. Whilst there is some overlap many of the issues relating the understanding and implementation of payment page functionality fall into one of these three categories. The three categories into which this study is broken down are therefore; The Big Picture of the e-commerce environment, Front-End Implementation (development of the actual payment page) and Back-End Implementation (development of web server code and configuration).

The Big Picture chapter relates therefore to broad macro level issues relating to the operating environment of an e-commerce site. For instance understanding in broad terms the process of a card payment is important for a developer. Also important is; an understanding of the organisations needed in order to process payments, accounts that

need setting up by the merchant, choosing appropriate payment processors or payment gateway and other issues. These are all issues about which you, as a developer, would potentially need to provide advice to your client. This chapter explains why, and also focuses on studying the different services offered by two different payment gateway companies. Two advanced APIs (Application Programming Interfaces) will be chosen for further study in the following two chapters.

For all e-commerce web applications adhering to financial service industry regulations is also a necessity. Whilst many readymade solutions take care of this on behalf of the merchant, some do not, and using the more advanced API's requires that the developer ensures that, for example, personal information is kept safe. So depending on the type of payment page implementation requested by a potential client, the developer may be responsible for ensuring that some or all regulations are met. Most developers do not have a legal background and find it difficult or frustrating to understand the issues of regulatory compliance (msdn, 2006). It is therefore vital that developers have a practical understanding of what implications these regulations have on an e-commerce site development. The Big Picture chapter will also cover the security issues that need to be addressed as part of regulatory compliance, but also in order to ensure the integrity of the website. Security threats and ways to combat them will therefore be analysed.

The Front End Implementation chapter looks at the implementation of the payment page HTML and JavaScript. Front-end security measures such as HTML form validation scripts will be presented. The section looks at issues relating to checkout page usability and issues of consumer confidence. Related to this, factors that causes users to decide not to click the button to pay for their order and what can be done to prevent this in relation to form design will also be analysed. The chapter also looks at the extent to which users themselves understand the security threats they are faced with when shopping online and looks at what can be done to increase the awareness of security issues.

The Back End Implementation chapter looks at the implementation of server side code. The chapter focuses on presenting example code for implementing API calls to

make a credit card payment using the payment gateways and APIs chosen in the Big Picture chapter of the study. Necessary server settings relating to the security of the site will also be presented for two web servers commonly used for hosting ASP.NET and PHP websites, Apache 2.4.2 and IIS 7 for Window Server 2008.

C# ASP.NET and PHP have both been chosen to provide a contrasting picture between code implementations, but mainly to increase the potential readership of the study. PHP has been chosen as a popular open source language for writing web applications, C# ASP.NET has been chosen to serve those more familiar with Microsoft programming environments. To be able to differentiate between sections of code relating to PHP and sections of code relating to ASP.NET the following colour coding scheme will be used throughout the study;

PHP code	ASP.NET code
----------	--------------

Aims and scope of the study

E-commerce site implementation is an extensive area, too large to cover in its entirety, so this study focuses on arguably the most important element of the process, the transfer of money from the customer's bank account to the merchant's bank account. Strangely this critical part of the process is often brushed over in many e-commerce books, but this study aims to provide detailed, but accessible, information to web application developers about all aspects of implementing card payment functionality. The study is an exclusively secondary research paper which aims to pull together information from many different sources into one concise and easy to read document, which can be used as a guide for further study or of course as a guide to implementing payment functionality to a website. The information required to understand and develop a card payment page would involve many sources, finding and studying these sources would require a lot of time and patience. So this study is aimed at busy developers who need the important facts presented in a concise way. Primary research is not a part of this study, as the principal aim of the study is to provide a guide, rather than build a research document. In addition to these aspects what is included in the scope of the study is detailed in the previous pages.

Regarding what will not be covered; comprehensive instructions on building a workable PHP or ASP.NET website will not be covered, only code will be provided for payment page event handling. Instructions on deploying a web application to Apache or IIS will not be given. CSS code implementations will not be included, as these would be entirely subjective to the individual branding of the company selling through the site. Database security issues are out of scope as the decision to save sensitive data is a wider issue relating to the type of business and shopping cart implementation, rather than specifically relating to the payment page. Indeed shopping cart process analysis and implementation is a very broad subject so is also outside the scope of this study. Log-in implementation and user authentication are not covered as again these are broader decisions relating to the general website implementation. The assumption will be made throughout this study that the user has been authenticated prior to navigating to the payment page, as this is common practice.

2 The big picture of the e-commerce environment

This section aims to give an overview of broader macro issues relating to the card payment process. The topics of this section are not things an average web application developer would need to know in building a site accepting card payments, but having knowledge of them is nonetheless important. The topics of this section are summarised below, along with why they are important for a web application developer to know.

- **Process.** The main stages of the process of essentially transferring monies from one bank account to another bank account. This section covers the various agents or organisations involved and what their roles are. Knowledge of this process and the organisations involved is important as a developer may have to advise the client or customer on setting up a merchant account or opening an account with a payment gateway or payment processor(s).
- **Security.** The main threats that a web application faces will be covered, along with the standard set of security measures that are taken. Security for web applications accepting card payments is more complex than other web applications due to the fact that you have to secure not just connections to your site, but also in many cases ensure the connections from your site to the payment processors are also secure
- **Payment Gateways** are a key part of operating a web application with payment functionality, this section covers the important aspects of choosing an appropriate payment gateway, and analyses the different types of services they provide. For the purposes of the next section of the study, two payment gateways will be chosen along with two contrasting integration services. These choices will form the basis of the study in chapters three and four.

2.1 The payment process

This section of the study will summarise the transaction process of gaining authorisation for the card payment, and the subsequent settlement process, specifically the transfer of money from one bank account to another. The section also studies the various organisations involved in the process and sets out what their roles are. The role of the majority of these organisations has no bearing on the work of a web developer building a site accepting card payments, but it is important to have an understanding of how these organisations fit into the e-commerce process.

2.1.1 The parties involved

Payment gateway

This organisation is a service provider providing card authorisation services for e-commerce websites, or indeed any website accepting payments by card. Indeed many traditional ‘bricks and mortar’ business now use payment gateways instead of standard ‘point of sale’ card readers or scanners. Indeed the Payment Gateway is the equivalent of the card reader machine you are no doubt accustomed to using in most retail outlets (Wikipedia, 2012). Some well known brand names in this business sector would be PayPal and Authorize.Net. These organisations are commonly interfaced via an SSL (Secure Sockets Layer) connection. SSL security will be discussed later in the study. The payment gateway co-ordinates communication with all other parties in the process, so therefore provides a ‘one-stop’ solution for the e-commerce site, they are then the main organisation of interest to the work of the web developer.

Many payment gateways offer a number of different API’s (Application Programming Interface) (Wikipedia, 2012), each one offering different user interface experiences, for example one site may prefer to handle the entire process within their own pages, useful if the site is well trusted. Others may feel it more advantageous to transfer the customer to the payment gateway’s own site, thus adding the trust that the payment will be handled securely, if they themselves are yet to build that same level of respect and trust. Each API has its own security processes and standards, based on how differently they are implemented.

The payment gateway server is where the e-commerce site will connect to and send all details of the payment. This information is quite extensive, and of course highly confidential. It is therefore the case that most reputable websites accepting payments will connect via an SSL connection to the payment gateway. (SSL shopper, 2010), (Chan, Raymond, Tharam, & Elizabeth, 2001, p. 242)

The operations of most payment gateways are also restricted to certain countries, so it is therefore important to know from what countries your client wishes to accept payments from. (Knowledgebase, 2011)

It is not a necessity to employ the services of a payment gateway company, although that means that the e-commerce site must co-ordinate the communication between all parties themselves, which is often not an efficient and cost effective way to operate as payment gateway fees are quite small. If the merchant does not employ the services of a payment gateway they will have to implement much more rigorous security methods on their own, and as a payment gateway normally has a leased broadband connection to various payment processors, only the very largest online retailers can afford to bypass the services offered by the payment gateways. (Ragan, 2009)

Payment Processor

A payment processor is often associated with the merchant's bank. A 'Merchant' is the company selling, and most often running the e-commerce site.

The payment processor's job is to forward details of the transaction to the card network in order to gain an authorisation that the customer has enough money to pay for the goods. The processor has no role in the actual transfer of monies between banks, it is a 'middle man' which gets the authorisation code, and passes it back to the payment gateway or to the merchant e-commerce site. (Schneider, 2011, p. 422).

A payment gateway will forward a transaction to one of many payment processors

Interbank network / banking interchange

This organisation handles all communication between banks and other key financial institutions. Their role is important as they handle the actual ‘settlement’ process i.e. the transfer of the transaction amount from one bank account to another. This is done via a collection of interbank settlement accounts. The organisation essentially acts as a middleman between banks. There are national interbank networks (each country has its own way of organising the network) and also international interbank networks. Their network topology is complex and only sparse information is publically available.

(Garfinkel, 2002, p. 612)

2.1.2 Stages of an online payment

Please refer to Appendix 1 for a simplified diagram of the card payment process. This section describes in basic terms the stages of the payment process and briefly about the security methods undertaken at each. The process described below refers to making a payment on an e-commerce website. Sources used throughout the following numbered description are Garfinkel’s book Web Security Privacy and Commerce (Garfinkel, 2002, p. 613) , and online developer resources at developer.authorize.net (authorize.net, 2012).

1. Customer submission

The user initiates the payment by inputting their card details and clicking a submission button on a ‘checkout’ page. The page will be secured via SSL and web form data, including credit card information, is sent encrypted via HTTPS to the payment gateway. Many other variations exist including web service calls, but mostly all involve a simple HTTP POST method to transmit the data.

2. Payment Gateway processes transaction

The merchant is first authorized via submitting their merchant account credentials as part of the form POST. If the merchant’s log-in credentials can be authenticated the transaction details are forwarded to the payment processor’s servers, most commonly via another SSL connection.

3. Payment Processor processes transaction

The merchant's payment processor then routes the transaction to the merchant's bank (or acquiring bank to use the official term), asking for a transfer authorization.

4. Acquiring Bank processes authorization request

The merchant's bank then forwards the request to the interbank network, for the transaction in question.

5. Interbank network processes authorization request

The card interchange contacts the bank that has issued the card to the consumer to get authorization that there is enough funds in the account to fund the transaction.

6. Issuing bank processes authorization request

An authorization or non-authorization code is generated by the issuing bank.

7. Interbank network returns the response to the acquiring bank

Information about the implementation of network communications architecture in the interbank network is not readily available publically on the internet, as I would assume there are inherent security issues with this, but indications suggest that banking infrastructure uses its own set of protocols for the transfer of data in a highly secure way. This infrastructure also has no bearing at all on the work of a developer of a website accepting card payments, so is out the scope of this study.

8. Acquiring bank sends the response back to payment processor

The acquiring bank then returns the authorization or non/authorization code back to the processor who initiated the request, hence the response starts making its way all the way back through the communication chain.

9. Payment processor sends the response back to the payment gateway

Similarly then the processor returns the authorization or non-authorization code back to the payment gateway that initiated the request.

10. Payment gateway processes the response

The gateway prepares its own return values, normally some form of success or failure value and posts it back to the webpage which posted the request via HTTPS. This return value is determined not just on the basis of the authorization code but also on the basis of other checks that are performed such as fraud checks, stolen card checks and others dependant on the services the gateway provides. The gateway then, of course if the transaction is successful, records the transaction details against the merchant's user account.

11. The e-commerce site handles the response from payment gateway

The site displays a confirmation to the customer based on the authorization received from the payment gateway, usually either an order confirmation or order declined message. The site then records the order in its database to be fulfilled at a later time.

12. Settlement process

You will have no doubt noticed that as yet no money has been exchanged, this is handled in essence by a separate process known as the settlement process. Settlements are generally carried out in batches. The merchants will send a batch of transactions (the authorization codes) to their bank (the acquiring bank). The acquiring bank then sends a set of settlement requests to the interbank network which in turn sends it to the issuers' (the customers') bank. The issuer's bank account is debited the amount and the amount is held in an interbank settlement account. The acquiring bank then credits the merchants' account with the sum and subsequently withdraws this figure from the interbank settlement account.

2.2 Security threats and other risks to online payments

One of the biggest threats to the growth of online commerce remains to be concerns over the theft and subsequent misuse of personal information required by websites to process payments. (Eisen, 2009) Below is a summary of the common security issues and general risks encountered when paying online.

2.2.1 Process complexity

The online card payment process is a complex one (Garfinkel, 2002, p. 612), and that complexity brings with it inherent risks. With every transfer of a data packet from one server to another there is the risk that the information could be attained illegitimately. This is made even riskier in the sense that each organisation involved in the process described earlier is a separate entity, with its own standards of service and security, and potentially separate laws and guidelines it has to adhere to.

As a developer of an e-commerce site you may have absolute confidence in the implementation of security measures on your site, but as soon as the receiver has received the packets from your site the security of your customer's data is no longer in your hands, and could be compromised at a number of different points in the process. The only preventative measure a developer can take in this regard is to choose their payment gateway wisely.

2.2.2 Sniffer Programs

Theft of personal information such as names and addresses as well as of course card numbers and related details happens most frequently via programs that are designed to record network traffic between client and server. These programs essentially intercept and record http packet contents, and also record the source and destination IP addresses. To use an analogy it is the equivalent of tapping a phone line and listening in on a private conversation (Schneider, 2011, p. 460). Sniffer programs have many legitimate uses in regards to monitoring network traffic, but there are also used illegitimately to scan networks for packets indented for the IP addresses of vulnerable payment gateway servers, or they scan the packet contents directly.

How can this type of theft of personal and sensitive information be prevented? Encrypting the contents of http packets so that only the intended recipient can understand the contents is the most commonly employed solution. SSL, which will be discussed in the next section, is the most widely implemented way of http packet encryption.

2.2.3 Backdoors

Whilst these security threats are not specifically designed to steal personal information, and for our purposes payment card details, they are broader weaknesses commonly found in web applications, which if exploited could compromise the database a website uses, leaving a lot of personal and highly valuable data at risk of theft.

Backdoors are essentially security weaknesses that are found in, for example, e-commerce website implementations. Common weaknesses are those relating to poor user authentication relating to the site log-in or database log-in. Many developers build in, for example, test user accounts to sites to allow them easy access while developing and testing, but forget to remove them when the site is live (Schneider, 2011, p. 460). An alarming number of sites for instance can be accessed by simply using Username: Test, Password: Test, or Admin/Test, and so on. Using these forms of usernames and passwords is highly dangerous as there are many hundreds of websites also using the exact same credentials that have likely already been compromised.

Clearly this problem can be combated through more careful programming, and insisting all users are first authenticated via a valid email address, and then insisting that passwords are Strong. A strong password can be defined (Wikipedia, 2012) as

- A minimum password length of 12 to 14 characters if permitted
- Generating passwords randomly where feasible
- Avoiding passwords based on repetition, dictionary words, letter or number sequences, usernames, relative or pet names, romantic links (current or past), or biographical information (e.g., ID numbers, ancestors' names or dates).
- Including numbers, and symbols in passwords if allowed by the system
- If the system recognizes case as significant, using capital and lower

Other types of backdoor attacks include poor user input validation on web forms. As a payment screen for a website invariably involves a web form, these forms of attack are disturbingly common.

A fairly old technique, to exploit the weaknesses of the C programming language is the buffer overflow exploitation. This technique can be used on input fields that have no maximum allowable length, and allows the malicious user to input code into a text input field in a high enough volume that, when sent to the server and assigned to a variable, the memory buffers in the server are exceeded and the malicious code then spills over and starts overwriting data in the C application's stack frame, where it can then be executed as a program. Although an old exploit it is reported that many developers still build applications that can be exploited in this way. (Garfinkel, 2002, p. 398)

Luckily buffer overflow exploitations are rarely possible for applications written in C#/ASP.NET, due to variable data types being inherently restricted in terms of the amount of data that a single instance can hold. There are of course workarounds such as compiling a C# ASP.NET web application in unsafe mode and then using the C# Fixed statement.. The fixed statement can only be used in unsafe mode (msdn, 2012). The unsafe context can be added to any type or member declaration, so would be commonly seen like this;

```
unsafe static void FastCopy(byte[] src, byte[] dst, int count)
{
    // Unsafe context: can use pointers here.
}
```

(msdn , 2012)

This member declaration can then contain the fixed statement to essentially fix the position of certain variables using pointers, also the size of the buffer can be fixed using this statement. Both of these actions means that the C# garbage collector (the facility that resizes the buffer and empties it of discarded data) cannot move variables and properly manage the buffer, making buffer overflows a real danger. It is therefore im-

portant as an ASP.NET developer to be aware of it, while working on web applications.

The issue is reportedly also still possible for PHP applications, even though PHP is reputedly not vulnerable to such risks. Avoiding buffer overflow problems is one of the language's founding principles (Ballad & Ballad, 2009, p. 37), but PHP language developers, just like other ordinary developers are human, and so there are errors in the PHP language. Well known weaknesses in the past have been the potential buffer overflow attacks that can stem from the use of the `htmlspecialchars()`, `htmlspecialchars()` (Ballad & Ballad, 2009, p. 43), and the `serialize()` collection. The important take away from this is to ensure that the version of PHP on your web server is the latest and is always updated. Many developers report that simply limiting the maximum length of your text fields in web form submissions or at least validating length as part of the wider validation process is the most efficient and safest way to prevent these kinds of attacks rather than relying on the robustness of the coding language or the environment it operates in.

SQL injection is the process of forcefully inserting and then executing SQL statements into an application. (PHP, 2012) This can be the result of a successful buffer overflow attack or by submitting SQL statements to a server via a web form's input fields, i.e. textboxes. The hacker will insert characters, most commonly a termination character ';' into the textbox, followed by a second command. Consider the PHP example below, directly inserting a HTTP post variable into an SQL statement;

```
$name = param('name');  
sql_send("insert into names (name) value ('$name');");
```

(Garfinkel, 2002, p. 470)

If the user were to insert the following line into the 'name' textbox;

```
Joe Bloggs); delete from names;
```

then the DBMS would interpret this as actually 3 separate statements, one which inserts Joe Bloggs into the table 'names', one which then deletes all data in the table, and

a third which would generate a SQL syntax error (Garfinkel, 2002, p. 470). There are many other similar techniques that a hacker can use to delete or attain information from a database. MySQL, perhaps the most commonly used DBMS system with PHP, unfortunately would commit each of these statements to the database before the syntax error was found. Microsoft SQL server is a little more secure in this instance, by employing an 'all or nothing' principle, essentially checking the whole statement for syntax errors, and integrity constraint errors, before anything is committed to the database, however SQL Server has other serious weaknesses such as allowing the ability to execute operating system level command within an SQL statement (PHP, 2012). The simplest way to prevent this problem, as has been mentioned, is by fully validating the input received from users, either in a client script, or in server side code. User input should never be directly inserted into an SQL statement and then submitted without it being checked extensively for dangerous characters or anything else which is outside of a strict range of allowable characters.

To summarise most backdoor attacks can be prevented by heavily limiting what the user can submit to your web server. Happily both PHP and ASP.NET offer good native Regular Expression validation tools as part of their standard library. Indeed the .NET framework offers validation tools in the form of JavaScript, which allows the user input to be validated before being posted to the server, which is of course the most secure method. PHP offers server side functions to do the same job. The `preg_match()` PHP function is used to match a regular expression defined by the programmer to a string variable, it returns a simple Boolean value to express whether the expression matched the string. Methods for validating user input from web forms will be covered in more detail in section two of this study

2.2.4 Spoofing and DNS cache poisoning

These security threats both relate to when a website masquerades as another in order to dupe users in to believing they are accessing the 'real' site. This is an increasingly common problem where a DNS server's records are hacked and changed so that someone typing `www.mysite.com` actually get sent to another bogus IP address.

(Wikipedia, 2012)

A more direct form of spoofing is by using Javascript to alter information displayed in the browser, such as the address bar, or by replicating browser warning alerts to confuse the user into taking action which ultimately leads them to provide security related data to third parties (Garfinkel, 2002, p. 354). A common technique is using Javascript to display alerts to the user saying they need to reconnect to the website, and input their log-in details again, thus unwittingly sending their credentials to a third party server.

Although tough to stop the problem, preventative measures can include regularly clearing your browser's DNS cache, and ensuring your browser is up to date, as modern browsers are becoming better equipped to counteract such attacks.

As a developer you should consider informing the user to the potential threat and inform them what they can do about it.

2.3 Security methods employed

This section discusses the protocols used to transmit data between client and server and between different servers. This is an important part of the paper as the protocols chosen for various parts of the communication process between two network nodes governs to a large extent how secure the process is.

2.3.1 Asymmetric cryptography

This type of encryption is possibly the most widely used for securing internet traffic (i.e. HTTP packet transfer), and is the encryption method used in arguably the most commonly used secure network protocol, SSL (discussed later).

Unlike symmetric key encryption, asymmetric encryption involves two keys, one to encrypt, one to decrypt (Garfinkel, 2002, p. 51), no one key will do both. The two keys are mathematically linked, although in most cases a knowledge of one key does not make it possible to derive the other. This is due to a mathematical phenomenon, which is out of the scope of this study, but the benefit to securing network communication is

that the transfer of the public key does not need itself to be secure, and the key's value is not secret, hence the term 'public' key.

The process in very basic terms can be summarized as anyone who wishes to be sent secure data needs to produce a pair of keys, one public, and one private kept by the producer of the keys. The public key is then made available to anyone wanting to send data to the key producer. The sender encrypts the plain text using the public key, which is sent and subsequently decrypted by the private key.

Digital signatures use the method as a way of authenticating the sender (Wikipedia, 2012), as a digital signature is written with the private key, hence the receiver knows that it has come from the assumed source, and that the public key has definitely come from that sender.

2.3.2 Secure Sockets Layer (SSL)

I will first give an overview of how SSL works and then discuss how and where it is employed as part of the online payments process.

SSL is a program layer protocol located between the Internet's Hypertext Transfer Protocol (HTTP) and Transport Control Protocol (TCP) layers (Cusack, 2000). SSL has recently been superseded by Transport Layer Security, but as TLS is based on SSL, and the fact that SSL is the most commonly used term and protocol, this is what I will focus on. The term 'sockets' is used to refer to the process of a client computer communicating with a server computer over a network, or even within the same computer.

SSL uses Public and Private Key Encryption (so asymmetric cryptography) via RSA to secure data packet contents (Wikipedia, 2012), (Garfinkel, 2002, p. 113), and also includes a digital certificate – an important part of the security process for online payments. RSA is the main algorithm method for Asymmetric cryptography (Public and Private Key Encryption). RSA is a three stage process involving key generation, encryption and decryption. I won't discuss the mathematical basis for the algorithm but an explanation of asymmetric cryptography and digital certificates is important.

A website developer or administrator (or any user with root access to a web server) can enable and configure SSL on the server (there are a variety of programs to do this, dependant on the web server (Windows IIS, Apache etc)). The configuration would typically involve the generation of keys, choosing the algorithm for that generation, requesting a certificate, and then choosing which web pages to use SSL for. All of these functions are typically performed by command line functions or via the web server SSL program's Graphical User Interface (GUI).

SSL is in most cases already part of your web browser's implementation, so for the average user SSL is something that just 'happens' without your knowledge. Most users will only be aware of it is when a lock icon appears on the browser's address bar when often the web address starts with https://. As a user you may also receive a message from the Secure HTTP page request that 'the digital certificate provided by this site cannot be trusted', and then you will be prompted to add an exception for it, or leave the site. This happens because the web server's SSL software has been set up to sign its own Digital Certificate rather than involve PKI (Public Key Infrastructure) bodies, or more specifically a CA (Certificate Authority), but also an RA (Registration Authority) (Garfinkel, 2002, p. 160) (essentially a central registry database of all certificates issued) and possibly also a Digital Certificate Management System. Whilst this warning might be fine when the user is being asked for minimal personal information it drastically harms the business of the website accepting payments if the user receives warnings about the authentication of the website as soon as they navigate to the payments page. This is something which must be prevented, I will talk about this when discussing about Digital Certificates.

Most bodies in the payment process already discussed will implement an SSL connection to their servers, so for an organisation like a payment gateway, a Digital Certificate Management System to manage the large number of keys and certificates that it issues itself but also that it holds to communicate with payment processors and other companies is vital. Verisign, one of the more well known CA's is reported to have the most digital certificates, and implements one a comprehensive system to manage them (Garfinkel, 2002, p. 182)

2.3.3 Digital certificates

When making a payment online it is important to the consumer that the website can be trusted. Web development standards dictate that a payment page must be implemented with an SSL connection and must also involve a Certificate Authority in signing its digital certificates. (PCI Security Standards Council, 2010, p. 35), (authorize.net, 2012, p.

2) But firstly what is a digital certificate?

A digital certificate basically allows a web browser or other client software to verify that the sender of an HTTP packet is indeed sent from the expected source, so it provides a method of authentication. The certificate contains information about the company running the web site, the expiration dates of the certificate and the public key (required to communicate with the company's software via the SSL connection). A Certificate Authority (CA) provides an additional level of trust as the client software contacts the CA separately to verify the digital certificate, and so can authenticate the website accepting payments, this method also aims to counteract so called 'man in the middle' attacks. (Garfinkel, 2002, p. 168)

However, if a website opts to purchase a basic SSL certificate, offering only Domain Verification, then in reality the CA is doing very little in the way of authentication, essentially confirming that the `www.onlinestore.com` domain name belongs to a company named Online Store Inc.

2.3.4 Certificate Authorities and requesting a certificate

As mentioned most web browsers are configured to inform the user when a digital certificate has been self signed, so it is vital that a proper certificate is attained for any website accepting card payments. Many Certificate Authorities (CAs) are trusted organisations and indeed offer a logo to be placed on the e-commerce website payment page to inform users that the site is trusted, although I would question how many users understand how this logo ensures the security of the site. Please refer to Appendix 2

for a selection of the most commonly used and sought after CA logos. Of course a CA is only as useful as how trusted they are. There are differences as well in the level of authentication that each provides.

Many of these differences in level of service are reflected in the prices for having your site verified by a CA. Because of this cost it is generally only large online retailers and financial institutions such as banks and credit card companies that subscribe to the highest levels of verification, as it has become the industry standard. These higher levels of verification are commonly called EV-SSL certificates or Extended Validation SSL. This process requests more information from the company requesting the certificate and then performs regular checks to ensure they are still who they say they are, often by authenticating their physical presence (business address, phone number etc) (Wikipedia, 2012), as well as basics such as domain name ownership. Each CA offers slightly different services at slightly different costs.

To attain a basic digital certificate the set of details you would need to provide are as follows:

Country Name [C] A capitalized two-letter country code for you or your organization. For example, UK, RO, DE, etc.

State or Province Name [SP] The name of the state or province for you or your organization in full textual form. For example, Quebec, Gaza, Alaska, etc.

Locality Name [L] The name of the city or town for you or your organization in full textual form. For example Edinburgh, Stockholm, Ulooloo, etc.

Organization Name [O] The name of your organization/company/domain in full textual form. **Organizational Unit Name [OU]** The name of the organization's department in full textual form.

Common Name [CN] Regardless of the type of certificate you intend to obtain or the purpose for which you intend to use it, this field is of critical importance. This field must contain the exact domain name for which it will be used including the sub domain part. If the sub domain does not match then it will still be possible to use the certificate, but another one of those large warning messages will be presented to the user. So if you have `www.domainname.com` as the CN in the certificate and a user connects to `domainname.com` (without the `www`). then the domain names will be considered as non-matching and the warning will be issued; however, if the main domain part of the domain name does not match then it is unlikely that Apache will even start at all. So make certain that you specify the correct sub domain and that the domain name is completely free of typos.

Email Address[EMAIL] Even in its abbreviated form it's pretty obvious what this field is for. Assuming that they exist, good choices would be `webmaster@domainname.com` or `admin@domainname.com`.

(Webopedia, 2012)

2.3.5 PCI Standards

Payment Card Industry Security Standards Council (PCI SCC) is an organization responsible for publishing a set of security standards for businesses handling card payment transactions. The organization's work has come to focus more and more on the necessary standards needed for online card transactions. Their principle standards are published as PCI DSS (Payment Card Industry Security Standard). The standards document is currently version at 2.0 (PCI Security Standards Council, 2010). A summary of the requirements to meet the standards is given below;

Build and Maintain a Secure Network

Requirement 1: Install and maintain a firewall configuration to protect cardholder data

Requirement 2: Do not use vendor-supplied defaults for system passwords and other security parameters

Protect Cardholder Data

Requirement 3: Protect stored cardholder data

Requirement 4: Encrypt transmission of cardholder data across open, public networks

Maintain a Vulnerability Management Program

Requirement 5: Use and regularly update anti-virus software or programs

Requirement 6: Develop and maintain secure systems and applications

Implement Strong Access Control Measures

Requirement 7: Restrict access to cardholder data by business need to know

Requirement 8: Assign a unique ID to each person with computer access

Requirement 9: Restrict physical access to cardholder data

Regularly Monitor and Test Networks

Requirement 10: Track and monitor all access to network resources and cardholder data.

(PCI Security Standards Council, 2010)

Of course these standards relate to card payment processing service organisations specifically but many of these standards are relevant also for websites accepting card payments. Especially if, for example, the website chooses to store its customers card details on in its own database. A payment gateway company would normally be the one storing card details, to reduce the unnecessary transfer of valuable data across the internet, but many e-commerce sites also allow users to store their credit card details, for the sake of convenience. Therefore these standard requirements are given here as

they provide a useful checklist for any developer building a website including card payment functionality.

2.3.6 Payment Application Best Practice (PABP)

Whilst the PCI DSS set of standards is important for e-commerce websites as well as those organisations who are subject to financial regulations, the PABP set of standards is specifically tailored for e-commerce website operators and developers. The set of best practices is developed by Visa and includes 13 security standards for payment applications (which obviously includes any websites accepting card payments). Whilst these best practices are not yet required by the payments industry, it is anticipated that it will be in the near future (Authorize.Net, 2008, p. 4). The full set of best practices is given as in Appendix 4.

2.4 Payment gateway integrations

As discussed earlier a payment gateway essentially provides a basket of services to e-commerce sites which would otherwise be financially unviable for the site to develop alone. These include merchant account services, payment processor communications and fraud protection (Wikipedia, 2012), (Schneider, 2011, p. 498). It is possible to implement these functions independently, but for the purposes of this study we focus on the most expedient and cost effective way to implement these necessary functions, for a small or medium sized business or organisation. Hence employing the ‘one-stop-shop’ services of a payment gateway is the choice to make.

This section discusses the implementation options generally offered by payment gateways and focuses on the options offered by two of the market leading payment gateway businesses. Chapters three and four of this study will go on to provide example implementations of two contrasting options offered by the gateways, i.e. in real terms how would we integrate our website payments page with the payment gateway servers.

Based on my independent online research I can summarise that payment gateway integrations are implemented in two main ways. The first option is implemented simply by a secure payment form hosted by the payment gateway's own web server. This is most cases means the customer is transferred to the gateway's website to complete the transaction. The second implementation is by using Application Programming Interfaces (APIs) offered by the payment gateway company. In this case, the user never leaves the website where the user places the order so they have the perception that the process is handled entirely by the e-commerce website. An Application Programming Interface is essentially an imported set of routines/methods/functions, data structures and object classes (Wikipedia, 2012). They are called in just the same way as other classes or libraries that you may use in your code.

Although these are two distinctly identifiable processes, in reality the real options offered are a mixture of the two. Web API's for example basically involve a process of posting an HTTP request, like any other CGI formatted URL to the gateway server, giving a list of specified variable values (credit card number, amount, expiry date, etc), and then handling the response, which is again specified in documentation provided by the payment gateway, such as Authorize. Net's developer guideline document (Authorize.net, 2012). Whilst this process is not a traditional form of API, it is extensively used and mostly simply called an API.

Web service protocols are another common way of making an API available to web developers, common web service implementations are SOAP and REST. SOAP or Simple Object Access Protocol, is a simple process that uses XML to transfer messages between servers. It relies on the parsing of the XML response. REST is newer and does not rely on XML parsing of the response message. Whilst this may seem complicated in reality it is not. In all instances discussed above the process is very similar and summarised in simple terms below;

1. The payment form submit button is clicked
2. The HTTP request URL is built as a string including all necessary required by the payment gateway, from the details submitted by the user
3. The HTTP request is submitted to the payment gateway.

4. The HTTP response is handled by the website, most commonly by parsing an XML file in some standard format.

Summarised from Authorize.net developer documentation (Authorize.Net, 2012)

2.4.1 Factors influencing payment gateway choice

Many gateways offer easier implementations for certain shopping cart modules. Certainly for any e-commerce site selling a range of products, a shopping cart is a necessary addition to the site. There are many readymade shopping carts where the code can be inserting into your PHP scripts or ASP.NET code behind files, or included simply as a complete module with some CSS customization. Shopping cart implementation is out of the scope of this study, which focuses specifically on the payment process, but it is accepted that shopping cart implementation is important and might also influence your choice of payment gateway

The factors discussed below to the end of section 2.4.1 are based on an article by Vangie Beal (Beal, 2012).

Card and currency support is an important factor. You want to be able to offer your customers as much choice as possible, particularly if you are aiming to reach a global market. You need to think carefully about your customer profile and determine which cards you need to provide support for, and which currencies you need to accept payment in. It is normal that a good payment gateway also offers other forms of online payment as well as card payments.

Compliance with standards is also vital. The standard set of compliances, PCI-DSS, has been previously covered. Any reputable payment gateway must ensure they implement these guidelines. Some card issuers publish lists of card payment service organisations that currently have been checked as being compliant. The only problem with security standards is that they are only a snapshot of the business's compliance with the guidelines, they can of course choose to not comply just as soon as they have passed the inspection. (Visa, 2012). Companies are checked against the standards once per

year, so perhaps some record of consistent compliance would be more useful, but as it stands this is the best we have.

Understand fully the charges the company uses. Different companies charge differently for different services. For example, if your client business offers a generous money back guarantee if not completely satisfied, then you need to ensure you pick a payment gateway that doesn't over-charge for refund payments (or chargebacks). Or for example if your client offers subscription services, you need to make sure that the gateway company you choose has to capacity to offer ARB services or Automated Recurring Billing. Not all offer this.

What kind of customer support they offer is another important factor. Do they offer phone support, what hours do they provide this support. Also what level of information do they provide to their customers regarding transaction details, histories, and account balances?

Finally all reputable payment gateways offer fraud protection services, so you should make yourself aware of what they do to protect your client's customers, and in turn your client's business. As a minimum you should ensure the gateway supports the industry standard AVS (address verifications service). If your payment gateway is a United States based company, they should list AVS as part of their services. Other countries offer similar schemes whereby the consumer has to provide the billing address registered against the card in order to complete the transaction.

2.4.2 Transaction types

It is important that the developer considers in advance what type or types of transaction are needed for the e-commerce site. This is because the decision will affect greatly how the payment process is implemented and therefore will affect server side code enormously. The decision around what transaction type or types to choose is generally made according to what types of goods or services the business operating the e-commerce site sells. (Paypal, 2009, p. 23) I will focus on the two main types of transaction as all others are a variation of these two.

The first type is known as Sale Authorization and Settlement. This process attains the authorization and immediately flags the transaction for settlement (Paypal, 2009, p. 28). This transaction type is most suited for service businesses or where the customer essentially receives the goods or services straight away. The second type of transaction is a two stage transaction known as Authorization Only (Paypal, 2009, p. 29). These transactions attain an authorization code, and then at some point later a Delayed Capture transaction is initiated to then flag the transaction for settlement. The Delayed Capture transaction can be for the same amount as the original transaction or less (in the case of split shipment orders). These transactions are suited to e-commerce sites selling physical goods where the transaction is not flagged for settlement until the goods have at least been dispatched to the customer. This transaction type is very common for business to business (B2B) firms. Business to consumer firms (B2C) often operate 1 stage transactions (i.e. your card is charged soon after your order) even if they are selling physical goods, meaning the customers card is charged long before they receive the goods. For the purposes of Sections 2 and 3 of this study I will focus on the 1 stage transaction type i.e. Sale Authorization and Settlement.

2.4.3 Google Checkout and Amazon Payments

I cover these services first, as they will not be covered in detail in later chapters of this study with regards to analysing API integrations for ASP.NET and PHP as they are not a payment gateway in the full sense of the term. This is principally because they only offer the type of integration API where the user is transferred to the Google checkout or Amazon website, for payment processing (Amazon, 2010), (Google, 2012) This is fine for businesses that do not have, or do not plan to have, a trusted brand name, but is limited for businesses who wish to present themselves as serious and responsible sellers of goods and services.

People have become familiar with the Google brand and under certain circumstances their integration solution, being very simple, will suit the situation. Principally though, due to this simplicity the e-commerce site developer is not required to handle credit card details at all, indeed they are not required to handle even any code, and hence it is

not studied further here. Google's integration solutions all comprise of a collection of button designs, and corresponding html code for a simple html form, which is posted over a non secured HTTP connection to the Google Checkout site, to which the user is also transferred (Google, 2012). Google also have many shopping cart integrations, but still with ultimately the same process, everything being handled by pages served from Google servers. As mentioned shopping cart implementation is not a part of this study, for this reason also it is not covered further.

Google requires consumers to have an account with them in order they can use the checkout services they provide. That obviously requires users, who have already logged in to the e-commerce site, to be transferred to Google's site where they are required to log-in again to complete the payment. This system is generally thought of as being bad for an e-commerce site, as there are many instances of 'cart abandonment' due to it, meaning customers never complete the transaction. This might be due to many reasons; they cannot remember their Google log-on credentials, they feel concerned and/or confused that they have left the original site, they feel frustrated with the complexity of the process, there is a technical error being transferred to or from Google's site, and other reasons.

From a developer point of view it is worth noting in relation to the wider development issues of your sites, that perhaps providing Google checkout as an additional payment option would be beneficial. The reason for this is that there are reportedly significant Search Engine Optimization (SEO) benefits to having the Google checkout button on the site. Google will essentially add an 'accepts Google Checkout' image against your site, and the listing will appear nearer the top of search results. Whilst no documentation could be found to support this, there are many anecdotal developer blog posts on this issue, stating that there are SEO benefits.

2.4.4 PayPal

Paypal, owned by Ebay, has long provided, very successfully, an easy solution for two individuals to exchange money, with nothing more than knowing the receiver's email address. This process lent itself very well to C2C (Consumer to Consumer) environ-

ments such as Ebay and other small businesses, and private entrepreneurs. Again they are known for their very simple integrations that are essentially the same as Google Checkout, both in process, functionality, and pricing. Based on my research of the Paypal developer's website these integrations take the form of 'hosted buttons' which again transfer basic order details to the Paypal server through an HTTP Post (discussed in the next chapter). The button also contains re-direction code, to transfer the customer to Paypal where they are prompted to log-in again to their Paypal account. For the purposes of this study, as mentioned, these integrations will not be studied further. However Paypal have expanded their services to include a range of other full payment gateway services, including card payment processing, fraud protection and better security. These services are branded as Payflow integrations

Paypal Payflow is a package of solutions, ranging from more simple, to quite complex. Payflow Link is a useful integration, which still transfers the user to Paypal servers, but the design of the page they are taken to matches that of the original site. Essentially developers can adapt payment forms to suit their needs, and alter page HTML and CSS coding to match their site. This interesting solution reduces the security implications for the e-commerce site, whilst retaining more customers as all payment processing appears to happen on the same website. Paypal ensure all communication is through SSL (Paypal, 2012, s. 2), the only down side is that the SLL certificate will also state Paypal as the owner, as opposed to the e-commerce site, which may alarm some consumers. Also Payflow abides by PCI guidelines.

Because Payflow Link is still entirely an HTML solution, it requires no server-side code, so will not be covered further in this study. Payflow Pro however is a full API, which is published as an SDK (Software Development Kit). SDK's are offered for .NET, and Java, and a raw HTTPS interface is also offered for other languages, or for the purposes of building a custom API (Paypal, 2009, s. 13). Chapter four (Back-end Implementation) of this study will go into detail about how to implement this service for an ASP.NET website, and a PHP website.

2.4.5 Authorize.net

This payment gateway is one of the most widely used for e-commerce websites based in the United States. The company aims to deliver a ‘transparent’ service, essentially meaning that the customer will in most instances remain on the e-commerce site, and will be mostly unaware of Authorize.com involvement in the process. This is an interesting business model in comparison to Paypal who push their company branding in all their implementations. Based on my research of the Authorize.Net developer’s website a summary of the main integration options they offer is below;

Simple Checkout

Similar to Google Checkout and Paypal’s hosted buttons, this option requires no Server-code programming, and is a simple redirect to Authorize.com servers where the customer inputs their card details. Upon successful completion of the transaction they are then transferred back to the original site.

Direct Post Method (DPM)

This method is a HTTP post handled by the e-commerce web server. The HTTP post can be over an SSL connection (this can be developer defined) but is commonly posted using a standard HTTP request. The developer of the e-commerce site will need to implement the payment page (although the payment form fields are generated automatically by the API, additional fields can be added by the developer), handle the security of personal data, and handle response information from the Authorize.com servers.

Server Integration Method (SIM)

This method is intended for those e-commerce companies that do not want to purchase and manage an SSL certificate for their site. Similarly to Paypal’s Payflow Link integration, Authorize.Net’s servers will serve the payment form, which can be totally customized to match the e-commerce site’s styling, they also handle the processing, and secure connections.

Advanced Integration Method (AIM)

This is the most customizable method of implementation, which relies on the e-commerce site developer to implement from scratch the payment form, required fields, optional fields, implementing a secure SSL connection, handling card data and user input validation. As this requires development in most of the areas discussed in this study, it will be studied further in the following chapters.

The Authorize.Net website: developer.authorize.net has been used as a source throughout section 2.4.5 (Authorize.Net, 2012)

2.5 Chapter Summary

To summarise I will present below a list of recommendations aimed at developers in order for them to understand more about what is required to ensure the effective security of an e-commerce site card payment process, and inform them in order that they are able to decide what additional services their client may need to register for . This is not a comprehensive list but aims to cover the main issues:

Security threats and other risks to online payments

- Fully validate all text fields, preferably with regular expressions, on the web form so that only allowable values get posted to the server.
- Only record card numbers in the database if absolutely needed for your site's user experience, and do not transfer card numbers or other personal data across the internet unless it is required.
- Ensure your site implements user log ins, that are strongly authenticated.
- Implement also a database of user accounts that employ a minimal rights policy

Standard security methods employed

- Enable the SSL software on your web server and implement an SSL connection for at least the payment form page.
- Familiarize yourself with security standards and ensure they are implemented as necessary for your site, as well as being implemented as part of your chosen payment gateway's service. For instance PCI DSS and PABP standards.

- Buy a subscription to a Certificate Authority, or perhaps the payment gateway provides this, dependant on the service you subscribe to. This will make your site harder to spoof, and provide customer confidence.

Payment gateway implementations

- Look to choose a reputable payment gateway that meets your requirements relating to services, and implement their suggested interface.
- Consider the transaction types you will need according to the type of business to be operated through the site i.e. 1 stage transactions or 2 stage transactions.

3 Front end implementation

This section looks at all issues relating to the user interface, basically the payment web page itself, and all related client script issues. As previously mentioned page design, is not covered as part of this study, principally as the level of variation is large, and is depending on individual business needs, and corporate branding. For the purposes of this study I present the more technical aspects of implementing a payment page, or more specifically a payment form. The topics covered by this section are summarized below;

- HTML forms. This part will look at the mechanics behind a web form, which is the most important aspect of payment page integration. Understanding how HTML forms work is vital to a developer.
- User input validation. This part will present a client side Javascript validation script for a payment form. As previously discussed, validation in a client script is an important security measure, and must be implemented in the absence of validation assistance offered by the API used.
- Payment page usability. This part analyses the elements of general page design and technical implementation that affect usability and therefore impact on how easy the page is to use.
- Informing the user. In the previous section it was highlighted that there are potential security threats posed by vulnerable browser versions, browser settings, and a simple lack of understanding of web security. Informing the user about security issues is therefore important.

3.1 HTML Forms

The HTML `<form>` tag is what facilitates users to be able to interact with a website. More specifically most instances where a user needs to submit information to the hosting web server are handled by implementing the necessary user input controls (for simple HTML these are the various forms of `<input>` tags available) inside the `<form>` `</form>` tags. Common uses of an HTML form include submitting a contact form, submitting booking or reservation information, registering to a website, or for the purposes of this study submitting credit or debit card details to make a payment.

Users generally "complete" a form by modifying its controls (entering text, selecting menu items, etc.), before submitting the form to an agent for processing, by 'agent' we mean the web server (World Wide Web Consortium, 2012)

3.1.1 Form attributes

The 'action' attribute

The list of attributes specified by W3C for the <form> tag includes only one which is required, namely the 'action' attribution. This attribute specifies the server side form handler (World Wide Web Consortium, 2012) . For the languages we are focusing on, a PHP form handler value would be the .php file handling the user input such as paymentSubmitted.php, for ASP.NET the rendered action value is typically the .aspx page handling the form event. By 'rendered value' I refer to how the code looks when viewing rendered page source code. As an ASP.NET programmer you wouldn't typically need to specify these values, as they are automatically generated.

These values will submit user input to the server without any validation, something which is not recommended as discussed in the previous section. If client side validation is needed the action attribute needs to refer to a JavaScript function to validate the input and then, if it passes validation, the function will submit the form. You can see examples of this from the code given later.

The 'method' attribute

Although this method is not required, for the purposes of this study it will need to be specified for the PHP implementation of the payment page. There are two allowable values for the method attribute, namely 'get' and 'post'. The 'get' value is the default. When a 'get' value is used all user submitted data is appended to the URI given as the 'action' attribute value. (World Wide Web Consortium, 2012). The different fields are appended after a '?' question mark and separated with a '&' symbol. So the 'action' value in the PHP example above would change from;

```
http://www.myserver.com/paymentSubmitted.php
```

to;

```
http://ww.myserver.com/paymentSubmitted.php?cardNumber=1234123412341234&expDate=0812&amount=9.99
```

The get method, although the default should be used only when the form is idempotent (i.e., causes no side-effects) (World Wide Web Consortium, 2012). This basically means that it should be used only for simple data requests and not for requests where database records could be altered. For this reason and others it should never be used for submitting card payment details. The ‘post’ method submits user data in a less visible way, as part of the HTTP packet contents. ASP.NET submits all form data using the ‘post’ method as the default, and again it is not something an ASP.NET programmer would need to specify. Although when making an HTTP request from the code-behind file, something which is needed to submit the transaction data to the payment gateway, then it must be specified that the request must use the ‘Post’ method. This will be covered in more detail in the next chapter (Back-end Implementation).

Hidden fields

Most payment form implementations, including those required by the payment gateway APIs’ used in this study make use of hidden fields within the HTML form. Hidden fields are essentially input fields that are not rendered on the page, but whose values are submitted with the form (World Wide Web Consortium, 2012). They are an essential part of ASP.NET pages and are used for example to maintain the state of all controls after the page is posted back. Other uses include storing common data used by the server such as user credentials in order to create a user session. Hidden fields are created used the <input> tag specifying the type attribute as ‘hidden’.

3.1.2 Payment page HTML

This section presents what form fields need to be included between our <form></form> tags in order to gather all the required data needed to make a payment submission to our chosen payment gateway. I will give the HTML code for each payment gateway API case study covered in the previous chapter. This will include therefore examples of HTML forms posting directly to the payment gateway server, and HTML forms which post to a form handling page on the e-commerce web server,

from which the transaction gets posted to the gateway server. For these more complex examples the server-side code will be covered in the next chapter. As mentioned previously no CSS code will be included as this is entirely subjective to the wider design of a particular site.

Firstly I will cover HTML code for providing hosted buttons, .i.e. buttons that redirect the user to the payment gateway web server in order to make a payment. As a developer you may choose to include these buttons, in addition to providing your own payment form, in order to offer the user as many payment options as possible. In the previous section I mentioned about Google Checkout and Amazon Payments.

Google Checkout Button

The code below will give an example of a payment button which transfers the user, with basic details about their purchase, to the Google Checkout site for payment processing (Google, 2012)

```
<form method="POST" action="https://checkout.google.com/api/checkout/v2/checkoutForm/Merchant/1234567890" accept-charset="utf-8">

  <input type="hidden" name="item_name_1" value="Peanut Butter"/>
  <input type="hidden" name="item_description_1" value="Chunky peanut butter."/>
  <input type="hidden" name="item_quantity_1" value="1"/>
  <input type="hidden" name="item_price_1" value="3.99"/>
  <input type="hidden" name="item_currency_1" value="USD"/>

  <input type="hidden" name="ship_method_name_1" value="UPS Ground"/>
  <input type="hidden" name="ship_method_price_1" value="10.99"/>
  <input type="hidden" name="ship_method_currency_1" value="USD"/>

  <input type="hidden" name="tax_rate" value="0.0875"/>
  <input type="hidden" name="tax_us_state" value="NY"/>

  <input type="hidden" name="_charset_"/>

  <input type="image" name="Google Checkout" alt="Fast checkout through Google"
src="http://checkout.google.com/buttons/checkout.gif?merchant_id=1234567890&w=180&h=46&style=white&variant=text&loc=en_US"
  height="46" width="180"/>
</form>
```

Developers should consider including this button with a view to the possible SEO benefits, and customer experience benefits of offering wider payment options. Note that basic order details are required. The developer should therefore consider whether to add the relevant values to the hidden fields outlined above as part of their server

side code implementation, so the fields are populated when the customer moves to the payment page. Note also that the URL specified in the action attribute will include a merchant ID number, this means a merchant account would need setting up (at least a test account). The rendered design of this button can be seen in Appendix 5.

Amazon Payments SimplePay Button

Below is the HTML implementation of Amazon's simplest payment button, thus potentially offering a payment option to those users most comfortable using Amazon's own payment interface (Amazon, 2010).

```
<form method ="POST" action
="https://authorize.payments.amazon.com/pba/paypipeline">
<!--REQUIRED FIELDS -->
  <input type ="hidden" name ="accessKey" value ="YourAccessKeyId">
  <input type ="hidden" name ="amount" value ="USD 10">
  <input type ="hidden" name ="signature" value
="K2ryWe7s/0AHI0/PbuAveuUPksTefhmNCzDTold2VYA=">
  <input type ="hidden" name ="description" value ="pay for dinner">
  <input type ="image" src
="https://authorize.payments.amazon.com/pba/images/SLPayNowWithLogo.png" border
="0">
  <input type ="hidden" name ="signatureVersion" value ="2">
  <input type ="hidden" name ="signatureMethod" value ="HmacSHA256">
<!--NON REQUIRED FIELDS -->
  <input type ="hidden" name ="ipnUrl" value
="http://yourwebsite.com/instantpaymentnotification">
  <input type ="hidden" name ="returnUrl" value ="http://yourwebsite.com/success">
  <input type ="hidden" name ="processImmediate" value ="1">
  <input type ="hidden" name ="cobrandingStyle" value ="logo">
  <input type ="hidden" name ="abandonUrl" value
="http://yourwebsite.com/abandon">
  <input type ="hidden" name ="referenceId" value ="MyTransaction-001">
  <input type ="hidden" name ="immediateReturn" value ="1">
  <input type ="hidden" name ="collectShippingAddress" value ="0">
</form>
```

I have indicated above the required and non required fields on the form. From a user experience perspective a developer should consider implementing also the non required fields. Please refer to the Amazon's development document (Amazon, 2010) for an explanation of what these fields hope to achieve. Similarly to the Google Checkout example the rendered design of this button is given in Appendix 5.

Payment form implementation

Now we move onto the HTML implementations for the payment form itself, including user input fields, something not included in the two button forms above. The form examples given below are relevant for both the PayPal PayflowPro and Authorize.net

Advanced Integration Method chosen as the payment gateway providers for this study. The fields given here are the industry standard set of data required from the site user in order to submit a transaction. Indeed the bare minimum fields required to submit a transaction are actually only credit card number and expiry date, but this entails essentially no security checks, so it is highly recommended that full name and address is asked for as well as the CVV2 (Card Verification Value) number (where this number is found varies between card issuers but is most commonly the last 3 or 4 digits from the security number printed on the back of the card) (Wikipedia, 2012). In a practical implementation of an e-commerce website you may choose as a developer to allow a user to save previously used card details, but as this is out of the scope of this study we will assume the user has to submit their card details for each transaction. It is important to note that PCI compliance demands that if the site is storing card details it must not under any circumstances store CVV numbers. (PCI Security Standards Council, 2010, p. 8) That way if the site's database is compromised and card numbers are stolen, not having the CVV number dramatically limits the places that a thief can try and make purchases using the card number.

There are a number of other required fields (or more correctly required parameters), many similar to those hidden fields given in the hosted button examples shown previously, but as the HTTP request to the payment gateway server happens in server side code, these additional parameters are not given here as HTML, they will be given in the next chapter when discussing server side implementation. For security reasons also data held in hidden fields should only be required data for such reasons as maintaining user session information.

```
<!--ASP.NET-->
<form id="paymentForm" runat="server">

    <!--Card choices here are decided by your merchant account and what cards
    your account can accept! -->
    <asp:DropDownList ID="ddlCardType" runat="server">
        <asp:ListItem>Visa</asp:ListItem>
        <asp:ListItem>Mastercard</asp:ListItem>
        <asp:ListItem>American Express</asp:ListItem>
    </asp:DropDownList>

    <asp:Label ID="lbCardNumber" runat="server" Text="card number
    :"></asp:Label>
    <asp:TextBox ID="tbCardNumber" runat="server" MaxLength="16"></asp:TextBox>
    <asp:RegularExpressionValidator
    ID="revCardNumber" runat="server" ErrorMessage="Please check your card num-
    ber!"
```

```

ValidationExpression="^(?:4[0-9]{12}(?:[0-9]{3})?|5[1-5][0-9]{14}|6(?:011|5[0-9][0-9])[0-9]{12}|3[47][0-9]{13}|3(?:0[0-5]||[68][0-9])[0-9]{11}|(?:2131|1800|35\d{3})\d{11})$"
ControlToValidate="tbCardNumber">
</asp:RegularExpressionValidator>
<br />
<asp:Label ID="lbExpires" runat="server" Text="expires : "></asp:Label>
<!--Include obviously all months to '12'-->
<asp:DropDownList ID="ddlExpiryMonth" runat="server">
  <asp:ListItem>01</asp:ListItem>
  <asp:ListItem>02</asp:ListItem>
</asp:DropDownList>
<asp:DropDownList ID="ddlExpiryYear" runat="server">
  <asp:ListItem>2012</asp:ListItem>
  <asp:ListItem>2013</asp:ListItem>
  <asp:ListItem>2014</asp:ListItem>
  <asp:ListItem>2015</asp:ListItem>
</asp:DropDownList>

<asp:Label ID="lbCVV2" runat="server" Text="CVV2 number"></asp:Label>
<asp:TextBox ID="tbCVV2" runat="server" MaxLength="4"
Width="44px"></asp:TextBox>
<asp:RequiredFieldValidator ID="rfvCVV" runat="server" ErrorMessage="CVV
number is required!"
ControlToValidate="tbCvV2"></asp:RequiredFieldValidator>
<br />
<asp:Label ID="lbFirstName" runat="server" Text="first name: "></asp:Label>
<asp:TextBox ID="tbFirstName" runat="server" MaxLength="30"></asp:TextBox>
<asp:RequiredFieldValidator ID="rfvFirstName" runat="server"
ErrorMessage="First name is required!"
ControlToValidate="tbfirstName"></asp:RequiredFieldValidator>

<asp:Label ID="lbLastName" runat="server" Text="last name: "></asp:Label>
<asp:TextBox ID="tblastName" runat="server" MaxLength="30"></asp:TextBox>
<asp:RequiredFieldValidator ID="rfvLastName" runat="server"
ErrorMessage="Last name is required!"
ControlToValidate="tblastName"></asp:RequiredFieldValidator>
<br />
<asp:Label ID="lbStreetAddress" runat="server" Text="street address:
"></asp:Label>
<asp:TextBox ID="tbStreetAddress" runat="server"
MaxLength="30"></asp:TextBox>
<asp:RequiredFieldValidator ID="rfvStreetAddress" runat="server"
ErrorMessage="Street address is required!"
ControlToValidate="tbStreetAddress"></asp:RequiredFieldValidator>

<asp:Label ID="lbPostCode" runat="server" Text="ZIP/postal code:
"></asp:Label>
<asp:TextBox ID="tbPostCode" runat="server" MaxLength="9"></asp:TextBox>
<asp:RequiredFieldValidator ID="rfvPostCode" runat="server"
ErrorMessage="ZIP/Postal Code is required!"
ControlToValidate="tbPostCode"></asp:RequiredFieldValidator>
<br />
<asp:Button ID="btPay" runat="server" Text="Pay" CausesValidation="True" />
</form>

```

For the sake of readability, only the required fields have been included. The example above also assumes that data relating to the order, most crucially the amount to be charged to the card, is already held in the e-commerce site's database, and would normally be displayed somewhere on the page, for the user to review as they make their

payment. You will notice above that in all instances the textbox MaxLength properties are given values. This is an important and very simple security task. Maximum field lengths are defined in the PayPal developer's guide: (Paypal, 2009) . Where possible, list boxes are utilized to ensure only allowable values are possible. You will also notice a Regular Expression Validator is used to validate the credit card number, and Required Field Validators are used for all other text fields. Although the payment gateway will also check the number against their own validators it is prudent to check it at the first possible opportunity to reduce transaction errors. The regular expression used will be explained in the next section.

For PHP developers much of the code in the above example is the same except that all fields are standard HTML controls, as opposed to ASP.NET controls, and the validation is handled by a JavaScript call from the form's submit button.

```
<!--PHP / HTML-->
<form id="paymentForm" action="payment_form_handler.php">
  card type:
  <!--Card choices here are decided by your merchant account and what
  cards your account can accept! -->
  <select id="ddlCardType">
    <option>Visa</option>
    <option>Mastercard</option>
    <option>American Express</option>
  </select>
  card number: <input name="tbCardNumber" type="text" maxlength="16" />
  <br />
  expires :
  <!--Include obviously all months to '12'-->
  <select id="ddlExpiryMonth">
    <option>01</option>
    <option>02</option>
    <option>03</option>
  </select>
  <select id="ddlExpiryYear">
    <option>2012</option>
    <option>2013</option>
    <option>2014</option>
  </select>
  CvV2 number : <input name="tbCVV2" type="text" maxlength="4" />
  <br />
  first name : <input name="tbFirstName" type="text" maxlength="30" />
  last name: <input name="tbLastName" type="text" maxlength="30" />
  <br />
  street address : <input name="tbStreetAddress" type="text" maxlength="30" />
  ZIP/ Postal code : <input name="tbPostCode" type="text" maxlength="9" />

  <input id="btPay" type="button" value="pay" onclick="javascript:
  submitform()" />
</form>
```

3.2 User input validation

As discussed in Section 1 validating the inputted text from users of the payment form is vital to combat many security threats. Mainly threats to the integrity of the e-commerce site itself, but also the resultant threats to the privacy of its users' personal information. Validating all user input fields on the payment form it therefore vital.

JavaScript is the most widely used client scripting language for the purposes of validation, so this section will present a script for PHP programmers to check user inputted text before the form is posted. The script will use Regular Expressions as a validation tool. For .NET programmers they can make use of .NET's regular expression validators, so no further JavaScript needs to be written to validate form data.

There are tools available for bypassing client JavaScript validation functions, essentially by manipulating the validated contents of the HTTP request packet after it has left the browser environment. (Lawrence, 2012) hence it is important to repeat the validation process once HTTP requests are received by the web server.

3.2.1 The composition of a credit/debit card number

Most card numbers are 16 digits long, the first one or two digits specifies the industry, all financial service organizations begin with 4 or 5 (hence all Visa cards start with 4, all MasterCard cards start with 5). The first 6 digits together make up the Issuer Identification Number (IIN) so from the first 6 digits you can tell that the card is a Nordea Bank Visa Card for example. The 7th to the second to last digit is the customer's account number, so for a 16 digit card number the account number is 9 digits long. The final digit is the check sum, used to validate the card according to the Luhn algorithm (MintLife, 2011). The regular expression validators presented below take advantage of this format to determine if the card number is valid.

3.2.2 Regular expression validators

Regular expressions are a powerful tool for checking that a data string matches a specified rule, or indeed set of rules, and consequently they can ensure that a string, or part of a string matches exactly with what is specified in the expression.

Before I present the expressions for validating card numbers, for the benefit of those not familiar with regular expression syntax I will explain the basic syntax in relation to the examples presented below;

^	The match must occur at the beginning of the tested string
\$	The match must occur at the end of the tested string
The examples below feature ^ at the start and \$ at the end, this means the whole string tested must match exactly to the expression.	
[]	Character grouping (only those numbers specified are allowed for that digit) can be a range [1-5] or an array [12345]
(?:)	Means a sub expression
{n}	The previous sub expression or grouping must be matched n number of times

(msdn, 2011)

Below are the regular expressions for validating each major card type, of course these can be easily adapted to suit other card types if needed (Goyvaerts, 2009)

Visa	<code>^4[0-9]{12}(?:[0-9]{3})?\$</code>
Mastercard	<code>^5[1-5][0-9]{14}\$</code>
American Express	<code>^3[47][0-9]{13}\$</code>
Diners Club	<code>^3(?:0[0-5] [68][0-9])[0-9]{11}\$</code>
Discover	<code>^6(?:011 5[0-9]{2})[0-9]{12}\$</code>
JCB	<code>^(?:2131 1800 35\d{3})\d{11}\$</code>

(Goyvaerts, 2009)

3.2.3 JavaScript validation and form submission script

The script below is relevant only to the PHP developers reading this study. As mentioned ASP.NET developers do not need to manually write validation scripts as these are automatically generated by the .NET framework, if there are validator tags included within the HTML document.

```
<script type="text/javascript">
function SubmitForm() {
    var e = document.getElementById("ddlCardType");
    var cardType = e.options[e.selectedIndex].value;
    var e2 = document.getElementById("tbCardNumber");
    var cardNumber = e2.value;

    if (isValidCreditCard(cardType, cardNumber)) {
        document.forms["paymentForm"].submit();
    }
    else{

        // Show a message to the user that card details are wrong
    }
}

function IsValidCreditCard(type, ccnum) {
    if (type == "Visa") {
        // Visa: length 16, prefix 4, dashes optional.
        var re = /^4[0-9]{12}(?:[0-9]{3})?$/i;
    } else if (type == "MasterCard") {
        // Mastercard: length 16, prefix 51-55, dashes optional.
        var re = /^5[1-5][0-9]{14}$/i;
    } else if (type == "Discovery") {
        // Discover: length 16, prefix 6011, dashes optional.
        var re = /^6(?:011|5[0-9]{2})[0-9]{12}$/i;
    } else if (type == "AmEx") {
        // American Express: length 15, prefix 34 or 37.
        var re = /^3[47][0-9]{13}$/i;
    } else if (type == "Diners") {
        // Diners: length 14, prefix 30, 36, or 38.
        var re = /^3(?:0[0-5]|[68][0-9])[0-9]{11}$/i;
    }
    else if (type == "JCB") {
        // Diners: length 14, prefix 30, 36, or 38.
        var re = /^(?:2131|1800|35\d{3})\d{11}$/i;
    }

    if (!re.test(ccnum)) return false;
    // Checksum ("Mod 10")
    // Add even digits in even length strings or odd digits in odd length
    // strings.
    var checksum = 0;
    for (var i = (2 - (ccnum.length % 2)); i <= ccnum.length; i += 2) {
        checksum += parseInt(ccnum.charAt(i - 1));
    }
    // Analyze odd digits in even length strings or even digits in odd
    // length strings.
    for (var i = (ccnum.length % 2) + 1; i < ccnum.length; i += 2) {
        var digit = parseInt(ccnum.charAt(i - 1)) * 2;
        if (digit < 10) { checksum += digit; } else { checksum += (digit -
9); }
}
}
</script>
```

```
    }  
    if ((checksum % 10) == 0) return true; else return false;  
  }  
</script>
```

IsValidCreditCard function adapted from (Breaking Par Consulting Inc, 2012)

If you refer back to the PHP HTML form example given earlier you will see that the SubmitForm() function is called from the button within the 'paymentForm' form. This function then simply calls the IsValidCreditCard(type,ccnum) function to check the card number's validity. If it is valid, it posts the form to the server handler (the .php file) specified in the 'action' attribute of the form. If the card is not valid, a message would normally be shown somewhere on the form, and the form is not posted.

This script should be extended also to validate all user input fields such as name and address fields.

3.3 Payment page usability

Shopping cart abandonment is a major barrier to the success of an e-commerce website. As mentioned in the introduction to this study up to 60% of shopping carts get abandoned at some point using the process (Baymard Institute, 2011, p. 4). A large percentage of those will be at the payment page, so for the purposes of this study it is important to look at the factors that affect payment page usability.

Whilst many studies cover the full range of e-commerce usability issues, there is always a section focused on the payment page specifically, commonly referred to as Checkout Usability. There are a number of aspects relating to payment page design that can affect a user's perception of trust, leading them to abandon their order even this late stage (Governor Technology, 2009, p. 11). Furthermore failure to implement a simple easy to use design or implement adequate debugging and testing across all browsers can lead to users abandoning their order. Next I outline a set of recommended consideration according to Governor Technology, a respected UK based software development company (Governor Technology, 2009). There is also several of my own recommendations.

Form design

Order the user fields are ordered in the same way as would appear on the card. Also ensure expiry date list controls are in the same format as would appear on the card. Do not have a default card type chosen, regardless of whether you believe most customers use say 'Visa'. Ensure the card type list box is set to 'choose'. Regarding the CVV number many users need help finding the code, this information is specific to the card issuer so should be adapted to match their choice from the card type box, often an image is used to show an example of where to find it. Do not include a Cancel button next to the 'Pay' or 'Confirm' button, many users will click the wrong one. Finally for validation errors, ensure you again provide helpful information about what is wrong, the regular expressions given earlier in this section for example could tell you if the card number is in the wrong format for the card type selected.

User instructions

Explain to users how to complete each field on the form, indicating required format, if the field is mandatory and also how the data they input will be used (Holst, 2011). Appendix 7 gives a good example of how this can be achieved. Ensure that even fields that seem obvious to you are explained, thus ensuring the user does not become frustrated. Remember that it is more likely that the user will not complete the payment, so all guidance is a help. Giving an example of what the inputted data should look like instead of extensive explanations can sometimes be more helpful. It is important to also indicate which fields will be treated as sensitive, for example include a lock icon and SSL certificate logo next to the card number field, as shown in Appendix7.

Payment options

Adding additional payment methods can greatly prevent shopping cart abandonment (Governor Technology, 2009, p. 11). I have already mentioned about the important benefits of adding Google Checkout and Amazon Payments buttons. If a user is only comfortable with using a certain type of payment method, and the site you are developing does not offer it, they will most likely go to another store that does. Offer as many different payment options as is practical and possible to offer.

Gateway response handling

Handle card errors effectively, telling the user what they can do to rectify the problem, do not display error codes with only a generic message. For example instead of “Error code: 17834, payment failed” use “We are unable to process your payment, please double check your card details”. Indeed make sure you are able to provide information for all potential errors received from the payment gateway such as card declined errors: “Unfortunately your card has been declined, is there another card you could try?”.

The other side of error code handling is success code handling, and this is equally as important. Although these issues relate to after the payment has been processed, return business is an important thing to garner in e-commerce. ‘Dead end’ receipt pages should be avoided, as the minimum offer a ‘continue shopping’ button, or offer a tell-a-friend’ form or display customer service FAQs.

Unsecure page browser errors

Comprehensive cross browser testing is important to ensure your SSL certificate is operating properly and there are no HTML or script elements that are causing page errors. HTML frames on the page often cause errors such as the common Internet Explorer error: “This page contains insecure as well as secure items: Do you want to download insecure as well as secure items” (Entrust, 2006). Either delete the offending frames or inform the user not to worry about the error.

Other details on the page

Whilst it is very important to keep the payment page simple, in terms of limiting the number of choices the user has to make, there is other information that should be provided to reassure the user at to what they are paying for, and inform them about the security credentials of the page. Firstly the user should be able to see a summary of their order and more importantly a full breakdown of the amount they are being charged, including taxes, delivery, and any other charges. The page should not include additional advertisements, add-on purchases or other options requiring user decisions. Do not display disclaimers, warnings or other messages that might concern the user. If these must be provided, add them as a link, which would open a pop-up window, easily achieved with JavaScript. Include a customer services telephone number and the name and address of the company. Users may want to ask a question before submitting

the payment, if they start navigating back to find contact details, it is likely the session will be lost.

3.4 SSL Certificate logos and informing users

I have discussed about the necessity for PCI compliance if you are handling payment card information through your site in the previous section. To be compliant means you need an SSL certificate, preferably one issued by a Certificate Authority (CA). A study on SSL consumer awareness by Entrust found that a key element of consumer confidence is the presence of a third party security logo, and a padlock icon somewhere in the browser; “85 percent of those who routinely conduct transactions online look for a specific icon or indicator (e.g., the “padlock” used with SSL certificate technology)” (Entrust, 2007). Whilst this is likely true, as this is not independent research one needs to be cautious of the findings. I would argue that the findings are essentially true, however far fewer regular users would really understand what these CA logos mean, i.e. what SSL is, what the padlock icon means, and how all these things help to ensure the security of your data. “Unfortunately, standard SSL certificates can sometimes be illegitimately obtained and the padlock icon can be reproduced on browsers, tricking even seasoned Internet users into phishing or man-in-the-middle fraud attacks” (Entrust, 2007). Consequently people who see a CA logo, such as those shown in Appendix 2, and a lock icon in the browser assume they are safe. If their personal information then gets stolen, their trust in your e-commerce site and others will be permanently damaged. As a developer of an e-commerce site it is your role to explain the importance of extended validation (EV-SSL) certificates, to your client’s business, and ensure also that the site’s users are informed fully about SSL, and about checking the EV-SSL certificate’s validity from the browser address bar, prior to making a payment. Site users should be told what they should see when they view details of the SSL certificate (as shown in Appendix 3). Indeed Nordea Bank is one of the few sites found as part of this study that explain the idea of SSL to their site users, something which is positive, and demonstrates that many users do not understand the concept. Indeed it would be fairly easy to include a script to detect the browser and version and then show an image, showing users exactly what they should see from the browser’s address bar.

Informing users in this way will help to build a strong bond of trust between the site's users and the company operating it.

3.5 Chapter Summary

This chapter has covered the main technical aspects of implementing a basic payment form, with a view to it being added to into a fully implemented checkout page. It has covered the HTML and JavaScript implementation of the form, looked in detail at page usability and consumer confidence issues. The important points to note are summarized below;

HTML Forms

- Use only the 'post' method for submitting payment card details, never the 'get' method
- Include extra 'hosted buttons' from payment providers such as Google Checkout, and Amazon Payments. As well as potentially attracting a wider customer base by offering more payment options, Google Checkout is also reported to have Search Engine Optimization (SEO) benefits.
- Include only required information in hidden HTML fields, necessary for such things as maintaining a user session.

User Input Validation

- Validate all user input fields in a client side script, as well as when received by the server.
- Use regular expressions and the check sum calculation for validating card numbers.

Payment page usability

- Design an easy to use payment form, with a commonly used layout, informing the user about each and every field, such as where to find the CVV number.
- Include as many payment options as viably possible.
- Handle all possible response codes from the payment gateway, and handle them effectively, providing useful feedback to the user as to what is happening.

SSL Certificate logos and informing the user

- Clearly display the logo provided by the Certificate Authority (CA).

- Explain to the user what SSL is, and how visually what they should expect to see if they click to show details of the certificate in their browser.

4 Back end implementation

This section looks at the server-side code implementation of a payments page. It looks in detail at how to interface with the payment gateway APIs chosen in chapter 2 of this study. In order to interface with such advantaged APIs, card details must be handled on the web server of the e-commerce business. Consequently Secure Socket Layer (SSL) connections must be implemented for the payment page. The subjects discussed in this section can therefore be summarised as;

- Admin issues relating to each payment gateway, for instance the setting up of test accounts, altering code to ensure the correct credentials are used, etc.
- Server side code will be presented relating to how to submit an HTTP request to the payment gateway servers, and how to handle the response. As two payment gateway implementations and two programming languages have been chosen for this study a total of four separate code examples will be presented in this section.
- SSL implementations will be presented for two separate types of server commonly used for ASP.NET and PHP websites namely IIS 7 for Windows Server 2008 and Apache 2.4.2.

4.1 Payment gateway user accounts

It may be the case that the client you would be developing for already has a merchant bank account, it being necessary to accept traditional point-of-sale card transactions, but unless they already have had an online payments facility for their customers they will need to set up a user account with the payment gateway of their choice. Initially a test user account will need setting up to debug your application extensively before live credit card details are used. It is also important potentially to pass on the test account log-in credentials to your client. If they are not familiar with the process of the payment gateway, they will need to become familiar with the transaction admin/management interface which is invariably included as part of the user account. The management interface is where the account holder can alter their account preferences, but most importantly they can pass transactions for settlement. This is normally

handled in batches automatically according to a default schedule, but it is important for your potential client to understand what information they can access, and alter through their management interface.

I have discussed in chapter 2 about the decisions affecting which payment gateway to choose, the options offered for managing transactions is a factor affecting that decision. The detailed discussions regarding the options for different accounts are out of the scope of this section, but here we discuss the accounts necessary for testing transactions between an e-commerce application and the gateways chosen as a focus for this study.

4.1.1 Paypal manager account

The transaction interface is not available unless the client signs up for a payflow pro account, and is prepared to pay for the signup costs. If they wish to do this they can from the link below:

https://www.paypal.com/us/cgi-bin/?cmd=_payflow-get-started-outside

However test transactions can be sent, using specific test card numbers and calling specific test API urls. This way the e-commerce site can be debugged, and transactions can be sent, and response codes can be checked and handled as they would for real live transactions.

4.1.2 Authorize.net test account

Authorize.net offer that you can set up a fully functioning test account, which functions in the same way as a live account. The merchant (i.e. your client) then later has the option of upgrading the account to a live account. The test account can see set up from the link below:

<https://developer.authorize.net/testaccount/>

As a developer you then have the option of giving the test account log-in details to your client in order for them to familiar themselves with the transaction management interface.

4.2 Server side code implementation

4.2.1 PayPal Payflow Pro

As discussed in the previous chapters the first payment gateway integration we will look at is PayPal's most advanced integration, Payflow Pro. PayPal publish the APIs as code libraries for developers to include locally in their e-commerce site implementation. This is useful as it no doubt ensures fewer errors when transmitting to PayPal's server, as instances of the API classes demand that all necessary information is provided prior to data being posted. The code examples are based on SDK downloads which can be viewed for downloaded from PayPal (PayPal, 2012).

ASP.NET Implementation

This payment gateway integration for ASP.NET is taken care of by including a .DLL file in your site project as a 'new reference', or simply place the .DLL in the bin folder and include the 'using' references relating to PayPal as shown below.

For the purposes of this example most commenting has been deleted to more clearly show the objects that the code behind file is using from the .DLL, and better show the process of submitting a transaction. Although all required objects are shown in the code below, each object requires a number of properties to be set, which can be referred to in the full API documentation.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using PayPal.Payments.Common;
using PayPal.Payments.Common.Utility;
using PayPal.Payments.DataObjects;
using PayPal.Payments.Transactions;
using System.Threading;

public partial class Default2 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        String RequestID = PayflowUtility.RequestId;
        UserInfo User = new UserInfo("-", "-", "-", "-");
        PayflowConnectionData Connection = new PayflowConnectionData();
        Invoice Inv = new Invoice();
    }
}
```

```

String usCurrency = "USD";
Currency Amt = new Currency(new decimal(19.99), usCurrency);
Inv.Amt = Amt;

// *** Set the Billing Address details. ***
BillTo Bill = new BillTo();
Bill.BillToFirstName = tbFirstName.Text;
Bill.BillToLastName = tbLastName.Text;
Bill.BillToStreet = tbStreetAddress.Text;
Bill.BillToZip = tbPostCode.Text;

// Set the BillTo object into invoice.
Inv.BillTo = Bill;

ShipTo Ship = new ShipTo();
// If shipping address is different to billing address this can be set here

// ***          Create Customer Data ***

CustomerInfo CustInfo = new CustomerInfo();

UserItem nUser = new UserItem();
    nUser.UserItem1 = "TUSER1"; //e-commerce site user-name
    Inv.UserItem = nUser;

// *** Create a new Payment Device - Credit Card data object. ***

CreditCard CC = new CreditCard(tbCardNumber.Text,
    ddlExpiryMonth.SelectedValue+ddlExpiryYear.SelectedValue);
    CC.Cvv2 = tbCvV2.Text;
CardTender Card = new CardTender(CC);
SaleTransaction Trans = new SaleTransaction(User, Connection, Inv, Card,
    RequestID);
ClientInfo cInfo = new ClientInfo();
cInfo.IntegrationProduct = "Test";
cInfo.IntegrationVersion = "1.0";
Trans.ClientInfo = cInfo;
Trans.Verboesity = "HIGH";

// Try to submit the transaction up to 3 times with 5 second delay. This
// can be used
// in case of network issues. The idea here is since you are posting via
// HTTPS behind the scenes there
// could be general network issues, so try a few times before you tell cus-
// tomer there is an issue.
int trxCount = 1;
bool RespRecd = false;
while (trxCount <= 3 && !RespRecd)
{
    // Submit the Transaction
    Response Resp = Trans.SubmitTransaction();

    // Display the transaction response parameters.
    if (Resp != null)
    {
        // Here would go response code handling. There is an extensive num-
        // ber of codes, all of which should
        //Be handled effectively
    }
    else
    {
        Thread.Sleep(5000); // let's wait 5 seconds to see if this is a tem-
        // porary network issue.
        trxCount++;
    }
}

```

```
    }
    if (!RespRecd)
    {
        // Transaction failed - provide a message
    }
}
}
```

PHP Implementation

The PHP example of how to submit a card payment transaction to PayPal is a broadly similar process, involving name and value pairs (NVPs) collected from data submitted on the HTML form. Then a call to a function in another referenced .PHP file (CallerService.php), namely 'hash_call("doDirectPayment",\$nvpstr)'. The hash_call function within the CallerService.PHP is the one which constructs the HTTP (or correctly HTTPS) request using a cURL PHP library. CURL essentially allows a PHP application to communicate easier with many different servers over many different protocols, for instance http, https, ftp, gopher, telnet, dict, file and others. (PHP, 2012). This makes the process of transmitting over an SSL connection (i.e. HTTPS) easier to configure. CURL was added to PHP from version 4.0.2, so you will need to check the version of PHP being used on your web server, and possibly enable the cURL library if the functions used in the example above cannot be found.

```
<?php
/*****
DoDirectPaymentReceipt.php

Submits a credit card transaction to PayPal using a
DoDirectPayment request.

The code collects transaction parameters from the form
displayed by DoDirectPayment.php then constructs and sends
the DoDirectPayment request string to the PayPal server.
The paymentType variable becomes the PAYMENTACTION parameter
of the request string.

After the PayPal server returns the response, the code
displays the API request and response in the browser.
If the response from PayPal was a success, it displays the
response parameters. If the response was an error, it
displays the errors.

Called by paymentPage.php.

Calls CallerService.php and APIError.php.

*****/
```

```

require_once 'CallerService.php';
session_start();

/**
 * Get required parameters from the web form for the request
 */
$paymentType =urlencode('Sale');
$firstName =urlencode( $_POST['tbFirstName']);
$lastName =urlencode( $_POST['tbLastName']);
$creditCardType =urlencode( $_POST['ddlCardType']);
$creditCardNumber = urlencode($_POST['tbCardNumber']);
$expDateMonth =urlencode( $_POST['ddlExpiryMonth']);

// Month must be padded with leading zero
$padDateMonth = str_pad($expDateMonth, 2, '0', STR_PAD_LEFT);

$expDateYear =urlencode( $_POST['ddlExpiryYear']);
$cvv2Number = urlencode($_POST['tbCvV2']);
$address1 = urlencode($_POST['StreetAddress']);
$address2 = urlencode('');
$city = urlencode('');
$state =urlencode('');
$zip = urlencode($_POST['tbPostCode']);
$amount = urlencode('19.99');
//$currencyCode=urlencode($_POST['currency']);
$currencyCode="EUR";
$paymentType=urlencode(''); // check this value from the documentation

/* Construct the request string that will be sent to PayPal.
   The variable $nvpstr contains all the variables and is a
   name value pair string with & as a delimiter */
$nvpstr="&PAYMENTACTION=$paymentType&AMT=$amount&CREDITCARDTYPE=$creditCardType&ACCT
=$creditCardNumber&EXPDATE="
.$padDateMonth.$expDateYear."&CVV2=$cvv2Number&FIRSTNAME=$firstName&LASTNAME
=$lastName&STREET=$address1&CITY=$city&STATE=$state".
"&ZIP=$zip&COUNTRYCODE=US&CURRENCYCODE=$currencyCode";

/* Make the API call to PayPal, using API signature.
   The API response is stored in an associative array called $resArray */
$resArray=hash_call("doDirectPayment",$nvpstr);

/* Display the API response back to the browser.
   If the response from PayPal was a success, display the response parameters'
   If the response was an error, display the errors received using APIError.php.
   */
$ack = strtoupper($resArray["ACK"]);

if($ack!="SUCCESS") {
    $_SESSION['reshash']=$resArray;
    $location = "APIError.php";
    header("Location: $location");
}

?>

<html>
<head>
    <title>PayPal PHP SDK - DoDirectPayment API</title>
    <link href="sdk.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <br>

```

```
<center>
<font size=2 color=black face=Verdana><b>Do Direct Payment</b></font>
<br><br>

<b>Thank you for your payment!</b><br><br>

<table width=400>

    <?php
        require_once 'ShowAllResponse.php';
    ?>
</table>
</center>
<a class="home" id="CallsLink" href="index.html">Home</a>
</body>
</html>
```

Of course in this example the full response from the payment gateway server would be displayed on the webpage. While this is useful for testing, in practice, as I have discussed, the handling of response information should be extensive, and only useful, necessary and helpful information should be displayed on screen.

In reality it is likely that proper response code handling would account for the majority of the total lines of code, as there are many different eventualities that must be accounted for. Thankfully in the case of this particular payment gateway integration, and also with the Authorize.net integration to follow, both offer clear documentation on what response information is included in the associative array returned, and what it means to the end-user.

4.2.2 Authorize.net Advanced Integration Method (AIM)

As discussed this method involves an HTTP request using the POST method, and includes all required data as name and value pairs (NVPs). The HTTP response then provides details of whether the payment was successful or not. The example code implementations used here as a basis can be viewed or downloaded from <http://developer.authorize.net/downloads/samplecode/>.

ASP.NET Implementation

As you can see below the HTTP response is sent to the .DLL file handling the transactions so no installation of additional code libraries are needed, making it quite a simple process.

```

using System;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Net;
using System.IO;
using System.Collections.Generic;

namespace AIM_Example
{
    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        protected void btPay_Click(object sender, EventArgs e)
        {
            // By default, this sample code is designed to post to our test server
            // for
            // developer accounts: https://test.authorize.net/gateway/transact.dll
            // for real accounts (even in test mode), please make sure that you are
            // posting to: https://secure.authorize.net/gateway/transact.dll
            String post_url = "https://test.authorize.net/gateway/transact.dll";

            Dictionary<string, string> post_values = new Dictionary<string,
            string>();
            //the API Login ID and Transaction Key must be replaced with valid val-
            //ues
            post_values.Add("x_login", "API_LOGIN_ID");
            post_values.Add("x_tran_key", "TRANSACTION_KEY");

            post_values.Add("x_version", "3.1");
            post_values.Add("x_delim_data", "TRUE");
            post_values.Add("x_delim_char", "|");
            post_values.Add("x_relay_response", "FALSE");

            post_values.Add("x_type", "AUTH_CAPTURE");
            post_values.Add("x_method", "CC");
            post_values.Add("x_card_num", tbCardNumber.Text);
            post_values.Add("x_exp_date",
            ddlExpiryMonth.SelectedValue+ddlExpiryYear.SelectedValue);

            post_values.Add("x_amount", "19.99");
            post_values.Add("x_description", "Sample Transaction");

            post_values.Add("x_first_name", tbFirstName.Text);
            post_values.Add("x_last_name", tbLastName.Text);
            post_values.Add("x_address", tbStreetAddress.Text);
            //post_values.Add("x_state", "WA");
            post_values.Add("x_zip", tbPostCode.Text);
            // Additional fields can be added here as outlined in the AIM integra-
            //tion
            // guide at: http://developer.authorize.net

            // This section takes the input fields and converts them to the proper
            // format
            // for an http post. For example:
            "x_login=username&x_tran_key=a1B2c3D4"
            String post_string = "";

            foreach (KeyValuePair<string, string> post_value in post_values)
            {

```



```

        post_string += post_value.Key + "=" +
        HttpUtility.UrlEncode(post_value.Value) + "&";
    }
    post_string = post_string.TrimEnd('&');

    // The following section provides an example of how to add line item
    // details to
    // the post string. Because line items may consist of multiple values
    // with the
    // same key/name, they cannot be simply added into the above array.
    //
    // This section is commented out by default.
    /*
string[] line_items = {
    "item1<>golf balls<|><|>2<|>18.95<|>Y",
    "item2<>golf bag<|>Wilson golf carry bag, red<|>1<|>39.99<|>Y",
    "item3<>book<|>Golf for Dummies<|>1<|>21.99<|>Y";

foreach( string value in line_items )
{
    post_string += "&x_line_item=" + HttpUtility.UrlEncode(value);
}
*/

// create an HttpRequest object to communicate with Authorize.net
HttpRequest objRequest = (HttpRequest)WebRequest.Create(post_url);
objRequest.Method = "POST";
objRequest.ContentLength = post_string.Length;
objRequest.ContentType = "application/x-www-form-urlencoded";

// post data is sent as a stream
StreamWriter myWriter = null;
myWriter = new StreamWriter(objRequest.GetRequestStream());
myWriter.Write(post_string);
myWriter.Close();

// returned values are returned as a stream, then read into a string
String post_response;
HttpWebResponse objResponse = (HttpWebResponse)objRequest.GetResponse();
using (StreamReader responseStream = new
StreamReader(objResponse.GetResponseStream()))
{
    post_response = responseStream.ReadToEnd();
    responseStream.Close();
}

// the response string is broken into an array
// The split character specified here must match the delimiting charac-
// ter specified above
Array response_array = post_response.Split('|');

// the results are output to the screen in the form of an html numbered
// list.
resultSpan.InnerHtml += "<OL> \n";
foreach (string value in response_array)
{
    resultSpan.InnerHtml += "<LI>" + value + "&nbsp;</LI> \n";
}
resultSpan.InnerHtml += "</OL> \n";
// individual elements of the array could be accessed to read certain
// response
// fields. For example, response_array[0] would return the Response
// Code,
// response_array[2] would return the Response Reason Code.

```

```

// for a list of response fields, please review the AIM Implementation
Guide
    }
}
}

```

The code here represents an example of how one might implement the code behind file of the web form example presented in the previous section. The `btPay_Click` method, as well as the control IDs given here also reflect those presented in the previous chapter detailing the HTML form code. In the previous chapter it was presented, in the payment form HTML code, that the card security number (CVV2) was also being collected from the user. It was discussed that this would be good practice for security purposes. To submit this number as part of the post string, it would need to be added as an additional variable by adding it to the `post_values` dictionary. You would need to check the AIM integration guide, as to the correct key and value pair format, as it is not a standard variable. The API Login ID and Transaction Key values together provide the merchant authentication required for access to the payment gateway (Authorize.net, 2012, p. 74). You will receive your ID and Transaction Key on opening either an Authorize.net test account or upgrading a test account to an actual merchant account.

All API fields should now be fairly self explanatory except for the ‘`x_version`’, ‘`x_delim_data`’, ‘`x_delim_char`’. These are referred to in the documentation as best practice fields. They are not required but it makes the connection with the payment gateway more robust. In addition to these ‘`x_ncap_char`’ and ‘`x_relay_response`’ are also recommended. None of these fields’ values should ever need changing but in order to understand what they relate to please refer to the table below:

<code>x_version</code>	<p>Format: 3.0, 3.1</p> <p>Notes: Indicates to the system the set of fields that will be included in the response: 3.0 is the default version. 3.1 allows the merchant to utilize partial authorizations and the Card Code feature, and is the current standard version. It is highly recommended that you submit this field on a per-transaction basis to be sure that the formats of transaction requests and the responses you receive are consistent.</p>
<code>x_delim_data</code>	<p>Format: TRUE, T, YES, Y</p> <p>Notes: A value of TRUE indicates a request for delimited re-</p>

	<p>sponse from the payment gateway. Since all AIM transactions are direct response, a value of TRUE is required.</p> <p>Submit this field for each transaction to be sure that transaction responses are returned in the correct format.</p>
x_delim_char	<p>Notes: The character that is used to separate fields in the transaction response. The payment gateway will use the character passed in this field or the value stored in the Merchant Interface if no value is passed.</p> <p>If this field is passed and the value is null, it overrides the value stored in the Merchant Interface and the transaction response contains no delimiting character.</p> <p>It is recommended that you submit this field on a per-transaction basis to be sure that transaction responses are returned in the correct format.</p>
x_encap_char	<p>Notes: The character that is used to encapsulate the fields in the transaction response. This is only necessary if it is possible that your delimiting character could be included in any field values.</p> <p>The payment gateway will use the character passed in this field or the value stored in the Merchant Interface if no value is passed.</p>
x_relay_response	<p>Notes: This field, when set to TRUE, instructs the payment gateway to return transaction results to the merchant by means of an HTML form POST to the merchant's Web server for a relay response.</p> <p>For our AIM example here it should always to be set to false.</p>

(Authorize.net, 2012, pp. 64-75)

PHP Implementation

This implementation follows the logic of the ASP.NET version, and gives similar comments for each section of code. The only difference is the library used for the HTTP request, it being a special library which might not be enabled as standard. I will explain this after presenting the code.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<HTML lang='en'>
<HEAD>
      <TITLE> Sample AIM Implementation </TITLE>
</HEAD>
<BODY>
<P> This sample code is designed to generate a post using Authorize.net's
Advanced Integration Method (AIM) and display the results of this post to
the screen. </P>
<P> For details on how this is accomplished, please review the readme file,
the comments in the sample code, and the Authorize.net AIM API documentation
found at http://developer.authorize.net </P>
<HR />

```

```

<?PHP

// By default, this sample code is designed to post to our test server for
// developer accounts: https://test.authorize.net/gateway/transact.dll
// for real accounts (even in test mode), please make sure that you are
// posting to: https://secure.authorize.net/gateway/transact.dll

$post_url = "https://test.authorize.net/gateway/transact.dll";

        // VALUES FROM THE FORM
$firstName      = $_POST['tbFirstNmae'];
$lastName       = $_POST['tbLastName'];
$streetAddress  = $_POST['tbStreetAddress'];
$postcode       = $_POST['tbPostCode'];
$ccNum          = $_POST['tbCardNumber'];
$cvv2          = $_POST['tbCVV2'];
$expiryMonth    = $_POST['ddlExpiryMonth'];
$expiryYear     = $_POST['ddlExpiryYear'];

$post_values = array(

        // the API Login ID and Transaction Key must be replaced with valid
        values
        "x_login"           => "API_LOGIN_ID",
        "x_tran_key"       => "TRANSACTION_KEY",

        "x_version"        => "3.1",
        "x_delim_data"     => "TRUE",
        "x_delim_char"     => "|",
        "x_relay_response" => "FALSE",

        "x_type"           => "AUTH_CAPTURE",
        "x_method"         => "CC",
        "x_card_num"       => $ccNum,
        "x_exp_date"       => $expiryMonth+$expiryYear,

        "x_amount"         => "19.99",
        "x_description"    => "Sample Transaction",

        "x_first_name"     => $firstName,
        "x_last_name"      => $lastName,
        "x_address"        => $streetAddress,
        "x_zip"            => $postcode
        // Additional fields can be added here as outlined in the AIM integra-
tion
        // guide at: http://developer.authorize.net
);

// This section takes the input fields and converts them to the proper format
// for an http post.  For example: "x_login=username&x_tran_key=a1B2c3D4"

$post_string = "";
foreach( $post_values as $key => $value )
    { $post_string .= "$key=" . urlencode( $value ) . "&"; }
$post_string = rtrim( $post_string, "& " );

// The following section provides an example of how to add line item details to
// the post string.  Because line items may consist of multiple values with the
// same key/name, they cannot be simply added into the above array.
//
// This section is commented out by default.
/*

```

```

$line_items = array(
    "item1<>golf balls<><>2<>18.95<>Y",
    "item2<>golf bag<>Wilson golf carry bag, red<>1<>39.99<>Y",
    "item3<>book<>Golf for Dummies<>1<>21.99<>Y");

foreach( $line_items as $value )
    { $post_string .= "&x_line_item=" . urlencode( $value ); }
*/

// This sample code uses the CURL library for php to establish a connection,
// submit the post, and record the response.
// If you receive an error, you may want to ensure that you have the curl
// library enabled in your php configuration

$request = curl_init($post_url); // initiate curl object
    curl_setopt($request, CURLOPT_HEADER, 0); // set to 0 to eliminate
    header info from response
    curl_setopt($request, CURLOPT_RETURNTRANSFER, 1); // Returns response
    data instead of TRUE(1)
    curl_setopt($request, CURLOPT_POSTFIELDS, $post_string); // use HTTP
    POST to send form data
    curl_setopt($request, CURLOPT_SSL_VERIFYPEER, FALSE); // uncomment this
    line if you get no gateway response.
    $post_response = curl_exec($request); // execute curl post and store
    results in $post_response
    // additional options may be required depending upon your server con-
    figuration
    // you can find documentation on curl options at
    http://www.php.net/curl_setopt
curl_close ($request); // close curl object

// This line takes the response and breaks it into an array using the specified de-
// limiting character
$response_array = explode($post_values["x_delim_char"],$post_response);

// The results are output to the screen in the form of an html numbered list.
echo "<OL>\n";
foreach ($response_array as $value)
{
    echo "<LI>" . $value . "&nbsp;</LI>\n";
}
echo "</OL>\n";
// individual elements of the array could be accessed to read certain response
// fields. For example, response_array[0] would return the Response Code,
// response_array[2] would return the Response Reason Code.
// for a list of response fields, please review the AIM Implementation Guide
?>
</BODY>
</HTML>

```

Similarly to the PayPal example earlier you can see here directly the use of the cURL library to construct and send an HTTPS request. You will need to check that the version of PHP on your web server supports this library, also you may receive run-time errors.

4.3 Enabling SSL for Apache and Windows Server

We will now look at the process of enabling an SSL connection for the two most common servers hosting ASP.NET and PHP sites. The SSL connection will need to be enabled for the payment page in this example, but in practical terms, a full e-commerce site implementation may require the connection on several pages dependent on the site's design. As a simple guideline any page which is transmitting sensitive data is a page that should have SSL enabled.

4.3.1 IIS 7 for Windows Server 2008

For this section I will focus on how to set up SSL for a single page in IIS 7.0 for Windows Server 2008. For those not familiar with Windows Server, IIS (Internet Information Services) is the web server implementation on Windows Server, and includes a GUI for managing all aspects of site administration. On opening IIS you will see a tree navigation panel on the left, click on the server name (the tree root). This displays all related options in the main centre window. One of the options is 'Server Certificates'. On clicking this you have the option to import a certificate. This should be done if you have/ or your client has already bought an SSL certification service from a Certificate Authority (CA) (See Appendix 2 for commonly used CAs), and so will have a certificate to import. We discussed in chapter 2 that this is a necessity if an e-commerce site is handling credit card details. However for the purposes of testing you can also create your own certificate to use to encrypt HTTP responses and decrypt HTTP requests. To create a certificate just click 'create self-signed certificate', and give the certificate a name.

The next stage is to create a website binding so the website can be accessed through HTTPS protocol as well as HTTP. Please refer to Appendix 9 for pictures of the relevant windows. From the 'connections' tree navigation on the left pane in IIS, find and click on your website, then from the 'actions' menu on the right panel click 'bindings'. You should already see HTTP listed as a binding. You must click 'add' to add the HTTPS binding. Choose HTTPS from the 'type' menu, then from the SSL Certificate menu, choose the test certificate you created earlier, or choose your proper SSL certificate if you have one. On clicking OK you should see now HTTPS listed with HTTP in

the binding list. You have now enabled SSL for your e-commerce site. If you try opening a page, for example 'https://myecommercesite/paymentpage.aspx' it is likely you will see a warning about the self-signed certificate, if you have used the test certificate (Guthrie, 2007). This is one of the reasons that a certificate must be purchased from a CA. As well as meeting industry guidelines, a properly signed certificate stops browser warnings from being shown, which are guaranteed to alarm users and potentially cause a dramatic loss in sales.

So how do we enable SSL for only the payment page? At the moment all pages on the site can be requested using either HTTP or HTTPS, but for the payment page we want to ensure that only HTTPS requests are allowable. It is not enough to ensure that the navigation link to the payment page is specifying HTTPS:// as this still means that a hacker can exploit the fact that the page can still be accessed over HTTP. You should instead check each request to ensure that it is a secure request, or at least always redirect the request to the secure version of the page. This can be done in the Page_Load() method in the code behind file. The simple method below can be called from the Page_Load() method, redirecting the user to the secure version of the page if the HTTPS mode is not part of the request:

```
/// <summary>
/// Redirect to the corresponding secure page.
/// Assumption: IIS web site is configured in port 80.
/// </summary>
public void RedirectToSecurePage()
{
    var httpsMode = string.Empty;
    var serverName = string.Empty;
    var url = string.Empty;

    for (var i = 0; i < Request.ServerVariables.Keys.Count; i++)
    {
        var key = Request.ServerVariables.Keys[i];
        if (key.Equals("HTTPS"))
        {
            httpsMode = Request.ServerVariables[key];
        }
        else if (key.Equals("SERVER_NAME"))
        {
            serverName = Request.ServerVariables[key];
        }
        else if (key.Equals("URL"))
        {
            url = Request.ServerVariables[key];
        }
    }
    if (httpsMode.Equals("off"))
    {
        Response.Redirect(string.Concat("https://", serverName, url));
    }
}
```

```
}  
}
```

(Wickramasinghe, 2011)

4.3.2 Apache

I will focus here on guidelines for the implementation of SSL for the most current version of Apache at the time of writing (2.4.2). For these guidelines I will assume Apache is installed on a Linux Server

IIS 7.0 actually includes good support for PHP, but I would imagine that IIS would not be the first choice for a PHP developer to host his/her applications. Apache2 is build up from a large number of modules, PHP being one of them, so this level of integration makes it the most common choice of web server for PHP applications. PHP is a component of the popular LAMP stack (Linux, Apache, MySQL, PHP) for web applications. (Wikipedia, 2012)

SSL is a module within Apache and is easily enabled by issuing the following command from the linux server's terminal;

```
a2enmod ssl
```

then you will need to restart Apache, to enable to module. The next stage is to obtain or create a server certificate. As with our Windows Server example, we will create a test self-signed certificate to demonstrate the process. Again from the terminal issue this command;

```
openssl req -new -x509 -days 365 -sha1 -newkey rsa:1024 \  
-nodes -keyout server.key -out server.crt \  
-subj '/O=Company/OU=Department/CN=www.example.com'
```

This command creates a certificate and stores the keys in the default folders. The `--subj` option specifies basic certificate details, so these should be changed to match your e-commerce site details, most importantly the `/CN=` attribute must match the site's URL exactly, otherwise warnings will be produced in the requesting browser window. Next you need to ensure that the server accepts both secure requests and non-secure. This is done by editing the `/etc/apache2/ports.conf` file to include the following lines;


```
Listen 80
Listen 443
```

Port 80 is the standard HTTP port, while 443 is the standard port for secure HTTPS requests (Wikipedia, 2012). Now we need to configure our e-commerce site to use SSL. This is done via the config file for the site which is found at `/etc/apache2/sites-enabled/yourecommerbcesite`. A new virtual host can be added for the page or pages you want to be SSL enabled. Let's assume we would only want the `paymentPage.php` file to be SSL enabled, the config file found at the path above should therefore include the following;

```
# =====
# SSL/TLS settings
# =====
NameVirtualHost *:443

<VirtualHost *:443>

    DocumentRoot "/local/www/myecommercesite/ssl"

    SSLEngine on
    SSLOptions +StrictRequire

    <Directory />
        SSLRequireSSL
    </Directory>

    SSLProtocol -all +TLSv1 +SSLv3
    SSLCipherSuite HIGH:MEDIUM:!aNULL:+SHA1:+MD5:+HIGH:+MEDIUM

    SSLRandomSeed startup file:/dev/urandom 1024
    SSLRandomSeed connect file:/dev/urandom 1024

    SSLSessionCache shm:/usr/local/apache2/logs/ssl_cache_shm
    SSLSessionCacheTimeout 600

    SSLCertificateFile /etc/apache2/ssl/server.crt
    SSLCertificateKeyFile /etc/apache2/ssl/server.key

    SSLVerifyClient none
    SSLProxyEngine off

    <IfModule mime.c>
        AddType application/x-x509-ca-cert          .crt
        AddType application/x-pkcs7-crl            .crl
    </IfModule>

    SetEnvIf User-Agent ".*MSIE.*" \
        nokeepalive ssl-unclean-shutdown \
        downgrade-1.0 force-response-1.0
</VirtualHost>
```

The payment page, requiring SSL, could be placed in a separate folder called 'ssl', then this VirtualHost entry defines that folder as requiring SSL to access it. The Directory tags specify that this host (our payment page) can only be reached via SSL. Also the SSL engine option must be set to 'on'. Other important options set the types of Cipher to use, the versions of the SSL protocol to use and where the certificate and encryption keys can be found.

4.4 Chapter Summary

This chapter has presented example technical implementations of server side code for ASP.NET and PHP web applications, for each of the payment gateway integration APIs chosen as a part of chapter 2. Before this however the section presented the importance of setting up test accounts with the chosen payment gateway to allow for testing, and familiarization with the transaction management interfaces.

Finally the section presented a guide for enabling SSL connections to the payment page for both the Apache Web server (most commonly used for hosting PHP web applications) and IIS 7 for Windows Server 2008 (used for hosting ASP.NET web applications). SSL is a requirement for meeting payment industry regulations.

5 Evaluation

Broadly speaking I am very pleased with the outcome of the study, given that all milestones were reached on time, with only 4 day slippage achieving the final milestone MS5 (Section 3 written). As I have experience of project management, I know the importance of working closely in accordance with the schedule, and the importance of meeting incremental milestones, even if this impacts on the quality of the work. When I realized I had not met milestone MS4 on time I made sure that I re-forecast the schedule to ensure that I could still meet the remaining agreed milestones on time. I realised that the impact could be severe if several small failings to meet the schedule go unnoticed, as the cumulative effect can be large.

The principal aim of this study was to provide a guide for the necessary elements of implementing card payment functionality to a website. Ultimately I believe that there is the majority of information a developer would need in order to effectively implement such functionality, so I feel happy this has been achieved. Also having recently reviewed popular answers to similar questions relating to implementing payment functionality, on developer forums such as www.stackoverflow.com, I was happy to see that contributors had chosen to address the question in a similar way, or had at least covered the same topics. Part of the core objective was also to produce a guide that was accessible and easy to read. This is difficult for me to access, and is only something that can be known through others' feedback upon reading the study, although it has been conscious in my mind throughout writing it.

According to my plan I decided that the best way to tackle the research question was in three distinct sections (later to become the 3 main chapters). These sections became apparent during the preliminary research stage. It appeared to me at that stage that each section could be researched and written almost in isolation, as they appeared well defined and separate. However this turned out to be more challenging than expected and resulted in me having to overlap the research of each section to some degree, as certain elements of each section overlapped. An example of this could be that chapter 2 is where the payment gateways would be chosen as a target for further study, but a big informant for that process would be the developers' knowledge of payment page

usability, covered in later chapters, due to its close association with front end design. Also I repeatedly found that on progressing to the following section my overall knowledge naturally became deeper, and therefore I left that I could possibly have written previous sections better. Consequently it would have been easier to handle all research sections first, before any writing took place.

As this study has focused quite heavily on quite a specific area of the much wider subject of e-commerce, it was a challenge to ensure that the scope remained focused. Consequently there are many instances within the text where I had to state that although a certain subject could naturally be discussed at certain points, it was unfortunately out of scope. As a result the Further Development section of this study is possibly quite extensive.

I didn't experience any problems relating to lack of research resources, i.e. books. Although I did attain a selection of e-commerce and web security books, the internet was by far the most used resource for finding current research, particularly relating to consumer issues relating to e-commerce, which I used as the principal evidence of need for this study.

Although this is not an implementation study, in order to present coherent guidance, I ultimately had to write quite a lot of code, or at least adapt quite heavily code examples provided online, this had an unexpected impact on time resources, and make meeting the milestones discussed earlier (principally MS5) more difficult.

6 Further development

This study has covered a subject which has a number of complex elements combined together, and one the aims of this study was to provide an overview of all necessary factors to achieve, in basic terms, the acceptance of a card payment through a web-page. Each of these necessary factors could be studied independently, and would provide more than enough material for a separate thesis study. With this in mind this section highlights the areas where this study could be extended to include other subjects relating less directly to the principal aims, but would enhance the study further. Additionally this section also highlights areas where, due to limited resources, and the specific aims of this study, they could have benefited from further study. Due to these resource and scope issues, some subjects have not been covered as extensively as others, so these subjects would be the ones most immediately in mind when thinking of further development.

This study has not included any primary research. Although this was something I originally envisaged would form part of the study, on writing the thesis plan it became apparent that time resources would not allow it. Therefore this is foremost in my mind when thinking of further development. There is very little research about consumers' experiences of checkout pages, and even less independent research. Therefore an extensive usability study of payment pages would be useful. Also a consumer survey on their awareness of security issues relating to online payments would be useful, particularly their understanding of SSL. I have highlighted in this study that improving consumer knowledge of SSL and other security issues could directly impact the revenue of businesses that sell online.

Database security relating to e-commerce is an area which has not been covered by this study but which would in reality need to be understood as most e-commerce site implementations offer the facility for storing card numbers, along with other personal details. It is out of the scope of this study as doesn't really directly relate to the technicalities of accepting card payments through a website.

As mentioned there are many issues relating to a wider e-commerce site implementation that would be a natural extension of this study. These would include; shopping cart development, looking at the differences between the extensive choices of ready-made modules, stock reservation techniques, session handling and user authentication techniques.

A closer look at the business decisions involved in choosing a payment gateway would also be a good extension to this study. The interaction between the client and the developer is an important one, as the client will no doubt have ideas about what they want from the e-commerce side of their business, but it would normally be the developer's job to suggest an online payment gateway that could meet these requirements. Therefore the developer would need to have a good understanding of business issues such as; access to payment processors, the long term business costs of selling online, scalability of the system and the payment gateway service, how adaptable the payment gateway and indeed the e-commerce application is to changes in business rules and indeed the business model. An analysis of the impact of these questions for a developer would be useful to look at in more depth.

Finally a more in depth look at client side security could provide further details on the security elements discussed in this study. For instance client SSL certificates, certificates that authenticate the client browser, are not extensively used in many e-commerce implementations. An analysis as to why they are not considered important and what the implications might be for a developer would be useful to analyse.

7 Summary

The growth of ecommerce has been enormous and will continue to be so according to research in the area (growth reported earlier in this study as being 13,5% year on year until 2015). The potential sales generation for individual business is therefore also enormous, but a large proportion (60%) of these potential sales are not being realised, due to customers abandoning their online transaction before payment.

This study has aimed to address the reasons behind one possible cause of this so called 'shopping-cart' abandonment. Research suggests that many payment pages are poorly designed and implemented, which at best creates a confusing experience for the consumer, and at worst causes serious security issues, allowing criminals to exploit the website, and its customers. This study has aimed to inform developers of all the key factors they need to consider when designing and implementing a payment page, and has given example code explaining how to implement the page effectively.

Chapter 2 looked into broad issues relating to the operating environment of an e-commerce website, all of which are important for a developer to understand in order to advise a potential client as to their responsibilities as an online merchant.. Firstly the section explained the process of an online card payment, and discussed the roles of a payment gateway (the most important organisation of concern to a web developer), a payment processor, and also about the interbank communications structure concerned with the actual transfer of monies between bank accounts. Knowledge of payment gateways is necessary for a developer as in most cases this is the company that the e-commerce application will need to communication with, through an API, in order to receive payments.

Next security threats were discussed. The problems caused by poor network security (Sniffer programs and other HTTP packet manipulation problems), the problems caused by poor site security (such as user authentication) and webpage form implementation (poor user input validation), and also the problems caused by DNS cache poisoning were all discussed. Regarding network security, SSL (Secure Socket Layer) protocol is principally used, to transmit data across the internet securely. Asymmetric key

encryption is used by SSL to encrypt and decrypt HTTP packets, along with the use of Digital Certificates to also authenticate that packets have been sent by the correct sender. PKI (Public Key Infrastructure) is used to add a further layer of strength to this system by paying for the services of a Certificate Authority, essentially a third party who also authenticates that the Digital Certificate is genuine.

Next the study focused on security standards relating to e-commerce, something which must be understood by developers. The principal sets of standards which need to be understood are PCI standards (Payment Card Industry Security Standards), and also PABP (Payment Application Best Practice). Both provide a practical set of rules to adhere to when developing e-commerce applications.

The final part of the chapter covered the types of implementation offered by payment gateway companies, meaning the ways that an e-commerce site can accept payments by communicating with the payment gateway servers. In the main it was highlighted that there are two main types of implementation. The first being hosted solutions (those methods that require the site user to be transferred to the gateway site to complete the transaction), the second being API solutions, involving a more seamless experience for the user, where the e-commerce site server and the gateway server communicate behind the scenes. Next the study looked at the factors influencing the choice of payment gateway. Factors highlighted included what payment cards and currencies the company supports, how respected they are as a company, what their fee structure is like, what kind of customer support they offer, and also what types of transactions they support all affect the decision which gateway to choose. From a developer point of view, all these things are affected by the type of client business you would be developing the e-commerce site for. For instance what types of products they sell determine what types of transactions they require (one stage or two stage), therefore this also influences the choice of payment gateway.

Finally several payment gateways and their services were analysed and PayPal's Payflow Pro, and Authorize.Net's Advanced Integration Method were chosen for further study.

Chapter 3 focused on front-end issues, such as payment form design, implementation, and user focused issues such as page usability. Firstly the chapter looked at the HTML form and import security issues relating to it such as the importance of using the ‘post’ method instead of the ‘get’ method. The form design as HTML was also included explaining what form fields were needed for the payment gateway API calls relating to the choices made earlier in the study.

As discussed earlier in the study there are many security issues relating to user interface implementation that must be handled effectively. User input validation is a very important aspect of front-end design, and effective validation can minimize a host of security threats. JavaScript is used as a first line of defence, so a user input validation script was presented to validate user input fields prior to posting to the server. The script used Regular Expressions extensively as an effective way of heavily limiting the allowable values in certain form fields.

Next the chapter looked at payment page usability, and highlighted some important things to include in the design. Consistent layout, providing clear guidance on all user input fields (most importantly where to find the CVV (card verification value number), and information on what point the user is in the payment/order process are all important. It is important to include as many payment options as possible, so as not to alienate potential customers who are only familiar and comfortable with paying by PayPal, for instance. Handling all possible response codes from the payment gateway is also vital to inform the user what has happened, particularly if the payment authorisation failed, and most importantly tell them how they can rectify it.

Finally the chapter looked a little at what SSL means to the user of the e-commerce site, if anything. The study found that clearly displaying the Certificate logo, and placing it specifically close to where the ‘pay’ or ‘commit’ button appears, improves the trust users will have of the site. Also explaining to them what SSL is and how it keeps them safe also builds trust, and shows the consumer that your e-commerce site is responsible and pro-active when it comes to security.

Chapter 4 was the final chapter and covered back-end implementation, such as server side code, and setting up SSL for all HTTP connections to the payments page. The chapter began by outlining how to get test credentials from payment gateways chosen in order to test API calls and check transactions are getting transmitted. Each gateway has its own requirements that allow test transactions to be submitted to them, and in most cases this involves a specific credit card number and related details, or a test merchant account that is provided for the purpose of testing. In either case it is easy to convert the account to start receiving live transactions, simply by altering those credentials in the relevant code files.

The chapter went on to summarise the important details from the comprehensive published developer guides available, and referenced as part of this study. The most important aspect to understand is the minimum field values that must be submitted as name and value pairs with each transaction. Whilst these sets of fields differ for each payment gateway and each API, this study has aimed to highlight that they are actually broadly very similar.

Finally instructions were provided as to how to enable SSL for connections to a specific webpage hosted on either IIS 7 for Windows Server or Apache (the most relevant servers for hosting ASP.NET and PHP applications respectively).

8 References

- Amazon. 2000. Amazon Simple Pay Advanced User Guide. URL: <http://awsdocs.s3.amazonaws.com/SimplePay/latest/simplepay-adv.pdf>. Accessed: 22. June 2012.
- Authorize.Net. 2012. Advanced Integration Method (AIM) Developer Guide. URL: http://www.authorize.net/support/AIM_guide.pdf. Accessed 13. July 2012.
- Authorize.Net. 2012. Compare e-commerce APIs. URL: <http://developer.authorize.net/api/compare/>. Accessed 10. June 2012.
- Authorize.Net. 2012. Credit Card Processing Diagram.. URL: <http://www.authorize.net/resources/howitworksdiagram/>. Accessed 09 June 2012.
- Authorize.Net. 2008. Developer Security Best Practices. Utah: Cyber Source Corporation. URL: <http://www.authorize.net/files/developerbestpractices.pdf>. Accessed: 12 July 2012.
- Ballad, J, Ballad, W. 2009. Securing PHP Web Applications. Pearson Education.
- Baymard Institute. 2011. E-Commerce Checkout Usability. Baymard Institute.
- Beal, V. 2012. Buyer's Guide: Choosing a Payment Gateway Provider. URL: http://www.ecommerce-guide.com/solutions/secure_pay/article.php/3869546/Buyers-Guide--Choosing-a-Payment-Gateway-Provider.htm. Accessed: 23 May 2012.
- Breaking Par Consulting Inc. (2012). Credit Card Validation. URL: <http://www.breakingpar.com/bkp/home.nsf/0/87256B280015193F87256CC70060A01B>. Accessed 30. June 2012.
- Chan, H, Raymmond, L, Tharam, D, Elizabeth, C. 2001. E-Commerce. Wiley. Sussex.
- Cusack. 2000. Secure Sockets Layer. URL: <http://searchsecurity.techtarget.com/definition/Secure-Sockets-Layer-SSL>. Accessed 20. June 2012

- Eisen, O. 2009. Telltale Signs of E-Commerce Fraud. URL:
<http://www.ecommercetimes.com/story/66278.html>. Accessed 10. August 2012.
- Enright, A. 2011. Global e-commerce to reach \$1.4 trillion in 2015. URL:
<http://www.internetretailer.com/2011/06/07/global-e-commerce-reach-14-trillion-2015>. Accessed 6. July 2012
- Entrust. 2006. Entrust Certificate Services Support Knowledge Base. URL:
<http://www.entrust.net/knowledge-base/technote.cfm?tn=5716>. Accessed 29. June 2012.
- Entrust. 2007. News Releases: Entrust Survey Finds Online Consumers Ready to Adopt Advanced Anti-Phishing Measures. URL:
<http://www.entrust.com/news/index.php?s=27003&item=72404>. Accessed 30. June 2012.
- Garfinkel. 2002. Web Security Privacy and Commerce. O'Reilly.
- Google. 2012. Buy Now Buttons. URL:
https://developers.google.com/checkout/developer/Google_Checkout_Buy_Now_Button_How_To. Accessed 10. June 2012.
- Google. 2012. Google Checkout Basic HTML Integration. URL:
https://developers.google.com/checkout/developer/Google_Checkout_Basic_HTML_Overview. Accessed 22. June 2012.
- Governor Technology. 2009. E-Commerce Usability Best Practices (White Paper). Governor Technology.
- Goyvaerts, J. 2009. Finding or Verifying Credit Card Numbers. URL:
<http://www.regular-expressions.info/creditcard.html>. Accessed 24. June 2012.
- Guthrie. 2007. Tip/Trick: Enabling SSL on IIS 7.0 Using Self-Signed Certificates. URL: <http://weblogs.asp.net/scottgu/archive/2007/04/06/tip-trick-enabling-ssl-on-iis7-using-self-signed-certificates.aspx>. Accessed 15. July 2012.

Holst, C. 2011. Fundamental guidelines of e-commerce checkout design. URL: <http://uxdesign.smashingmagazine.com/2011/04/06/fundamental-guidelines-of-e-commerce-checkout-design/>. Accessed 6. July 2012.

Knowledgebase. 2011. Understanding payment gateways. URL: http://kb.worldsecuresystems.com/000/bc_68.html. Accessed 10. August 2012.

Lawrence, E. 2012. Introducing Fiddler. URL: <http://www.fiddler2.com/fiddler2/>. Accessed 24. June 2012.

MintLife. 2011. Cracking the Credit Card Code. URL: <http://www.mint.com/blog/trends/credit-card-code-01202011/>. Accessed 29. June 2012.

Msdn . 2012. Unsafe (C# Reference). URL: <http://msdn.microsoft.com/en-us/library/chfa2zb8.aspx>. Accessed: 28.May 2012.

Msdn. 2012. fixed Statement (C# Reference). URL: <http://msdn.microsoft.com/en-us/library/f58wzh21.aspx>. Accessed: 28 May 2012.

Msdn. 2011. Regular Expression Language - Quick Reference. URL: <http://msdn.microsoft.com/en-us/library/az24scfc.aspx>. Accessed 29. June 2012.

Msdn. 2006. Regulatory Compliance Demystified: An Introduction to Compliance for Developers. URL: <http://msdn.microsoft.com/en-us/library/aa480484.aspx>. Accessed 06. July 2012.

PayPal. 2012. Documentation Tools. URL: <https://www.x.com/developers/paypal/documentation-tools/paypal-sdk-index>. Accessed 08. 08 2012.

Paypal. 2012. Payflow Link User's Guide . URL: https://cms.paypal.com/cms_content/AU/en_AU/files/developer/PP_PayflowLink_FPS_Guide.pdf. Accessed 30 June 2012.

Paypal. 2009. Payflow Pro Developer's Guide. URL:
https://cms.paypal.com/cms_content/CA/en_US/files/developer/PP_PayflowPro_Guide.pdf. Accessed 02. July 2012.

PCI Security Standards Council. 2010. Payment Card Industry (PCI) Data Security Standard. Requirements and Security Assessment Procedures . PCI Security Standards Council.

PHP. 2012. cURL Introduction. URL:
<http://www.php.net/manual/en/intro.curl.php>. Accessed 13. July 2012.

PHP. 2012. SQL Injection. URL: <http://php.net/manual/en/security.database.sql-injection.php>. Accessed 10. August 2012.

Ragan, D. 2009. Payment Processors - What do I need to know if I want to accept credit cards on my website? URL:
<http://stackoverflow.com/questions/51094/payment-processors-what-do-i-need-to-know-if-i-want-to-accept-credit-cards-on>. Accessed 10. August 2012.

Schneider, J. 2011. Electronic Commerce 9th edition. Cengage Learning.

SSL shopper. 2010. Do I need an SSL certificate for my website? URL:
<http://www.sslshopper.com/article-do-i-need-an-ssl-certificate-for-my-website.html>. Accessed 10. August 2012.

Visa. 2012. Visa's Global Registry of Service Providers - PCI DSS Validated Entities. URL: <http://usa.visa.com/download/merchants/cisp-list-of-pcidss-compliant-service-providers.pdf>. Accessed 7. July 2012.

Webopedia. 2012. Do It Yourself SSL Guide. URL:
http://www.webopedia.com/DidYouKnow/Internet/2008/SSL_02.asp. Accessed 05. May 2012.

Webopedia. 2012. SSL. URL: <http://www.webopedia.com/TERM/S/SSL.html>. Accessed 22. 05 2012.

Wickramasinghe, P. 2011. How to configure SSL on particular pages of an IIS 7 website. URL: <http://onlinecoder.blogspot.fi/2011/02/how-to-configure-ssl-on-particular.html>. Accessed 15. July 2012.

Wikipedia. 2012. Application Programming Interface. URL: http://en.wikipedia.org/wiki/Application_programming_interface. Accessed 20. May 2012

Wikipedia. 2012. Application Programming Interface. URL: http://en.wikipedia.org/wiki/Application_programming_interface. Accessed 24. May 2012.

Wikipedia. 2012. Card Security Code. URL: http://en.wikipedia.org/wiki/Card_security_code. Accessed 29. June 2012.

Wikipedia. 2012. DNS Spoofing. URL: http://en.wikipedia.org/wiki/DNS_spoofing. Accessed 10. August 2012.

Wikipedia. 2012. Extended Validation Certificate. URL: http://en.wikipedia.org/wiki/Extended_Validation_Certificate. Accessed 10. August 2012.

Wikipedia. 2012. LAMP (Software Bundle). URL: [http://en.wikipedia.org/wiki/LAMP_\(software_bundle\)](http://en.wikipedia.org/wiki/LAMP_(software_bundle)). Accessed 10. August 2012.

Wikipedia. 2012. List of TCP and UDP port numbers. URL: http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers. Accessed 15. July 2012.

Wikipedia. 2012. Password Strength. URL: http://en.wikipedia.org/wiki/Password_strength. Accessed 09. May 2012.

Wikipedia. 2012. Payment Gateway. URL: http://en.wikipedia.org/wiki/Payment_gateway. Accessed 5. May 2012.

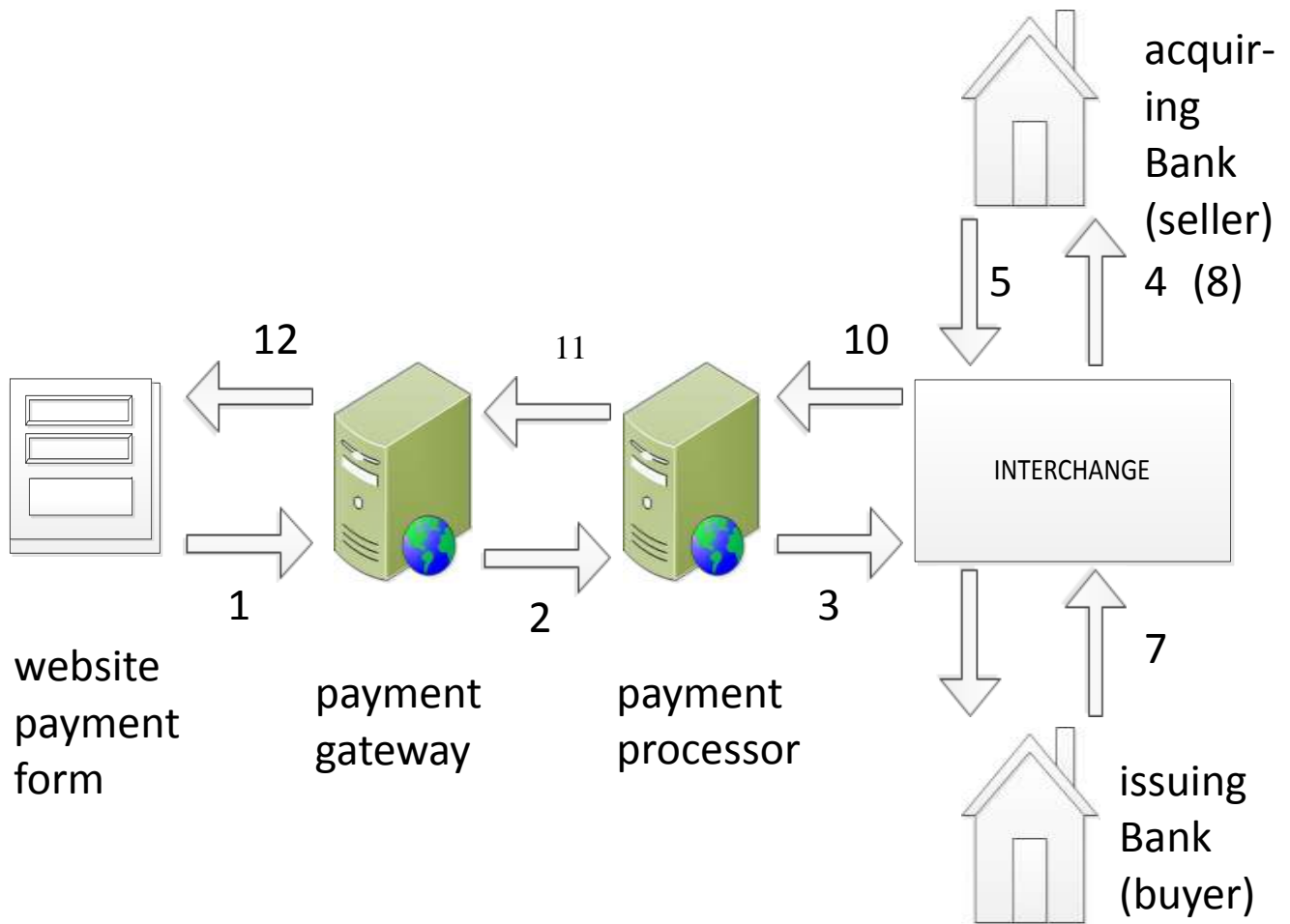
Wikipedia. 2012. Public-Key Cryptography. URL: http://en.wikipedia.org/wiki/Public-key_cryptography. Accessed 10. August 2012.

World Bank. 2011. The global outlook in summary, 2010-2014. URL:
<http://web.worldbank.org/external/default/main?theSitePK=659149&pagePK=2470434&contentMDK=20370107&menuPK=659160&piPK=2470429>. Accessed 6. July 2012.

World Wide Web Consortium. 2012. URL:
<http://www.w3.org/TR/html401/interact/forms.html#h-17.3>. Accessed 15. June 2012.

9 Appendices

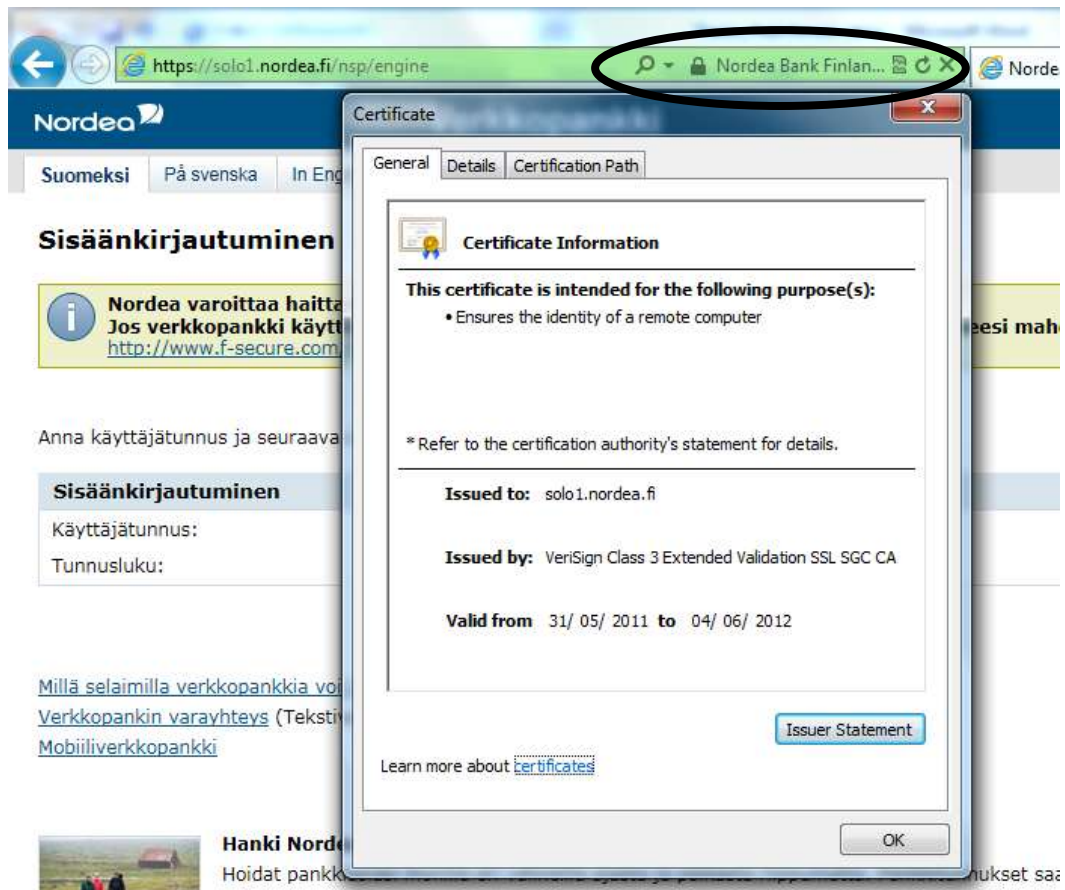
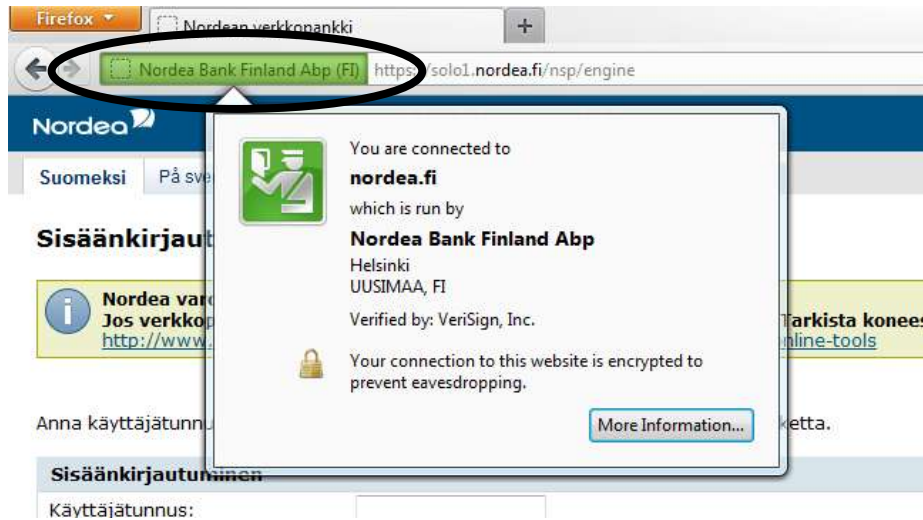
9.1 Appendix 1 - Diagram of the payment authorization process



9.2 Appendix 2 – Recognisable Certificate Authority brand logos



9.3 Appendix 3 - Current browser interface SSL connection and certificate notifications



9.4 Appendix 4 – Payment Application Best Practices (PABP)

1. Do not retain full magnetic stripe or CVV2 data

Do not store sensitive authentication data after transaction authorization (not even if encrypted).

2. Protect stored data

Mask sensitive cardholder data when displayed and when stored. Protect encryption keys against disclosure and misuse, and implement key management processes and procedures, including the generation, distribution, management, and storage of secure keys.

3. Provide secure password features

Require a unique username and complex password for all users with access to computers, servers, and databases with payment applications, including administrative access and especially access to cardholder data. Application passwords should be complex and if possible, encrypted.

4. Log application activity

Log all computer and network access by individual users, and implement functionality to link those activities to individual users. Implement an automated audit trail to track and monitor access.

5. Develop secure applications

Develop Web software and applications based on secure coding guidelines and industry best practices. Emphasize information security throughout the software development life cycle and routinely review custom application code to identify possible vulnerabilities.

6. Protect wireless transmissions

Securely encrypt all wireless transmissions of cardholder data over both public and private networks.

7. Test applications to address vulnerabilities

Establish a process to regularly test applications and identify potential security vulnerabilities. Develop and deploy security patches in a timely and secure manner.

8. Facilitate secure network implementation

Implement payment applications in a secure network environment. The applications should not interfere with the use of network address translation, port address translation, traffic filtering network devices, anti-virus solutions, patch or update installation, or hardware or software encryption.

9. Cardholder data must never be stored on a server connected to the Internet

Do not configure a database server and Web server to reside on the same server or in the “demilitarized zone” (DMZ) with the Web server.

10. Facilitate secure remote software updates

If software updates are delivered via remote access into customers’ systems, instruct customers to provide access to the system only when needed and to disable the connection immediately after downloads are complete. Alternatively, if delivered via virtual private network (VPN) or other secure connection, software vendors should advise customers to properly configure a personal firewall to secure “always-on” connections.

11. Facilitate secure remote access to applications

If employees, administrators, or vendors can access the application remotely, access should be authenticated using a 2-factor authentication mechanism. The application should allow for technologies with tokens, or VPN with individual certificates.

12. Encrypt sensitive traffic over public networks

Use strong cryptography and encryption techniques (at least 128 bit) such as SSL, Point-to-Point Tunneling Protocol (PPTP), or Internet Protocol Security (IPSEC) to safeguard sensitive cardholder data during transmission over public networks. Never send cardholder information via unencrypted e-mail.

13. Encrypt all non-console administrative access

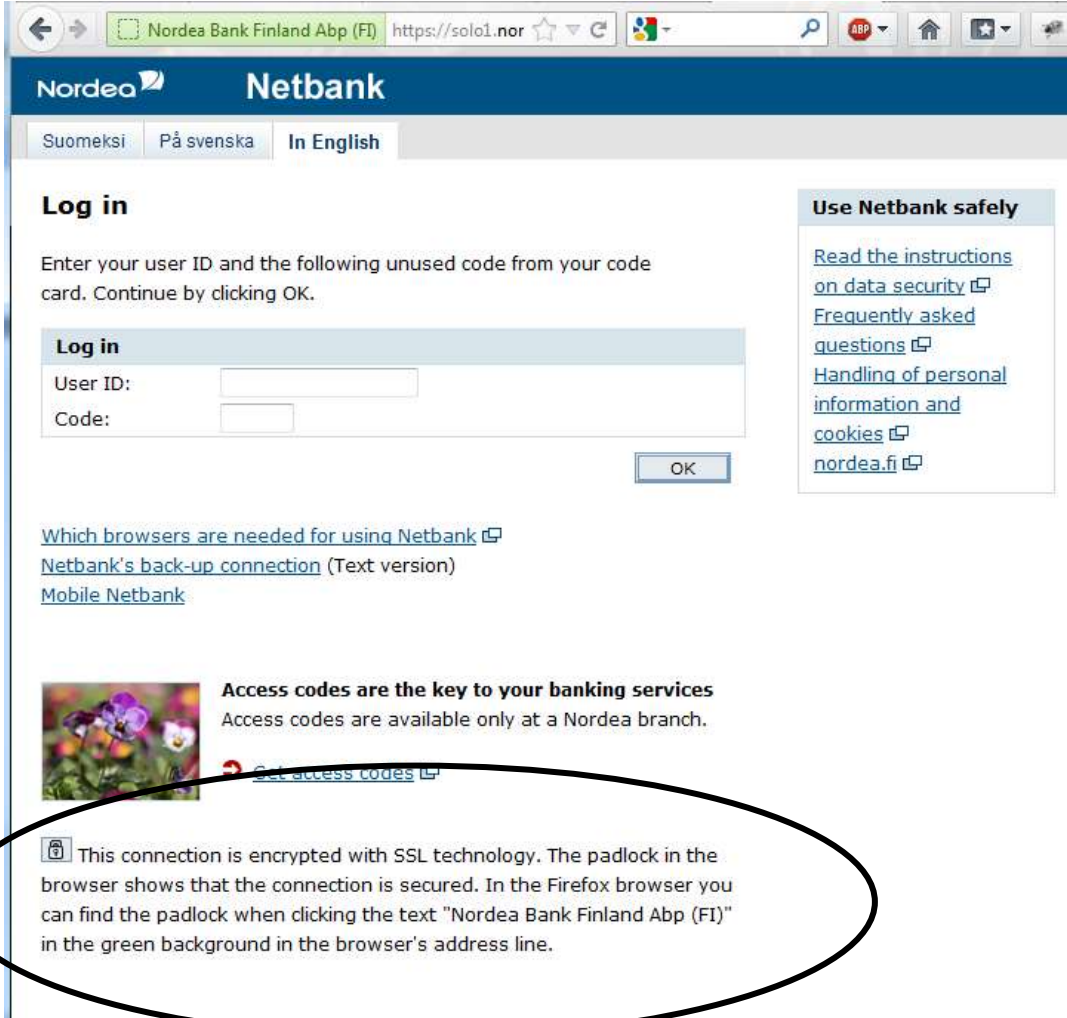
Use appropriate technologies for Web-based management and other non-console administrative access. Telnet or remote login must never be used for administration.

(Authorize.Net, 2008, pp. 4-6)

9.5 Appendix 5 – Rendered HTML payment buttons



9.6 Appendix 6 – Nordea Bank’s SSL information to users



The screenshot shows the Nordea Netbank login interface. At the top, the browser address bar displays "Nordea Bank Finland Abp (FI)" and "https://solo1.nor". The page header includes the Nordea logo and "Netbank" title, with language options for "Suomeksi", "På svenska", and "In English".

The main content area features a "Log in" section with the instruction: "Enter your user ID and the following unused code from your code card. Continue by clicking OK." Below this is a form with fields for "User ID:" and "Code:", and an "OK" button.

To the right, a "Use Netbank safely" sidebar contains links for "Read the instructions on data security", "Frequently asked questions", "Handling of personal information and cookies", and "nordea.fi".

Below the login form, there are links for "Which browsers are needed for using Netbank", "Netbank's back-up connection (Text version)", and "Mobile Netbank".

A section titled "Access codes are the key to your banking services" includes a small image of flowers and the text: "Access codes are available only at a Nordea branch." Below this is a link "Get access codes".

A security notice is circled in black, stating: "This connection is encrypted with SSL technology. The padlock in the browser shows that the connection is secured. In the Firefox browser you can find the padlock when clicking the text 'Nordea Bank Finland Abp (FI)' in the green background in the browser's address line."

9.7 Appendix 7 – Effective payment page design example


Checkout


step 1/2**B**





*** Your name**
The software license will be made out in this name.

*** E-mail address**
We'll send the receipt to this e-mail address.


Company name (optional)
If you want your company name on the invoice, just add it here.

 **Secure credit card payment**
This is a secure 128-bit SSL encrypted payment.



*** Credit card number**
The 16 digits on the front of your credit card.
    

*** Expiration date**
The date your credit card expires. Find this on the front of your credit card.
01 / 11

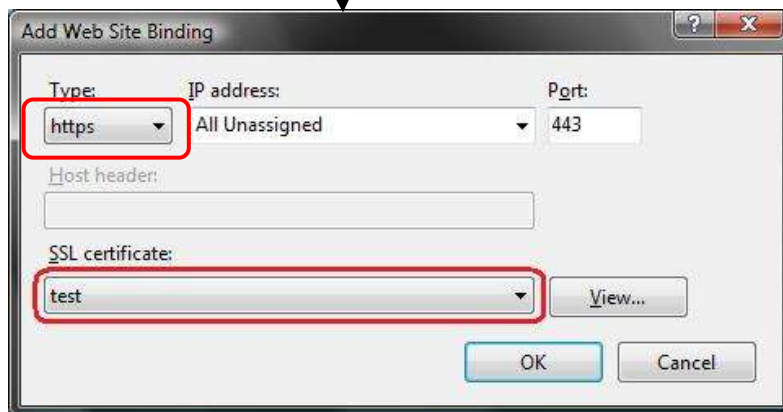
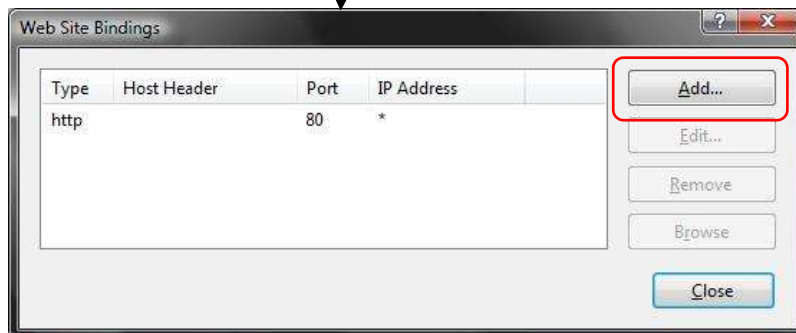
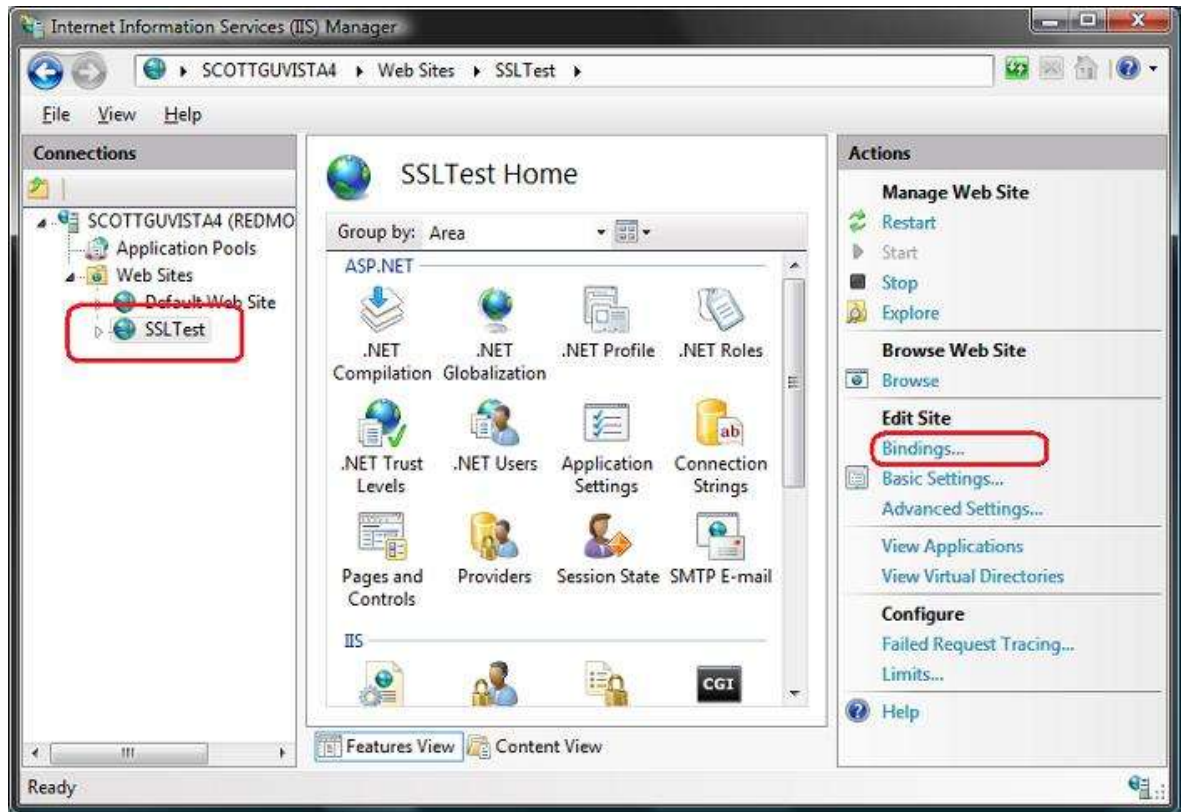
*** Security code** (or "CVC" or "CVV")
The last 3 digits displayed on the back of your credit card.
 

What happens now?
This is step 1 of 2. On the next page you can review your cart and product information. We will not bill you until you confirm the order on the next page.

[Next step »](#)

(Holst, 2011)

9.8 Appendix 8 – Implementing SSL in IIS 7.0 Windows Server 2008



(Guthrie, 2007)