Secure Coding.
Practical steps to defend your web apps.

Copyright SANS Institute
Author Retains Full Rights

This paper is from the SANS Software Security site. Reposting is not permitted without express written permission.

Interested in learning more?
Check out the list of upcoming events offering " ()"
at http://software-security.sans.orghttp://software-security.sans.org/events/
How to Audit ASP.NET Applications for PCI DSS Compliance

GIAC GSSP-NET Gold Certification

Author: Christian J. Moldes

Christian_moldes@hotmail.com

Adviser: Rodney Caudle

Accepted: October 23, 2011

Abstract

According to the 2010 Data Breach Investigation Report published by Verizon Business (2010), 40% of all the data breaches were the result of hacking attacks, out of that 40%, 54% were related to web applications. Application security remains one of the key factors in avoiding a security breach. PCI DSS (Payment Card Industry Data Security Standard) recognizes this and specific requirements have been outlined to ensure that companies have processes in place to ensure that applications are developed, deployed, and maintained securely. This paper intends to provide specific guidance to audit ASP.NET applications and verify that they meet PCI DSS requirements.
1. **Scope**

This paper intends to provide specific guidance on how to audit ASP.NET applications and validate that they meet PCI DSS requirements. It does not intend to provide guidance on how to conduct penetration tests on ASP.NET applications, identify secure coding vulnerabilities, or remediate ASP.NET vulnerabilities.

In order to work, applications rely on several other components that are external to the application such as infrastructure servers, network devices, databases, HTTP servers, and an OS platform. These components are critical to the security of the application; however, they were not included in this paper to limit the scope exclusively to the components that are directly under the control of the application.

This paper assumes that the reader is already familiar with ASP.NET coding, web applications’ architecture, web applications vulnerabilities, and the PCI DSS standard.

The audience for this paper are QSAs (Qualified Security Assessors), PA-QSAs (Payment Application – Qualified Security Assessors), compliance directors, IT auditors, and anyone responsible for PCI DSS compliance, and is particularly relevant for individuals who need to establish that an ASP.NET application is compliant with PCI DSS.

2. **Legal Disclaimer**

This paper contains many references to online tools and applications that can be used to conduct a PCI DSS audit. The author does not take responsibility for the contents of the websites mentioned in this paper nor does he provide any assurance that the websites or tools suggested are free of malware. The reader should make sure those resources are free of malware before browsing to the websites or using the tools on a production environment.

3. **PCI Security Standards Council and PCI DSS**

PCI Security Standards Council (PCI SCC) was founded by American Express, Discover Financial Services, JCB International, MasterCard Worldwide, and Visa Inc. in 2006.

Christian J. Moldes, christian_moldes@hotmail.com
Companies processing, storing or transmitting cardholder data from any of these payment brands have to comply with PCI DSS (Payment Card Industry Data Security Standard), a data security standard published and maintained by PCI SCC (PCI SCC, 2011a).

Non-compliant companies are exposed to higher transaction fees imposed by their acquirer banks, fines imposed by the payment card brands, higher liability in the event of a security breach, and even risk losing the authorization to process payment card transactions.

PCI SCC also published the PA-DSS standard (Payment Application Data Security Standard), which is a standard targeted toward vendors that develop and support payment applications. This paper is not focused on PA-DSS because this standard only applies to payment application vendors, its applications, and the process they have implemented to support their applications. As a matter of fact, most of the PA-DSS requirements applicable to applications are a subset of PCI DSS and reference its requirements. Hence, there is no added benefit to discussing PA-DSS requirements in this paper since most requirements would have already been discussed for PCI DSS.

4. ASP.NET Applications security and PCI DSS

It would be a mistake to assume that organizations, applications or IT infrastructure in general, would be secure by just meeting PCI DSS requirements. While PCI DSS is a security standard, at its core is a compliance standard that defines the acceptable risk of the council members. Security controls have been defined based on the most common vulnerabilities that affected companies in the past, and what PCI SSC considers would address most of those vulnerabilities and risk. In most cases, entities could obtain a less costly security posture by implementing security controls that fit their own individual companies rather than implementing all the controls that are required to be compliant with PCI DSS.

Unfortunately, while PCI DSS allow some flexibility, it does require a company to meet all their requirements as written or to implement compensating controls that go above and beyond what the standard already requires.

Christian J. Moldes, christian_moldes@hotmail.com
For payment applications, one of the key security controls that have the most impact on their PCI DSS compliance is the adoption of an industry best practice standard for application security. In previous versions of the standard, OWASP Top 10 was the only recommendation. As such, it has been widely adopted by companies and vulnerability management products and it has become the default standard for web application security.

With the release of PCI DSS v.2.0, PCI SSC suggested two additional standards: CWE/SANS Top 25 and CERT secure coding. As the date of this paper, CERT secure coding standard only provides guidance for applications developed using Java, C++, and C. Hence, companies using ASP.NET can only use either OWASP Top 10 Web Application Security Risks or CWE/SANS Top 25 Software Errors for secure coding guidance.

The main difference between OWASP Top 10 and CWE/SANS Top 25 is that OWASP Top 10 covers general concepts and is focused exclusively on web applications while CWE/SANS Top 25 covers specific software errors and could be applied to any type of software applications.

PCI DSS v.2.0 also expanded requirement 6.5 to include not only web applications but also all types of applications. Companies developing applications using mixed architectures may benefit by adopting and using a common standard across all their development teams. In this case CWE/SANS Top 25 may be the best option.

An important thing to remember is that OWASP Top 10 is updated every three years and CWE/SANS Top 25 is updated annually. In either case, PCI DSS requires entities to use the most current release of the standards, so an individual auditing applications for PCI DSS compliance should track changes on the standards at least annually.

References regarding how to audit specific software errors and security risks for each standard are included in the secure coding section.

5. **PCI DSS Requirements that Apply to Web Based Applications**

The following table lists PCI DSS v.2.0 requirements that apply to applications or
requirements that are directly under the control of the application. They have been grouped into categories to facilitate the audit process.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.3.7 Place system components that store cardholder data (such as a database) in an internal network zone, segregated from the DMZ and other untrusted networks.</td>
<td>Architecture Audit</td>
</tr>
<tr>
<td>2.2.1 Implement only one primary function per server to prevent functions that require different security levels from co-existing on the same server. (For example, web servers, database servers, and DNS should be implemented on separate servers.)</td>
<td>Architecture Audit</td>
</tr>
<tr>
<td>3.2 Do not store sensitive authentication data after authorization (even if encrypted). Sensitive authentication data includes the data as cited in the following Requirements 3.2.1 through 3.2.3:</td>
<td>Design Audit</td>
</tr>
<tr>
<td>3.3 Mask PAN when displayed (the first six and last four digits are the maximum number of digits to be displayed).</td>
<td>Design Audit</td>
</tr>
<tr>
<td>3.4 Render PAN unreadable anywhere it is stored (including on portable digital media, backup media, and in logs) by using any of the following approaches: One-way hashes based on strong cryptography (hash must be of the entire PAN) Truncation (hashing cannot be used to replace the truncated segment of PAN) Index tokens and pads (pads must be securely stored) Strong cryptography with associated key-management processes and procedures</td>
<td>Design Audit</td>
</tr>
<tr>
<td>3.5 Protect any keys used to secure cardholder data against disclosure and misuse: <strong>Note:</strong> This requirement also applies to key-encrypting keys used to protect data-encrypting keys—such key-encrypting keys must be at least as strong as the data-encrypting key.</td>
<td>Architecture Audit</td>
</tr>
<tr>
<td>4.1 Use strong cryptography and security protocols (for example, SSL/TLS, IPSEC, SSH, etc.) to safeguard sensitive cardholder data during transmission over open, public networks. Examples of open, public networks that are in scope of the PCI DSS include but are not limited to: The Internet Wireless technologies, Global System for Mobile communications (GSM) General Packet Radio Service (GPRS).</td>
<td>Design Audit</td>
</tr>
<tr>
<td>6.3 Develop software applications (internal and external, and including web-based administrative access to applications) in accordance with PCI DSS (for example, secure authentication and logging), and based on industry best practices. Incorporate information security throughout the software development life cycle. These processes must include the following:</td>
<td>Secure Coding</td>
</tr>
</tbody>
</table>

**Requirements from 6.3.1 thru 6.3.2**

| 6.5 Develop applications based on secure coding guidelines. Prevent common coding | Secure Coding |

Christian J. Moldes, christian_moldes@hotmail.com
How to Audit ASP.NET Applications for PCI DSS Compliance

Christian J. Moldes, christian_moldes@hotmail.com

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>vulnerabilities in software development processes, to include the following:</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** The vulnerabilities listed at 6.5.1 through 6.5.9 were current with industry best practices when this version of PCI DSS was published. However, as industry best practices for vulnerability management are updated (for example, the OWASP Guide, CWE/SANS Top 25, CERT Secure Coding, etc.), the current best practices must be used for these requirements.

6.6 For public-facing web applications, address new threats and vulnerabilities on an ongoing basis and ensure these applications are protected against known attacks by **either** of the following methods:

- Reviewing public-facing web applications via manual or automated application vulnerability security assessment tools or methods, at least annually and after any changes
- Installing a web-application firewall in front of public-facing web applications

7.1 Limit access to system components and cardholder data to only those individuals whose job requires such access. Access limitations must include the following:

**Requirements from 7.1.1 thru 7.1.4**

7.2 Establish an access control system for systems components with multiple users that restricts access based on a user’s need to know, and is set to “deny all” unless specifically allowed.

This access control system must include the following:

**Requirements from 7.2.1 thru 7.2.3**

8.4 Render all passwords unreadable during transmission and storage on all system components using strong cryptography.

**Design Audit**

8.5 Ensure proper user identification and authentication management for non-consumer users and administrators on all system components as follows:

**PCI DSS Requirements from 8.5.1 thru 8.5.16**

10.1 Establish a process for linking all access to system components (especially access done with administrative privileges such as root) to each individual user.

**Design Audit**

10.2 Implement automated audit trails for all system components to reconstruct the following events:

**Requirements from 10.2.1 thru 10.2.7**

10.3 Record at least the following audit trail entries for all system components for each event:

**Requirements from 10.3.1 thru 10.3.6**

**Table 1:** List of PCI DSS requirements that apply to ASP.NET applications.
6. **Architecture Audit**

PCI DSS Requirement 1.3.7 states that databases should be placed in an internal security zone, separated from the DMZ where web servers or other components externally accessed reside. Since Microsoft solutions are based on open technologies it should not be an issue for an application’s database to be located on a different network segment unless IP addresses and/or hostnames have been hardcoded.

In general, applications that use a multi-tier architecture in which presentation, business logic, and data management are processes running on logically separated systems can be more secure than applications that only relay on one or two tiers. i.e. By using web services, applications can became immune to many vulnerabilities that have affected them in the past such as SQL Injection and OS Command Injection. These types of vulnerabilities are not currently possible to exploit on systems using web services, as they only accept parameterized input. In addition, by having these tiers running on different logically separated systems, a stricter firewall policy can be applied to all the different tiers, limiting further penetration and escalation of privileges in the event that the most exposed tiers have been compromised.

In those cases, an attacker would be limited to the web server and the data residing on or being transmitted through the web server.

Web applications that directly access the database using SQL queries or that rely on stored procedures may need additional controls applied on the other tiers.

6.1. **Verifying servers only have one primary role**

PCI DSS requirement 2.2.1 emphasizes the need to implement only one function per server, thus, per requirement 1.3.7 and 2.2.1, a compliant application cannot have a web server and database residing on the same logical system.

The auditor has to inspect all systems, identify all services running, and obtain the IP addresses in use. A system hosting different services on the same logical system would not
How to Audit ASP.NET Applications for PCI DSS Compliance

Christian J. Moldes, christian_moldes@hotmail.com

qualify as having only one primary role, even if those services are published using different network interfaces.

6.2. Verifying databases are not located on the DMZ

Per requirement 1.3.7, any situation in which the system faces an insecure network such as the Internet or Extranet is incompatible with the database server role. The auditor has to obtain a dataflow diagram, identify all the locations where cardholder data is stored, and verify that the IP addresses for the web server and other components such as the database server are located on different security zones. In addition, the auditor should verify that firewall rules restrict connections between the web server and other components to only the services and protocols required by the application.

It is important to understand that by database, PCI DSS means a data repository. An application may use other types of data repositories besides relational databases. A good understanding of all the applications inputs and outputs should be gained during interviews and documentation review. Some examples of repositories are temporary files, transaction logs (TLOGs), error logs, trace files, and mail data files.

6.3. Verifying cryptographic keys are adequately protected

According to PCI DSS, applications should have at least two set of cryptographic keys: Data-Encrypting Keys (DEKs) and Key-Encrypting Keys (KEKs). DEKs and KEKs should be stored separately. As of the date of this paper, PCI Council does not provide specific guidance regarding what defines “stored separately”. Separately could be understood to mean not located in the same file, or not located in the same type of file, or not located on the same system. At least make sure that DEKs and KEKs are not located on the same type of file. Keep in mind that the interpretation of this requirement may differ among PCI QSAs and that the most conservative interpretation is to store DEKs and KEKs on separate systems.

A note on PCI DSS requirement 3.5 clearly states that both DEKs and KEKs should have the same level of protection. Therefore, since DEKs have to be encrypted so do KEKs.
This seems like a never-ending cycle; however, PCI DSS only mentions these two sets of keys (DEKs and KEKs), so if another set of cryptographic keys are used to protect KEKs, there is no need to encrypt them. The following examples show cases of cryptographic keys that has been deemed compliant in the past.

Example 1: A DEK can be stored in a configuration file encrypted with the KEK. The KEK can be stored on a different system and encrypted with the application credentials or with a value hard-coded in the application source code and obfuscated in order to avoid any strings searches.

Example 2: A DEK can be stored on the application server in a configuration file encrypted with the KEK. The KEK could be stored on a database table encrypted with encryption functionality provided by the database.

At any rate, DEKs and KEKs should not be hardcoded as this will complicate key rotation and would allow development staff to know the encryption key value. This would obviously affect PCI DSS compliance with requirement 3.6.6, as there is no business need for a developer to know cryptographic values on production systems.

The auditor should find out where cryptographic keys are located by conducting interviews, and if possible by inspecting source code. The latter will be the only way to verify whether cryptographic keys are encrypted using strong cryptography, as there are many ways to render keys unreadable that do not meet the intent of the standard. For commercial applications where source code is not available, the auditor should review the application’s PA-DSS implementation guide, administrators’ manuals, and/or end-user manuals. If the information is not available, the application vendor should be contacted to obtain additional details.

Note that .NET provides functionality to create key containers on which asymmetric keys can be stored. Key containers can be created at the machine-level or user-level.

There are two types of key containers: User-level and machine-level RSA key containers. User-level RSA key containers are stored with the Windows user profile for a particular user and can be used to encrypt and decrypt information for applications that run
under that specific user identity. User-level RSA key containers can be useful if a developer want to ensure that the RSA key information is removed when the Windows user profile is removed. However, because the user must be logged in with the specific user account that will make use of the user-level RSA key container in order to encrypt or decrypt protected configuration sections, they are inconvenient to use (MSDN Library, 2011a).

Machine-level RSA key containers are available to all users that can log in to a computer, by default, and are the most useful as they can be used to encrypt or decrypt protected configuration sections while logged in with an administrator account. A machine-level RSA key container can be used to protect information for a single application, all the applications on a server, or a group of applications on a server that run under the same user identity. Although machine-level RSA key containers are available to all users, they can be secured with NTFS Access Control Lists (ACLs) so that only required users can access them (MSDN Library, 2011a).

In either case, whether machine-level or user-level, verify that proper NTFS access controls lists have been set.

Applications can use many other key management solutions including homegrown or commercial key management solutions, which are out of scope for this paper.

Applications applying for PA-DSS compliance should not have hard-coded DEKs or KEKs. PCI SSC published specific guidance about this subject. If the application is compromised, all the companies using the same application would be in risk until a new version of the application addressing the compromised encryption keys is deployed (PCI SSC, 2010)

7. **Design Audit: Data at Rest**

Sensitive authentication data cannot be stored after authorization as per PCI DSS requirement 3.2. However, it is understood that data may be stored temporarily while being in the authorization process, especially during store-and-forward transactions. The best practice

Christian J. Moldes, christian_moldes@hotmail.com
for this data is to handle the data only in memory, if possible. If that is not an option, at least the same level of security should be applied for sensitive authentication data than the one applied to PANs.

Primary account numbers (PANs) should be rendered unreadable while at rest per PCI DSS requirement 3.4 using strong cryptography by using any of the following approaches:

- One-way hashes based on strong cryptography (hash must be of the entire PAN)
- Truncation (hashing cannot be used to replace the truncated segment of PAN)
- Index tokens and pads (pads must be securely stored)
- Strong cryptography with associated key-management processes and procedures

Truncated PAN numbers and index tokens are not considered cardholder data, and the applications that only used that type of data and have no access to the original PAN number are considered out of scope. This paper will focus on the other two options available, which are encryption and one-way-hashing, both acceptable if they are based on strong cryptography.

7.1. Strong Cryptography

PCI SCC (2001b) defines strong cryptography as the cryptography based on industry-tested and accepted algorithms, along with strong key lengths and proper key-management practices. Examples of accepted encryption algorithms include AES (128 bits and higher), TDES (Minimum double-length keys), RSA (1024 bits and higher), ECC (160 bits and higher), and ElGamal (1024 bits and higher).

PCI DSS bases its strong cryptography recommendation on NIST Special Publication 800-57. Therefore, the auditor should become familiar with the three parts of this standard. Part 1 provides general guidance and best practices for the management of cryptographic
keying material (NIST, 2007a), part 2 provides guidance on policy and security planning requirements for U.S. government agencies (NIST, 2007b), and part 3 provides guidance when using the cryptographic features of current systems (NIST, 2009).

As the date of this paper, NIST has a new draft version of Special Publication 800-57 Part 1 that has been published for comments and review. Keep track of this publication and take into consideration that guidance regarding key management and strong cryptography may change in the near future.

Section 7.3 details how to determine whether strong cryptography is in use.

7.2. Finding Cardholder Data in Clear Text

One way to ensure that a system does not store cardholder data in clear text is to run a tool that searches the entire file system and identifies files that contain PANs.

A tool such as dnGrep (Grep for Windows) can be used for that purpose. Enter regular expressions that matches card numbers and search inside all types of files. The current version of dnGrep can search text files, archives, MS Word, and PDF documents (Google, 2011). Spider from Cornell University (Cornell University, 2007) can also be used to find cardholder data on all accessible files. Commercial solutions that have proven to be successful in finding card numbers are also available. PowerGrep, for example, is capable of searching data on text, binary, Microsoft Word, Microsoft Excel, PDF, and OpenOffice files (Just Great Software Co., 2011). Use at least a couple of tools, and conduct controlled tests to validate the tools capabilities and regular expressions used to search card numbers in clear text before using them on a production environment.

Goyvaerts’ regular-expressions.info website details regular expressions that can be used to search for card numbers in clear text (Goyvaerts, 2011).

7.3. Is Data Encrypted?

Unless source code is inspected, it is very difficult to validate that data has been
rendered unreadable using strong cryptography. In order to visually recognize whether data is encrypted, it is advisable to know how encrypted data, encoded data, and hashed data looks.

Several websites provide online functionality to encrypt, decrypt, encode, and hash data for test purposes such as Yellowpipe.com and Chilkatsoft.com. An auditor can use those sites to validate claims that data is encrypted, especially when after visual inspection clues are obtained that the data may not be encrypted.

The following table lists the results of using different techniques to render a test card number unreadable. Some of the techniques are not compliant and applications using them would have to implement compensating controls in order to achieve PCI DSS compliance.
## How to Audit ASP.NET Applications for PCI DSS Compliance

**Christian J. Moldes, christian_moldes@hotmail.com**

### Data Format

<table>
<thead>
<tr>
<th>Data Format</th>
<th>Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Card number</td>
<td>541234567901234</td>
<td>Card number in clear text. This is not an acceptable technique to store cardholder data. Compensating controls would have to be implemented to achieve PCI DSS compliance.</td>
</tr>
<tr>
<td>Hexadecimal</td>
<td>133A7FED97AFF2</td>
<td>Hexadecimal is a base 16 encode technique. The resulting value should only contain numbers from 1 to 9, 0, and characters from A to F. This is not an acceptable technique to render cardholder data unreadable, as it would be very easy to reverse hexadecimal values to card numbers.</td>
</tr>
<tr>
<td>Base 64 Encode</td>
<td>NTQxMjM0NTY3ODkwMTIzNA==</td>
<td>One characteristic of base 64 are the trailing characters “==” and the use of uppercase and lowercase characters. This is not an acceptable technique to render cardholder data unreadable, as it would be very easy to reverse encoded values to card numbers.</td>
</tr>
<tr>
<td>DES</td>
<td>CR4m2sG47zvQ2</td>
<td>DES is no longer deemed strong cryptography and should not be used to render cardholder data unreadable. This is not an acceptable technique to render cardholder data unreadable.</td>
</tr>
<tr>
<td>MD5 (Hash)</td>
<td>3e90cee97bf4a7aaafadf118449084f9</td>
<td>The resulting value of hashing data is a hexadecimal string. An auditor may infer that a value has been hashed if the value only contains numbers from 1 to 9, 0, and characters from A to F. The alphabetic characters not necessarily should be stored as lowercase characters. A MD5 hash produces a string of 32 characters. This is an acceptable technique to render cardholder data unreadable according to PCI SSC; however, strongly not recommended. According to Integrigy (2007), “Storing of credit card numbers by simply hashing only the card number is unacceptable and can be easily compromised by brute force methods.” (p.13) PCI SSC only recommends the use of salt. In our opinion, the use of salt should be strictly required for this technique to be acceptable.</td>
</tr>
</tbody>
</table>
### Data Format | Value | Comments
--- | --- | ---
SHA-1 (Hash) | ea2c3c903f2d47540514c2502d5e016780b28422 | The resulting value of hashing data is a hexadecimal string. An auditor may infer that a value has been hashed if the value only contains numbers from 1 to 9, 0, and characters from A to F. The alphabetic characters not necessarily should be stored as lowercase characters.
A SHA-1 hash produces a string of 40 characters.
This is an acceptable technique to render cardholder data unreadable according to PCI SSC; however, strongly not recommended. According to Integrigy (2007), “Storing of credit card numbers by simply hashing only the card number is unacceptable and can be easily compromised by brute force methods.” (p.13)
PCI SSC only recommends the use of salt. In our opinion, the use of salt should be strictly required for this technique to be acceptable.

SHA-256 (Hash) | 1A8D8B311E0DA854A59B31F1148056AE69980C667B2EB4CFFB31F5508FCF760D | The resulting value of hashing data is a hexadecimal string. An auditor may infer that a value has been hashed if the value only contains numbers from 1 to 9, 0, and characters from A to F. The alphabetic characters not necessarily should be stored as uppercase characters as we seen in previous examples.
A SHA-256 hash produces a string of 64 characters.
This is an acceptable technique to render cardholder data unreadable according to PCI SSC; however, strongly not recommended. According to Integrigy (2007), “Storing of credit card numbers by simply hashing only the card number is unacceptable and can be easily compromised by brute force methods.” (p.13)
PCI SSC only recommends the use of salt. In our opinion, the use of salt should be strictly required for this technique to be acceptable.

SHA-512 (Hash) | AF7822A5F1E4BE8B27B136BB007D6CDA3C563BB2592E5702BB0DB60B724BBD8CE127DAC7405811CACF404AFBB8D3519BA926F0C458DCDECCF8E402FFA2EDE | The resulting value of hashing data is a hexadecimal string. An auditor may infer that a value has been hashed if the value only contains numbers from 1 to 9, 0, and characters from A to F. The alphabetic characters not necessarily should be stored as uppercase characters as we seen in previous examples.
A SHA-512 has produces a string of 128 characters.
This is an acceptable technique to render cardholder data unreadable according to PCI SSC; however, strongly not recommended. According to Integrigy (2007), “Storing of credit card numbers by simply hashing only the card number is unacceptable and can be easily compromised by brute force methods.” (p.13)
### How to Audit ASP.NET Applications for PCI DSS Compliance

Christian J. Moldes, christian_moldes@hotmail.com

#### Data Format

<table>
<thead>
<tr>
<th>Data Format</th>
<th>Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES 128 Bits AES / ECB / PKCS5 Padding (Encrypted Text in base 64)</td>
<td>2VLwiHgOoZEe2/UkS90fEQ==</td>
<td>PCI SSC only recommends the use of salt. In our opinion, the use of salt should be strictly required for this technique to be acceptable. Encryption produces values that sometimes cannot be stored in databases easily nor rendered easily on the screen. That is why most of the time they are encoded using Base 64. Note the trailing characters &quot;==&quot; which reveals that the string more than likely is encoded. Also, note the use of uppercase and lowercase values. This is an acceptable method to render cardholder data unreadable.</td>
</tr>
<tr>
<td>AES 128 Bits</td>
<td>6XA7orXT+I4zav7TKcAFSXDXAcHi6ubO89E+15xVe+y=</td>
<td>Note that encrypted values which in some cases will include characters, numbers, and symbols such as &quot;=&quot;, &quot;+&quot;, and &quot;/&quot;. This is an acceptable method to render cardholder data unreadable.</td>
</tr>
</tbody>
</table>

**Table 2: Different techniques to render cardholder data unreadable**
7.4. Inspecting IIS logs

IIS logs should be inspected to verify that cardholder data is not logged. Whenever applications use HTTP GET methods for form submissions, depending on the IIS log settings, data could be logged to the IIS logs files. An easy way to identify whether HTTP GET methods are in use is to observe the URL when submitting forms. If the URL includes a question mark symbol and parameters separated by ampersands symbols, HTTP GET is being used. While the use of HTTP GET instead of HTTP POST methods during posts is not an issue, if cardholder data is being posted using parameters, log files may contain cardholder data in clear text.

To locate the log files, start off by verifying the IIS configuration since the location of the log files may have been changed from the default location to a different directory or even a different system.

Log into the web server, go to the Start Menu, select Control Panel, select Administrative Tools, and select Internet Information Services (IIS). Find the website under the tree on the left. Right click on the website entry and choose properties. On the Web site tab, there is an option near the bottom that says “Active Log Format”. Click on the Properties button. At the bottom of the General Properties tab, there is a box that contains the log file directory and the log file name.

Open the log files and inspect the URI Query column, which is displayed as cs-uri-query on the IIS logs. Even if logging URI Query has not been enabled, potentially an administrator or even an attacker may enable URI query logging once a system has been compromised. Therefore, HTTP GETs methods should not be used to post cardholder data.

One way to verify whether this risk exists at all, would be to interact with the application using a Web Debugging Proxy such as Fiddler or IE Inspector HTTP Analyzer to capture the traffic at the time cardholder data is posted to the web server, and inspect what data is being posted using parameters.
7.5. **Inspecting log4net logs**

Log4j is a very popular way to generate logs among Java developers. A similar tool, log4net is available for .NET and more developers are incorporating this tool into their ASP.NET applications.

If Log4net is being used, search the web server file system for the Log4net configuration file, `log4net.Config.XmlConfigurator` and inspect the value for the `level value` attribute. Log4net can be configured to send logs not only to a local file but to several different local and remote databases as well (Apache Software Foundation, 2011).

There are no limitations regarding the type of information that can be logged in Log4net logs since they are created programmatically. Logging behavior can be drastically changed by modifying settings in Log4net configuration files. The auditor should verify that even the most verbose log levels such as debug do not log any cardholder data on those logs. Test should be conducted to replicate all possible scenarios with payment card transactions using the most verbose log level, and logs should be inspected after each test.

7.6. **Inspecting other logs**

The ASP.NET architecture works on a pipeline model. An HTTP request goes through the ASP.NET pipeline passing through HTTP modules, which examine the request and can act upon it, change it, rewrite the URL, or even throw exceptions, which stops any further progression and returns an error to the user (Dorrans, 2010). One of the most common uses of HTTP modules is to implement security features such as authentication and authorization.

Developers can create HTTP modules to inspect and more importantly log all the requests coming from users. For an application that process or transmits cardholder data, those logs may potentially contain sensitive data if the application has not been properly designed.

HTTP modules are registered in the `web.config` file for a machine or website. Examine the website’s `web.config` file located on the website directory and the machine’s `web.config`
file contained in the following directory:

```
C:\Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG
```

The actual log file could be located on a local or remote system. In both cases, inspect the `<httpModules>` section and request additional information for entries that have been added to the file.

```
<httpModules>
  <add name="LogModule"
       Type="MyHttpModule.LogModule, MyHttpModule"/>
  // other modules registered by default
</httpModules>
```

For additional information regarding HTTP Modules read “Securely Implement Request Processing, Filtering, and Content Redirection with HTTP Pipelines in ASP.NET” (Ewald & Brown, 2003) and “HTTP Handlers and HTTP Modules Overview” (Microsoft, 2011b).

### 7.7. Requirement 3.3. Verifying PAN Masking and Truncation

PCI DSS requirement 3.3 specifies that PANs should be masked when displayed and that a maximum of the first 6 and last 4 digits of the PAN can be displayed.

To verify proper masking the auditor has to identify all application functionality that deals with cardholder data and whether there is any way to enable displaying of full PANs on the screens and/or reports. If the application provides such a functionality, the access control systems in place to limit access to full PANs should be reviewed. For user roles or user IDs that have been granted the capability to see full PANs, a legitimate business need should exists. File-search utilities using regular expression should be run on systems where reports are generated to find any reports and files that may contain full PANs.

### 8. Design Audit: Data in Transit

The PCI DSS standard requires applications that transmit cardholder over open or public networks to encrypt the transmission using strong cryptography. For data in transit this
implies using SSL/TLS or IPSEC connections.

IPSEC connections, which are obviously external to the application and web server, can be verified by reviewing the firewall or VPN concentrator configurations and ensuring the traffic from the server is routed over those systems. Network and security device reviews are out of scope for this paper.

For SSL connections, it is necessary to identify whether load balancers and/or SSL off load devices are being used. In those cases, SSL connections may end on the load balancer or SSL off load device. If that is the case, the auditor should review the load balancer or SSL off load device configuration as well.

8.1. **Verifying that only trusted keys and/or certificates are accepted.**

PCI DSS Web servers use their digital certificates to establish a SSL/TLS connection with clients. As such, there is no way a client could use an untrusted key or certificate to establish a connection to the web server. This requirement would apply to SSH or IPSEC connections though, which are out of scope for this paper.

8.2. **Verifying that the protocol is implemented to use only secure configurations, and does not support insecure versions or configurations.**

In most production environments, SSL connections end at load balancers or SSL off load devices; therefore, the web server may not be the right place to validate secure configuration for SSL connections. The auditor should review the load balancer or SSL off load device instead. As mentioned previously, network and security device reviews are out of scope for this paper.

Most network vulnerability scanners include features to test whether insecure protocol versions are supported. OWASP provides detailed black and white box testing procedures to validate a website only accepts secure versions of SSL/TLS (OWASP, 2011a)

If load balancers or SSL off load devices are not in use, the auditor should check the...
web server registry keys in order to verify that insecure versions are not supported. The following registry keys should have a DWORD subkey named “Enabled” with the following value: 00000000

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\PCT 1.0\Server

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\SSL 2.0\Server

8.3. Verifying that proper encryption strength is implemented for the encryption methodology in use.

As in the previous requirement, the auditor can verify proper encryption strength by inspecting the web server configuration, if the SSL connections ends at the web server. The best approach to verify proper encryption regardless the use of load balancers or SSL off load devices, is by using tools to attempt to connect to the website using different encryption algorithms and strengths. Most network vulnerability scanners already include this as part of their vulnerability scanning process. OWASP provides detailed black and white box testing procedures to validate a website only accepts connections using proper encryption strength (OWASP, 2011a).

In order to verify that weak encryption algorithms have been disabled on the web server, the auditor should check that the following registry keys have a DWORD subkey named “Enabled” with the following value: 00000000

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers\DES 56/56

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers\NULL

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers\RC2 40/128

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers\RC2 56/128

Christian J. Moldes, christian_moldes@hotmail.com
8.4. Verifying cardholder data is encrypted during transit.

There are several approaches to verify that cardholder data is encrypted when transmitted over public, open networks. It can be verified by observing that HTTPS appears as a part of the browser URL when cardholder data is required. In addition, the auditor can install a network sniffer on their own system, interact with the website and capture packets while placing an order and/or entering test card data into the website. Packet captures should be inspected to verify that card numbers were not sent in clear text.

Since PCI DSS requires selecting a sample of transactions as they are received, it may make more sense to capture the packets on the server side of a production environment. Make sure that packets are being captured before the SSL/TLS connection ends. Wireshark, one of the most popular network sniffers, can be used to monitor and save the packet captures to a text file. Once the file has been obtained, a tool that uses regular expressions should be run to quickly validate that packets do not contain any full PAN numbers or sensitive authentication data in clear text.


9.1. Verifying password encryption, password policies, and lockout policies

Applications may use two types of credentials, credentials used to control access to the application features, and credentials used by the application to have access to resources such as databases and files commonly known as service or application accounts.

ASP.NET applications can use authentication modes based on Windows, Forms, and
Passport. Forms authentication is a set of .NET libraries and ASP.NET controls that allow an application to authenticate users. Even though these libraries and controls are available, developers may have developed proprietary authentication forms outside of these libraries. If developers did use these features, PCI DSS compliance is greatly facilitated as password and lockout policies can be easily configured by modifying attributes settings on the `web.config` file, membership section. Inspect the `web.config` file and verify the current setting for the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>PCI DSS Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxInvalidPasswordAttempts</td>
<td>The number of maximum password attempts allowed before a user account is locked out.</td>
<td>Per PCI DSS req. 8.5.13 the value has to be set to no more than six invalid logon attempts.</td>
</tr>
<tr>
<td>minRequiredNonAlphanumeric Characters</td>
<td>The number of special characters that must be present in a password</td>
<td>Not required; however unless regular expressions are used to validate password complexity, it may be the only way to guarantee compliance with PCI DSS req. 8.5.11. This would have the unwanted result of requiring a mixture of letter casings as well as non-alphanumeric characters, which obviously goes far beyond what is required. ASP.NET by default set this value to 1.</td>
</tr>
<tr>
<td>minRequiredPasswordLength</td>
<td>The minimum length of a valid password.</td>
<td>Per PCI DSS req. 8.5.10 the value has to be set to at least seven characters. ASP.NET by default sets this value to 8 characters.</td>
</tr>
<tr>
<td>passwordStrengthRegularExpression</td>
<td>Specifies a regular expression used to validate password strength.</td>
<td>This is the recommended way to enforce PCI DSS Req. 8.5.11, which requires passwords to contain both numeric and alphabetic characters. The other way is to set the minRequiredNonAlphanumeric Characters, although that would go far beyond what is required.</td>
</tr>
</tbody>
</table>

Christian J. Moldes, christian_moldes@hotmail.com
### Table 3: Membership Provider Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>PCI DSS Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>passwordFormat</td>
<td>Specifies the format of the stored password. This value can be set to Clear,</td>
<td>Per PCI DSS req. 8.4, passwords have to be rendered unreadable using strong cryptography.</td>
</tr>
<tr>
<td></td>
<td>Hashed, or Encrypted.</td>
<td>Encrypted and Hashed compliant with PCI DSS.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASP.NET by default sets this value to Hashed.</td>
</tr>
<tr>
<td>passwordAnswerAttemptLockoutDuration</td>
<td>Specifies the duration, in minutes, that a lockout due to a bad password answer is considered still in effect.</td>
<td>Per PCI DSS req. 8.5.14 to a minimum of 30 minutes or until administrator enables the user ID.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASP.NET by default sets this value to 30 minutes.</td>
</tr>
<tr>
<td>requireSSL</td>
<td>Specifies whether cookie requires SSL in order to be returned to the server. If cookies are not transmitted over a SSL connection, they could be captured and tampered with while crossing the network.</td>
<td>Per PCI DSS req. 6.5.4 this value should be set to True.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASP.NET by default sets this value to False, which is not PCI DSS compliant.</td>
</tr>
</tbody>
</table>

ASP.NET does not provide features to implement password aging (PCI DSS Req. 8.5.9) nor password history (PCI DSS Req. 8.5.12). These features would have to be implemented by developers. Interview developers to find out how these requirements are being met.

Note that PCI DSS states that password and lockout policies are only required for non-consumer users and administrators. It is not intended for these policies to be applied to consumer users since they only enter their own credit card information into the website.

Windows authentication relies on Windows accounts and passwords, and authentication is handled by IIS and the browser. Windows password and lockout policies should be inspected if they are being used for authentication. PCI DSS compliance would
depend on the IIS authentication method selected since there are some authentication methods that are not compliant.

Passport authentication has been deprecated. Do not expect this to be used as deprecated technologies are usually no longer supported by the vendor.

To verify which authentication mode is being used, inspect the web.config files for a machine and the ones located in the website directory. Once the XML file is open, search for the attribute “authentication mode”. A value of “Windows” indicates that .NET is configured to allow IIS to handle the authentication. Follow these steps to check the authentication scheme inside IIS.

1. Launch the IIS management application
   - Click “Start” button
   - Click “Run”
   - Type in “inetmgr”
   - Click “OK”

2. Navigate to the Authentication properties of the web-site
   - Expand the tree titled “Sites” on the left-hand side
   - Click on the web site in question
   - The right-hand pane should be populated with a series of icons
   - Click on the icon labeled “Authentication”

The following two tables lists the different authentication providers and authentication schemes supported by ASP.NET. Pro and cons columns were populated using information from Microsoft MSDN Library for ASP.NET authentication (Microsoft, 2011c) and IIS Authentication (Microsoft, 2011d).
## Authentication Provider

<table>
<thead>
<tr>
<th>Authentication Provider</th>
<th>Description</th>
<th>Pro</th>
<th>Con</th>
<th>PCI DSS Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>The Windows authentication provider relies upon IIS to perform the required authentication of a client. After IIS authenticates a client, it passes a security token to ASP.NET.</td>
<td>• Authenticated using Windows accounts, so developers do not need to write any custom authentication code.</td>
<td>• May require the use and management of individual Windows user accounts.</td>
<td>There are several types of Windows Authentication, consult the IIS Authentication Scheme Table in order to identify which schemes are compliant.</td>
</tr>
</tbody>
</table>
| Forms                   | The Forms authentication provider is an authentication scheme that makes it possible for the application to collect credentials using an HTML form directly from the client. The client submits credentials directly to the application code for authentication. If the application authenticates the client, it issues a cookie to the client that the client presents on subsequent requests. When authenticating credentials, the application can store credentials in a number of ways, such as a configuration file or a SQL Server database. | • Makes it possible for custom authentication schemes using arbitrary criteria.  
• Can be used for authentication or personalization.  
• Does not require corresponding Windows accounts. | • Is subject to replay attacks for the lifetime of the cookie, unless using SSL/TLS.  
• Is only applicable for resources mapped to Aspnet_isapi.dll. | Compliant if properly implemented.  
ASP.NET 3.5 and later provides a membership management service.  
Inspect the `web.config` file and verify the values for the following attributes: protection, timeout, requireSSL, passwordFormat, minRequiredPasswordLength.  
These attributes have the potential to affect requirements 8.4 (passwords unreadable during transmission and storage), 8.5.10 (Minimum password length), and 8.5.15 (Session timeout)  
For example if requireSSL has been set to False, passwords would be transmitted in clear text. If passwordFormat has been set to ‘Clear’, passwords would be stored in clear text.  
Microsoft provides additional information regarding ASP.NET membership management service on MSDN website (MSDN, 2011e) | |
| Passport                | Uses Microsoft Passport for the authentication process. This setting is deprecated. | • None.                                                                | • Deprecated.                                                        | Do not expect this one to be in use as it has been deprecated. |
How to Audit ASP.NET Applications for PCI DSS Compliance

Christian J. Moldes, christian_moldes@hotmail.com

<table>
<thead>
<tr>
<th>Authentication Scheme</th>
<th>Description</th>
<th>Pro</th>
<th>Con</th>
<th>PCI DSS Compliance</th>
</tr>
</thead>
</table>
| Anonymous             | Anonymous authentication gives users access to the public areas of a Web site without prompting them for a user name or password. Although listed as an authentication scheme, it is not technically performing any client authentication because the client is not required to supply any credentials. Instead, IIS provides stored credentials to Windows using a special user account, IUSR_machinename. By default, IIS controls the password for this account. | - Offers the best performance because Anonymous authentication imposes no appreciable overhead.  
- Does not require management of individual user accounts.  
- If IIS does not control the password, can access network resources. | - Does not authenticate clients on an individual basis.  
- If IIS does not control the password, account must have local logon ability. | Not compliant if the application is used to access cardholder data. It would not meet PCI DSS requirement 8.2 |

Table 3 - ASP.NET Authentication Modes

This mode is not recommended. Compliant if a custom-built authentication scheme has been properly implemented. A thorough review of the implementation would be required to verify that the authentication scheme meets all applicable PCI DSS requirements.
## Authentication Scheme

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Description</th>
<th>Pro</th>
<th>Con</th>
<th>PCI DSS Compliance</th>
</tr>
</thead>
</table>
| **Basic** | IIS implements Basic authentication, which is part of the HTTP 1.0 specification, using Windows user accounts. When using Basic authentication, the browser prompts the user for a user name and password. This information is then transmitted across HTTP where it is encoded using Base64 encoding. Although most Web servers, proxy servers, and Web browsers support Basic authentication, it is inherently insecure. Because it is easy to decode Base64 encoded data, Basic authentication is essentially sending the password as plain text. | ▪ Because it is part of the HTTP 1.0 specification, Basic is the most widely supported user authentication scheme.  
▪ Can authenticate through a proxy server.  
▪ Makes it possible to track individual users.  
▪ Can access network resources, if user account has local logon rights on the Web server.  
▪ Can be used in conjunction with Kerberos, enabling delegation of security credentials. | ▪ Is inherently insecure unless using SSL/TLS, which may affect performance.  
▪ Requires the creation of individual Windows accounts for each user. | Not compliant unless SSL/TLS is used.  
Without SSL/TLS, it would not meet PCI DSS requirement 8.4 |
| **Digest** | Digest authentication addresses the primary weaknesses of basic authentication: sending passwords in plain text. Digest authentication is a challenge/response mechanism, which sends a digest (also known as a hash) instead of a password over the network. A digest is a fixed-size result obtained by applying a mathematical function (called a hash function or digest algorithm) to an arbitrary amount of data. The fixed-size depends upon the level of encryption. When a client attempts to access a resource requiring Digest authentication, IIS sends a challenge. | ▪ Sends a digest over the network instead of a password.  
▪ Works with proxy servers and firewalls.  
▪ Does not require SSL/TLS for the sake of password protection. | ▪ Cannot delegate security credentials.  
▪ Is only supported by Internet Explorer 5.0 and later.  
▪ Is subject to replay attacks unless SSL/TLS is used.  
▪ Requires storge of passwords in clear text using reversible encryption.  
▪ Requires the creation of individual Windows accounts for each user. | Not compliant unless SSL/TLS is used.  
**OWASP Top 10**  
Without SSL/TLS, it would not meet OWASP Top 10 A3 and/or OWASP Top 10 A9, resulting in not being compliant with PCI DSS requirement 6.5.  
**CWE/SANS Top 25**  
Without SSL/TLS, it would not
How to Audit ASP.NET Applications for PCI DSS Compliance

Christian J. Moldes, christian_moldes@hotmail.com

<table>
<thead>
<tr>
<th>Authentication Scheme</th>
<th>Description</th>
<th>Pro</th>
<th>Con</th>
<th>PCI DSS Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digest Authentication</td>
<td>to the client to create a digest and send it to the server. The client concatenates the password with data known to both the server and the client. The client then applies a digest algorithm (specified by the server) to the combined data. The client sends the resulting digest to the server as the response to the challenge. The server uses the same process as the client to create a digest using a copy of the client's password it obtains from Active Directory, where the password is stored using reversible encryption. If the digest created by the server matches the digest created by the client, IIS authenticates the client.</td>
<td>of domain accounts for each user in Active Directory.</td>
<td>meet CWE-311, resulting in not being compliant with PCI DSS requirement 6.5.</td>
<td></td>
</tr>
<tr>
<td>Integrated Windows Authentication</td>
<td>Integrated Windows authentication (formerly known as NTLM authentication and Windows NT Challenge/Response authentication) can use either NTLM or Kerberos V5 authentication and only works with Internet Explorer 2.0 and later. When Internet Explorer attempts to access a protected resource, IIS sends two WWW-Authenticate headers, Negotiate and NTLM. If Internet Explorer recognizes the Negotiate header, it will choose it because it is listed first. When using Negotiate, the browser will return information for both NTLM and Kerberos. At the server, IIS</td>
<td>Can be used in conjunction with Kerberos, enabling delegation of security credentials. Integrated Windows authentication is the best authentication scheme in an intranet environment where users have Windows domain accounts, especially when using Kerberos. Integrated Windows authentication, like digest authentication, does not pass the user's password across the network. Instead, a</td>
<td>Cannot authenticate through a firewall via a proxy, unless used over a PPTP connection. Does not support delegation to other servers, if NTLM is chosen. It is only supported by Internet Explorer 2.0 and later. Kerberos is only supported by IIS 5.0 and later.</td>
<td>Not compliant when using NTLM authentication since it has been deemed insecure, resulting in PCI DSS requirement 2.2 not being met. When using Kerberos V5 and properly implemented, this scheme could be compliant.</td>
</tr>
</tbody>
</table>
### Authentication Scheme

<table>
<thead>
<tr>
<th>Authentication Scheme</th>
<th>Description</th>
<th>Pro</th>
<th>Con</th>
<th>PCI DSS Compliance</th>
</tr>
</thead>
</table>
| Kerberos              | will use Kerberos if both the client (Internet Explorer 5.0 and later) and server (IIS 5.0 and later) are running Windows 2000 and later, and both are members of the same domain or trusted domains. Otherwise, the server will default to using NTLM.  
- If Internet Explorer does not understand Negotiate, it will use NTLM.  
Therefore, which mechanism is used depends upon a negotiation between Internet Explorer and IIS. When used in conjunction with Kerberos v5 authentication, IIS can delegate security credentials among computers running Windows 2000 and later that are trusted and configured for delegation. Delegation enables remote access of resources on behalf of the delegated user. IIS uses SSL/TLS to authenticate a server and provide an encrypted HTTP session. IIS can also use SSL/TLS to authenticate the client by requiring the client to provide a certificate. When requesting a client certificate, the server provides the client with a list of CAs that the server trusts. This list is derived from the server's Certificate Trust List (CTL). If the client possesses a certificate issued by a CA from the CTL, it sends a copy of that certificate to the server for verification. If the certificate is valid, IIS authenticates the user | hashed value is exchanged. |  |
| Client Certificate Mapping | Includes strong authentication scheme.  
- Provides two-way authentication of server and client.  
- Can access network resources. | Cannot delegate security credentials.  
- Does not work with all browsers.  
- Requires SSL/TLS.  
- Is cumbersome to configure; however, many-to-one can be easier than one-to-one. | Not compliant when using many-to-one mappings as it will not meet PCI DSS requirement 8.1  
One-to-one mappings could be compliant if properly implemented. |
## Authentication Scheme

<table>
<thead>
<tr>
<th>Description</th>
<th>Pro</th>
<th>Con</th>
<th>PCI DSS Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>that maps to the provided certificate. Operating systems such as Windows still require the notion of a user account. Certificate mapping makes it possible for administrators to associate a single certificate (one-to-one mapping), or multiple certificates (many-to-one), to a user account. Many-to-one mapping uses rules to define certificate criteria for mapping.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 4 - IIS Authentication Schemes**
The other type of credentials an application may use are service or application credentials, which may be stored in configuration files such as `machine.config`, `web.config`, `log4net.Config.XmlConfigurator` (if Log4net is being used), databases tables, or the system registry.

ASP.NET provides functionality to encrypt sections of a `web.config` configuration file and automatically decrypt them when needed. By using the `aspnet_regiis`, a section in the `web.config` file can be encrypted. While this is useful to render credentials unreadable and meet PCI DSS requirement 8.4, anyone with local system access will therefore potentially be able to decrypt these sections by using the same command.

Nevertheless, Aspnet_regiis meets requirement 8.4 as it renders passwords unreadable using strong cryptography, through the RSA (Rivest, Shamir, Ademan) algorithm. Microsoft provides detailed information on how to protect configuration files on the MSDN website (Microsoft, 2011f).

Credentials can be stored on the registry securely by using the `aspnet_setreg` command. This command was used as a workaround for sensitive data stored on the `web.config` file before the release of `aspnet_regiis`. This approach or similar based on the same concept can still be used to store sensitive data in the registry. For additional information on how to use `aspnet_setreg`, consult the Microsoft Support Site, “How to use the ASP.NET utility to encrypt credentials and session state connection strings” (Microsoft, 2007).

`Aspnet_setreg` uses DPAPI and 3DES to encrypt data; therefore, it meets PCI DSS requirement 8.4. For additional information on DPAPI, consult MSDN Library, “Windows Data Protection” (Microsoft, 2001).

Credentials can also be stored on database records. A discussion of database security is out of scope for this paper.

Credentials should not be hardcoded. It is not a good practice because developers will then have credentials to access the production environment. In many assessments, it was
found that developers were using application and service accounts to support the application. Obviously, this does not provide any accountability as it would be difficult to track down which specific developer accessed the production environment. Furthermore, this practice would not meet PCI DSS requirement 8.5.8, which prohibits the use of group, shared or generic accounts for individuals.

There are several tools that can extract strings from executables and compiled code, so it would be very easy for an attacker to find out hardcoded credentials. PCI DSS does not provide any specific guidance regarding this, so technically as long as credentials are encrypted using strong encryption, it should not affect PCI DSS compliance.

Not all databases encrypt communication between the clients and the database by default. Auditors should verify that encryption has been properly configured; otherwise, credentials could be transmitted in clear text. Database security is out of scope for this paper.

### 9.2. Verifying the use of Identities and Impersonation

By default, ASP.NET runs under an account that has limited privileges; however, this could have been changed to provide an application higher privileges. Inspect the `machine.config` file and the values for the identity attributes in the `processModel` section. The following table lists possible values for the `userName` attribute:

<table>
<thead>
<tr>
<th>Values</th>
<th>Description</th>
<th>PCI DSS Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>machine</em></td>
<td>It forces ASP.NET to run under the system account (ASPNET or Network Service)</td>
<td>It is the most secure setting as it forces ASP.NET account to run under the fewest number of privileges possible.</td>
</tr>
<tr>
<td><em>system</em></td>
<td>It forces ASP.NET to run under the local SYSTEM account.</td>
<td>It has considerable more privileges to access networking and files than the privileges granted to <em>machine</em>.</td>
</tr>
<tr>
<td>Other Value</td>
<td>It uses the account and password provided in the configuration file.</td>
<td>It inherits all the privileges the specified account offers. Not compliant if the password is not encrypted.</td>
</tr>
</tbody>
</table>

**Table 5 – Identities**

Christian J. Moldes, christian_moldes@hotmail.com
ASP.NET applications can use impersonation to execute the application with the Windows account of the user making the request or under a specified account that it is declared using the `<identity>` element in the `web.config` file.

Inspect the `web.config` file, and verify that if impersonation is being used, passwords are encrypted as detailed in pervious sections.

Microsoft provides detailed information regarding ASP.NET impersonation on the MSDN Library website (Microsoft, 2011f).

9.3. **Verifying authorization and access controls**

In addition to the membership service, ASP.NET provides a role management service, which facilitates the implementation of roles and access controls. However, most of this functionality has to be implemented programmatically. Follow these steps to check whether ASP.NET roles are being used:

1. Launch the Internet Information Services (IIS) Manager application
   - Click “Start” button
   - Click “Run”
   - Type in “inetmgr”
   - Click “OK”

2. Inspect the .NET Users, .NET Roles, and .NET Authorization Rules settings for the web-site
   - Expand the tree titled “Sites” on the left-hand side
   - Click on the web site in question
   - The right-hand pane should be populated with a series of icons
   - Click on the icon labeled “.NET Users” for ASP.NET user accounts.
   - Click on the icon labeled “.NET Roles” for ASP.NET roles.
   - Click on the icon labeled “.NET Authorization Rules” for the authorization rules

Christian J. Moldes, christian_moldes@hotmail.com
assigned to rules and accounts.

In order to audit authorization and access controls, a deep understanding of a company’s business processes and the application’s role and functionality is needed.

10. **Design Audit: Logging**

ASP.NET provides several ways to log errors including logging to the Windows Event Log, sending logs by e-mail, using WMI (Windows Management Interface) Events, logging to a database, logging to a file, or using a framework such as log4net.

In all cases, logs should be programmatically created for the events PCI DSS requires to be logged. If Windows authentication is used, part of the required logs will be kept in the Windows event log.

Since some of the requirements are very specific, a good understanding of PCI DSS log requirements is critical.

10.1. **PCI DSS Requirements for Logging**

The following table lists the PCI DSS requirements that apply to logging:
## How to Audit ASP.NET Applications for PCI DSS Compliance

<table>
<thead>
<tr>
<th>PCI DSS Requirement</th>
<th>Description</th>
<th>Audit Procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1</td>
<td>Establish a process for linking all access to system components (especially access done with administrative privileges such as root) to each individual user.</td>
<td>This requirement is addressed externally to the application, usually by enabling logging on the system hosting the application. No action required from an application audit's perspective.</td>
</tr>
<tr>
<td>10.2.1</td>
<td>Implement automated audit trails for all system components to reconstruct all individual accesses to cardholder data</td>
<td>If the application provides access to cardholder data, audit trails should be enabled to reconstruct who accessed specific data. Requirement 10.3 details the information that each log entry should have. Inspect logs entries to verify that this requirement has been addressed and that logs entries contain all the information detailed in requirement 10.3</td>
</tr>
<tr>
<td>10.2.2</td>
<td>Implement automated audit trails for all system components to reconstruct all actions taken by any individual with root or administrative privileges</td>
<td>This requirement is addressed externally to the application, usually by enabling logging on the system hosting the application. Applications may have settings that if changed, would change the security behavior of the application. For example, disabling masking for PANs, or allowing unauthenticated access to the application, etc. In those cases, the application should also have audit trails to log any actions taken by an administrator. Inspect logs entries to verify that this requirement has been addressed.</td>
</tr>
<tr>
<td>10.2.3</td>
<td>Implement automated audit trails for all system components to reconstruct access to all audit trails</td>
<td>An audit trail entry should be created every time an individual access audit trails entries. Inspect logs entries to verify that this requirement has been addressed.</td>
</tr>
<tr>
<td>10.2.4</td>
<td>Implement automated audit trails for all system components to reconstruct invalid logical access attempts</td>
<td>An audit trail entry should be created every time an invalid logical access attempts occurs. Inspect logs entries to verify that this requirement has been addressed.</td>
</tr>
</tbody>
</table>

Christian J. Moldes, christian_moldes@hotmail.com
## How to Audit ASP.NET Applications for PCI DSS Compliance

<table>
<thead>
<tr>
<th>PCI DSS Requirement</th>
<th>Description</th>
<th>Audit Procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.2.5</td>
<td>Implement automated audit trails for all system components to reconstruct use of identification and authentication mechanisms</td>
<td>Audit trail entries should be created not only for invalid logical access attempts but also when someone successfully passed the authentication process. Inspect logs entries to verify that this requirement has been addressed by showing the use of identification and authentication mechanisms used during a successful authentication attempt.</td>
</tr>
<tr>
<td>10.2.6</td>
<td>Implement automated audit trails for all system components to reconstruct initialization of the audit logs</td>
<td>Per requirement 10.5.3, logs should be promptly backed up to a centralized log server or media that is difficult to alter. If that requirement is met by forwarding logs continuously, PCI DSS requirement 10.2.6 could be implemented on the centralized log repository. So, no further action is required from an application audit’s perspective. If that is not the case, an audit trail entry should be generated every time someone initializes the log. Additional security controls would be needed if the log file were kept outside the application’s control. It would be very easy to replace the file with a null file or with an altered copy of the file after purging audit trails, leaving the application unaware of the change. The application or the operating system should be able to detect any changes to the external local file. If this has not been implemented it will be very difficult to demonstrate that this requirement has been met. Inspect logs entries to verify that this requirement has been addressed. In the event that external files to the application are used, verify the audit trails enabled for that file on Windows or within the application.</td>
</tr>
<tr>
<td>10.2.7</td>
<td>Implement automated audit trails for all system components to reconstruct creation and deletion of system-level objects</td>
<td>This requirement is addressed externally to the application, usually by enabling logging on the system hosting the application. No action required from an application audit’s perspective. See comments on requirement 10.2.2 as well.</td>
</tr>
</tbody>
</table>

Christian J. Moldes, christian_moldes@hotmail.com
## How to Audit ASP.NET Applications for PCI DSS Compliance

Christian J. Moldes, christian_moldes@hotmail.com

### PCI DSS Requirement

<table>
<thead>
<tr>
<th>Description</th>
<th>Audit Procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use file-integrity monitoring or change-detection software on logs to ensure</td>
<td>This requirement is usually met by implementing a process to forward audit logs continuously as they are</td>
</tr>
<tr>
<td>that existing log data cannot be changed without generating alerts (although</td>
<td>generated to a centralized log repository, and implementing change-detection on that repository.</td>
</tr>
<tr>
<td>new data being added should not cause an alert).</td>
<td>It is very difficult to implement this requirement if the log is managed by the application on an external</td>
</tr>
<tr>
<td></td>
<td>file to the application. A file integrity monitoring solution will not be able to work properly on a file</td>
</tr>
<tr>
<td></td>
<td>that is changing continuously.</td>
</tr>
<tr>
<td></td>
<td>Even if audit trail entries are inserted into a database, it is difficult to detect that an individual entry</td>
</tr>
<tr>
<td></td>
<td>is missing from the audit trail table, unless database triggers are enabled to detect any alteration or</td>
</tr>
<tr>
<td></td>
<td>deletions on the audit trail table.</td>
</tr>
<tr>
<td></td>
<td>One other way to meet this requirement is keeping a Hash value of the audit trail entries, where the hash</td>
</tr>
<tr>
<td></td>
<td>record is obtained from the values of the current audit trail entry plus some values of the previous entry.</td>
</tr>
<tr>
<td></td>
<td>By doing this, it would be possible to detect that a previous entry has been deleted or changed.</td>
</tr>
<tr>
<td></td>
<td>A process should be implemented to periodically recalculate hashes and detect unauthorized changes;</td>
</tr>
<tr>
<td></td>
<td>otherwise, the application would have all the functionality to detect unauthorized changes but the</td>
</tr>
<tr>
<td></td>
<td>process to report the changes would never be run.</td>
</tr>
<tr>
<td></td>
<td>If the application is responsible for detecting changes to logs, observe audit trails and alerts entries</td>
</tr>
<tr>
<td></td>
<td>after an attempt to tamper with the log files.</td>
</tr>
</tbody>
</table>

**Table 6 – Audit Procedures for PCI DSS Log Requirements**
10.2. Error Handling

Error pages can provide useful information for developers and, unfortunately, for attackers as well. In order to verify that this does not occur, inspect the web.config file and verify that customErrors have been defined.

ASP.NET provides the following customer error modes off, on, and remoteOnly.

<table>
<thead>
<tr>
<th>Setting CustomErrors Mode</th>
<th>Description</th>
<th>PCI DSS Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>Specifies that custom errors are not enabled and that a page with detailed error information should be displayed in all cases.</td>
<td>Not compliant as this would provide potentially compromising information about a website to anyone who can cause an error to occur on the website.</td>
</tr>
<tr>
<td>On</td>
<td>Specifies that custom errors are enabled.</td>
<td>Compliant if implemented properly.</td>
</tr>
<tr>
<td>RemoteOnly</td>
<td>Specified that custom errors should be used for any external user accessing the website. For local users accessing the site from the web server itself a detailed error page is displayed.</td>
<td>Compliant if implemented properly.</td>
</tr>
</tbody>
</table>

Table 7 – CustomErrors Mode Settings

The following web.config excerpt shows custom errors properly configured:

```xml
<system.web>

    <CustomErrors mode="RemoteOnly"  
        defaultRedirect="/DefaultRedirectErrorPage.aspx" />

    <error statusCode="404"  
        redirect="/PageNotFound.aspx" />

    // Other Status Codes...

</system.web>
```

Not all errors would be raised at the page-level, application-level errors can also occur and they will not be handled by this custom error functionality. Microsoft provides detailed
information regarding how to manage error handling on the MSDN website (Microsoft, 2011h)

11. Secure Coding Audit

PCI DSS requirement 6.5 requires organizations to develop software applications using secure coding best practices such as OWASP, CWE/SANS Top 25, or CERT Secure Coding. In addition, PCI DSS section 6 also requires formal processes to review all custom code by individuals other than the author (PCI DSS req. 6.3.2), a formal change management process (PCI DSS req. 6.4), and a manual or automated application vulnerability security-assessment at least annually if web application firewalls have not been deployed (PCI DSS req. 6.6). In addition, PCI DSS requires and annual network and application penetration testing (PCI DSS req. 11.3.2).

For an organization meeting all these requirements, the auditor should have enough evidence to confirm that the application is free of application security risks (OWASP) and dangerous software errors (CWE/SANS Top 25). It is not the intent for the QSA or internal assessor to validate what should have already been validated during these previous tasks.

Nevertheless, auditors besides interviewing developers and observing documentation may need to conduct a quick check to validate that the code reviews, security assessments, vulnerability scans, and penetration tests have been able to uncover most of the security flaws and vulnerabilities listed on those standards.

The intent of the following sections is to provide basic guidelines and links to resources where to obtain additional information in the event that the auditor may need it.

The first table lists all the applicable PCI DSS requirements, a brief description of what it is being required, a reference to OWASP if that requirement is currently listed as one of the top 10 security risks, and a basic procedure to verify that applications compliant with that requirement.
### How to Audit ASP.NET Applications for PCI DSS Compliance

Christian J. Moldes, christian_moldes@hotmail.com

<table>
<thead>
<tr>
<th>PCI DSS Req.</th>
<th>Description</th>
<th>Standards Reference</th>
<th>Audit Procedure</th>
</tr>
</thead>
</table>
| 6.5.1        | Injection flaws, particularly SQL injection. Also consider OS Command Injection, LDAP and XPath injection flaws as well as other injection flaws. | OWASP Top 10 – A1 | Interview developers to identify the security controls implemented in the application to deal with injection vulnerabilities. Developers may be using one or more of the following options to protect applications against injection attacks.  
1) Prepared statements (Parameterized queries)  
2) Stored procedures  
3) Escaping all user supplied input  
4) Using LINQ  
5) Using web services  
Ferruh Mavituna’s website lists several samples of injection attacks the auditor may use to test an application (Mavituna, 2007). Commercial web vulnerability scanners such as Acunetix WBS, SPI Dynamics WebInspect, and IBM AppScan can detect these type of vulnerabilities. OWASP’s WebScarab can also be helpful to test this and other requirements (OWASP Foundation, 2011b). OWASP Top 10 – A1 audit procedures provides detailed steps on how to test for this vulnerability (OWASP Foundation, 2011c) |
<p>| 6.5.2        | Buffer overflow | - | In managed code such as VB or C#, numeric overflows cannot lead to buffer overflows as they can in unmanaged code (e.g., C or C++). This is due to the Common Language Runtime (CLR) handling the effects of the numeric overflow gracefully. Therefore, as long as unsafe or unmanaged code is not invoked, such as the use of P/Invoke or COM Interop, an ASP.NET application is not vulnerable to this type of security flaws. However, buffer overflows could affect the platforms on which web applications run. Network vulnerability scans and penetration tests reports should be clean of any buffer overflows vulnerabilities to pass this requirement. In the event that P/Invoke or COM Interop are used, OWASP provides test procedures for buffer overflow vulnerabilities (OWASP Foundation, 2008) |</p>
<table>
<thead>
<tr>
<th>PCI DSS Req.</th>
<th>Description</th>
<th>Standards Reference</th>
<th>Audit Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.5.3</td>
<td>Insecure cryptographic storage</td>
<td>OWASP Top 10 – A7</td>
<td>Audit procedures for this requirement were discussed in section 6.3 and sections 7 and 8 of this paper. OWASP Top 10 – A7 audit procedures provides detailed steps on how to test for this vulnerability (OWASP Foundation, 2011d)</td>
</tr>
<tr>
<td>6.5.4</td>
<td>Insecure communications</td>
<td>OWASP Top 10 – A9</td>
<td>Audit procedure for this requirement were discussed in section 8 of this paper. In section 9.1 of this paper, it was discussed the need to have requireSSL set to True. OWASP Top 10 – A9 audit procedures provides detailed steps on how to test for this vulnerability (OWASP Foundation, 2011e)</td>
</tr>
<tr>
<td>6.5.5</td>
<td>Improper error handling</td>
<td>-</td>
<td>Audit procedures for this requirement were discussed in section 10.2 of this paper.</td>
</tr>
<tr>
<td>6.5.6</td>
<td>All &quot;High&quot; vulnerabilities identified in the vulnerability identification process (as defined in PCI DSS Requirement 6.2).</td>
<td>-</td>
<td>An organization could become aware of newly security vulnerabilities through network vulnerability scans, penetration tests, and security newsletters. In order to verify compliance, for this requirement both network vulnerability scans and penetration tests should be clean of high vulnerabilities.</td>
</tr>
<tr>
<td>PCI DSS Req.</td>
<td>Description</td>
<td>Standards Reference</td>
<td>Audit Procedure</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>---------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>6.5.7</td>
<td>Cross-site scripting (XSS)</td>
<td>OWASP Top 10 – A2</td>
<td>Microsoft provides a library to deals with Cross-site Scripting (XSS), the Microsoft Anti-XSS Library. Interview developers to verify that the library is being used, and that it has been properly implemented. Examples of cross-site scripting attacks that can be copied and pasted into the application being tested are available on RSnake’s website (RSnake, 2008). One important thing to remember is that some web browsers may not be vulnerable to some types of attacks due to the implementation of anti cross-site scripting features. That does not mean that the application is not vulnerable to that type of attack; therefore, tests should include all most popular browsers. Microsoft provides additional details regarding the Anti-Cross Site Scripting Library on the MSDN website (Microsoft, 2011i). OWASP Top 10 – A2 audit procedures provides detailed steps on how to test for this vulnerability (OWASP Foundation, 2011f)</td>
</tr>
</tbody>
</table>
| 6.5.8       | Improper Access Control (such as insecure direct object references, failure to restrict URL access, and directory traversal) | OWASP Top 10 – A4 OWASP Top 10 – A8 | Insecure Direct Object References: In order to verify that this vulnerability does not exist, a web proxy such as Burp Suite, Paros, or WebScarab can be used to inspect POST and GET commands, and tamper with parameters and values as they are posted to the web site. Failure to Restrict URL Access: A good understanding of the application functionality, roles, and permissions is needed. Using unauthenticated and authenticated access and the website directory structure validate that critical URL and resources cannot be accessed directly. Directory Traversal / Path Traversal: According to Mitre (2011), “Automated techniques can find areas where path traversal weaknesses exist. However, tuning or customization may be required to remove or de-prioritize path-traversal problems that are only exploitable by the software’s administrator - or other privileged users - and thus potentially valid behavior or, at worst, a bug instead of a vulnerability. Manual white box techniques may be able to provide sufficient code coverage and reduction of false positives if all file access operations can be...
How to Audit ASP.NET Applications for PCI DSS Compliance

6.5.9 Cross-site request forgery (CSRF)

<table>
<thead>
<tr>
<th>PCI DSS Req.</th>
<th>Description</th>
<th>Standards Reference</th>
<th>Audit Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>OWASP Top 10 – A5</td>
<td></td>
</tr>
</tbody>
</table>

Assessed within limited time constraints.

The following automated tools may be helpful validating this requirement:

- DotDotPwn – The Directory Traversal Fuzzer, which is now included in the BackTrack penetration testing Linux distro (Chr1x, 2010)
- OWASP’s WebScarab can also be helpful to test this and other requirements (OWASP Foundation, 2011b).
- Commercial web vulnerability scanners such as Acunetix WBS, SPI Dynamics WebInsPect, and IBM AppScan can detect these type of vulnerabilities

OWASP published testing procedures for OWASP Top 10 A4 (OWASP Foundation, 2011g) and OWASP Top 10 A8 (OWASP Foundation, 2011h).

OWASP published a tool to test Cross-Site Request Forgery (CSRF) named CSRFTester, which could be very useful to test an application for this type of vulnerability (OWASP Foundation, 2011).

Developers may state that they are using ViewState, which is enabled by default; however, that may not be sufficient to deal with CSRF. Smolen (2008) discovered that ASP.NET anti-CSRF features do not work for non post-backs.

In order to verify that CSRF risks have been adequately treated, verify the following:

1) ViewState is enabled and that has not been disabled on the web.config files (site or directory) or programmatically for specific pages.
2) ViewSateUserKey is being used programmatically with a unique value per user/session.
3) ViewStateEncryptionMode has been set to Always or Auto if the
Table 8 – Audit Procedures for PCI DSS section 6

Out of the OWASP Top 10 list of most critical web application security flaws, seven have been addressed in a specific requirement by the current version of the PCI DSS Standard. However, per PCI DSS requirement 6.5, the current version of the best practices document should be used. In version 2010 of OWASP Top 10 security flaws there are still three security flaws that have not been addressed by PCI DSS directly. These flaws are listed in the following table.

<table>
<thead>
<tr>
<th>OWASP Top 10 Security Flaw</th>
<th>Description</th>
<th>Audit Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>A3 Broken Authentication and Session Management</td>
<td>Follow OWASP audit procedures for testing authentication (OWASP Foundation, 2010), and specifically for OWASP Top 10 – A3 (OWASP Foundation, 2011)</td>
<td></td>
</tr>
<tr>
<td>A6 Security Misconfiguration</td>
<td>This requirement is mostly related with the platform on which the application runs. Therefore, it is outside the scope for this paper. However, the following resources may be helpful to deploy a secure platform on which applications can run:</td>
<td></td>
</tr>
</tbody>
</table>

OWASP Top 10 – A5 audit procedures provides detailed steps on how to test for this vulnerability (OWASP Foundation, 2011k)
How to Audit ASP.NET Applications for PCI DSS Compliance

How to Audit ASP.NET Applications for PCI DSS Compliance

Christian J. Moldes, christian_moldes@hotmail.com

How to Audit ASP.NET Applications for PCI DSS Compliance

OWASP Top 10 Security Flaw | Description | Audit Procedure
--- | --- | ---
CIS (Center for Internet Security) provides secure configuration standards for Microsoft IIS web servers v.5.0, 6.0, and 7.0, and Windows OS 2003 and 2008 (Center for Internet Security, 2011)
As part of the IIS 6.0 Resource Kit, Microsoft published specific guidance on how to lock down an IIS 6.0 configuration, refer to Chapter 3 - Securing Web Sites and Applications (Microsoft, 2011j)
OWASP Top 10 – A6 audit procedures provides detailed steps on how to test for this vulnerability (OWASP Foundation, 2011m)

A10 | Unvalidated Redirects and Forwards | According to Mitre (2011), automated static analysis tools may not be able to determine whether input influences the beginning of a URL, which is important for reducing false positives. Automated black box tools that supply URLs to every input may be able to spot Location header modifications, but test case coverage is a factor, and custom redirects may not be detected.
Using white box techniques may be the best approach to validate compliance for this security flaw.
OWASP Top 10 – A10 audit procedures provides detailed steps on how to test for this vulnerability (OWASP Foundation, 2011n)

Table 9 – OWASP Top 10 Security Flaws not Included in PCI DSS Requirements

As mentioned previously, PCI DSS v.2.0 emphasized other standards besides OWASP Top 10. Some organizations may have decided that CWE/SANS Top 25 Most Dangerous Software Errors may fit their needs better than OWASP.

The following table lists all the CWE/SANS Top 25 software errors, a reference to OWASP Top 10 if the error is covered in some way by an entry on the OWASP Top 10 list, a description of the software error and a

Christian J. Moldes, christian_moldes@hotmail.com
basic procedure to validate applications are not vulnerable to those software errors.

<table>
<thead>
<tr>
<th>CWE/SANS Top 25 Most Dangerous Software Error</th>
<th>OWASP Top 10 Security Flaw v. 2010</th>
<th>Description</th>
<th>Audit Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-89</td>
<td>A1</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
<td>Refer to the audit procedure for PCI DSS Req. 6.5.1 and OWASP Top 10 Security Flaw A1 above.</td>
</tr>
<tr>
<td>CWE-78</td>
<td>A1</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
<td>Refer to the audit procedure for PCI DSS Req. 6.5.1 and OWASP Top 10 Security Flaw A1 above.</td>
</tr>
<tr>
<td>CWE-120</td>
<td>-</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
<td>Refer to the audit procedure for PCI DSS Req. 6.5.2 above.</td>
</tr>
<tr>
<td>CWE-79</td>
<td>A2</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
<td>Refer to the audit procedure for PCI DSS Req. 6.5.7 and OWASP Top 10 Security Flaw A2 above.</td>
</tr>
<tr>
<td>CWE-306</td>
<td>A4 A8</td>
<td>Missing Authentication for Critical Function</td>
<td>Refer to the audit procedure for PCI DSS Req. 6.5.8 and OWASP Top 10 Security Flaw A4 and A8 above.</td>
</tr>
<tr>
<td>CWE-862</td>
<td>A4</td>
<td>Missing Authorization</td>
<td>This should have been partly verified during the review of PCI DSS Req. 6.5.8, OWASP Top 10 Security Flaw A4 and A8, and the review of access controls as detailed in section 9.2 of this paper. Mitre list several other related CWEs that should be included as part of a review of CWE-862, such as direct requests (forced browsing),</td>
</tr>
</tbody>
</table>

Christian J. Moldes, christian_moldes@hotmail.com
<table>
<thead>
<tr>
<th>CWE/SANS Top 25 Most Dangerous Software Error</th>
<th>OWASP Top 10 Security Flaw v. 2010</th>
<th>Description</th>
<th>Audit Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>Authorization bypass through user-controlled keys, incorrect permission assignment for critical resources and exposed dangerous method or function.</td>
<td>Refer to MITRE CWE-862 for additional information (Mitre, 2011a).</td>
</tr>
<tr>
<td>CWE-798</td>
<td>-</td>
<td>Use of Hard-coded Credentials</td>
<td>Refer to the audit procedure described in section 9.1 above.</td>
</tr>
<tr>
<td>CWE0311</td>
<td>A7</td>
<td>Missing Encryption of Sensitive Data</td>
<td>Refer to the audit procedure for PCI DSS Req. 6.5.3 above.</td>
</tr>
<tr>
<td>CWE-434</td>
<td>-</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
<td>This software error is indirectly addressed by OWASP Top 10 and PCI DSS. This type of security flaw should be detected during application penetration testing, though. References and resources listed for OWASP Top 10 A6 may be helpful to address this security flaw. Refer to A6 – Security Misconfiguration above. Refer to MITRE CWE-434 for additional information (Mitre, 2011b).</td>
</tr>
<tr>
<td>CWE-807</td>
<td>-</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
<td>This software error is indirectly addressed by PCI DSS. This type of security flaw should be detected during application penetration testing, though. Refer to MITRE CWE-807 for additional information (Mitre, 2011c).</td>
</tr>
<tr>
<td>CWE-250</td>
<td></td>
<td>Execution with Unnecessary Privileges</td>
<td>This software error is indirectly addressed by PCI DSS Section 2 requirements. In order to verify that this security flaw does not exist, an inventory of all the roles and accounts used by the application is needed. Each role and account permissions and privileges should be reviewed to determine whether they are necessary or not.</td>
</tr>
</tbody>
</table>

Christian J. Moldes, christian_moldes@hotmail.com
<table>
<thead>
<tr>
<th>CWE/SANS Top 25 Most Dangerous Software Error</th>
<th>OWASP Top 10 Security Flaw v. 2010</th>
<th>Description</th>
<th>Audit Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Microsoft’s IIS.NET site has useful information regarding IIS accounts and their rights on the OS for ISS v.7.0 (Microsoft, 2010).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>References and resources listed for OWASP Top 10 A6 may be helpful to address this security flaw. Refer to A6 – Security Misconfiguration above.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Refer to MITRE CWE-250 for additional information (Mitre, 2011d).</td>
</tr>
<tr>
<td>CWE-352 A5</td>
<td>Cross-Site Request Forgery (CSRF)</td>
<td></td>
<td>Refer to the audit procedure for PCI DSS Req. 6.5.9 above.</td>
</tr>
<tr>
<td>CWE-22 A8</td>
<td>Improper Limitation of a Pathname to a Restricted Directory (‘Path Traversal’)</td>
<td></td>
<td>Refer to the audit procedure for PCI DSS Req. 6.5.8 above.</td>
</tr>
<tr>
<td>CWE-494</td>
<td>Download of Code Without Integrity Check</td>
<td></td>
<td>This software error is not addressed by either the current version of OWASP Top 10 or PCI DSS.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Interview developers to find out whether third party code is being used, how it was obtained, and how integrity check verified.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Refer to MITRE CWE-494 for additional information (Mitre, 2011e).</td>
</tr>
<tr>
<td>CWE-863 A4 A8</td>
<td>Incorrect Authorization</td>
<td></td>
<td>This should have been partly verified during the review of PCI DSS Req. 6.5.8, OWASP Top 10 Security Flaw A4 and A8, and the review of access controls as detailed in section 9.2 of this paper.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mitre list several other related CWEs that should be included as part of a review of CWE-863, such as direct requests (forced browsing), authorization bypass through user-controlled keys, incorrect permission</td>
</tr>
<tr>
<td>CWE/SANS Top 25 Most Dangerous Software Error</td>
<td>OWASP Top 10 Security Flaw v. 2010</td>
<td>Description</td>
<td>Audit Procedure</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------</td>
<td>-------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>CWE-829</td>
<td></td>
<td>Inclusion of Functionality from Untrusted Control Sphere</td>
<td>This software error is not addressed by either the current version of OWASP Top 10 or PCI DSS. Interview developers to find out whether functionality from untrusted control sphere is being used and what security controls are in place to minimize risks from that third party. Refer to MITRE CWE-829 for additional information (Mitre, 2011g).</td>
</tr>
<tr>
<td>CWE-732 A4 A8</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
<td>This requirement is mostly related with the platform on which the application runs. However, it may apply to resources within the application as well. For the application, if there is functionality to assign permissions to critical resources, all the roles and accounts with permissions and privileges should be reviewed. For the platform, refer to the audit procedure for OWASP Top 10 Security Flaw A6 above.</td>
<td></td>
</tr>
<tr>
<td>CWE/SANS Top 25 Most Dangerous Software Error</td>
<td>OWASP Top 10 Security Flaw v. 2010</td>
<td>Description</td>
<td>Audit Procedure</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------</td>
<td>-------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Use of Potentially Dangerous Function</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Incorrect Calculation of Buffer Size</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Improper Restriction of Excessive Authentication Attempts</td>
</tr>
</tbody>
</table>
### How to Audit ASP.NET Applications for PCI DSS Compliance

Christian J. Moldes, christian_moldes@hotmail.com

<table>
<thead>
<tr>
<th>CWE/SANS Top 25 Most Dangerous Software Error</th>
<th>OWASP Top 10 Security Flaw v. 2010</th>
<th>Description</th>
<th>Audit Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CWE-601</strong> URL Redirection to Untrusted Site (‘Open Redirect’)</td>
<td>A10</td>
<td>According to Mitre (2011), automated static analysis tools may not be able to determine whether input influences the beginning of a URL, which is important for reducing false positives. Automated black box tools that supply URLs to every input may be able to spot Location header modifications, but test case coverage is a factor, and custom redirects may not be detected. Using white box techniques may be the best approach to validate compliance for this security flaw. Refer to MITRE CWE-601 for additional information (Mitre, 2011i).</td>
<td></td>
</tr>
<tr>
<td><strong>CWE-134</strong> Uncontrolled Format String</td>
<td>-</td>
<td>Refer to the audit procedure for PCI DSS Req. 6.5.2 above.</td>
<td></td>
</tr>
<tr>
<td><strong>CWE-190</strong> Integer Overflow or Wraparound</td>
<td>-</td>
<td>Refer to the audit procedure for PCI DSS Req. 6.5.2 above.</td>
<td></td>
</tr>
<tr>
<td><strong>CWE-759</strong> Use of a One-Way Hash without a Salt</td>
<td>-</td>
<td>PCI Council recommends the use of Salt; however is not a requirement. For additional information go to PCI SSC FAQ 8718, <em>Are hashed Primary Account Numbers (PAN) considered cardholder data that must be protected in accordance with PCI DSS? (PCI SSC, 2011c)</em></td>
<td></td>
</tr>
</tbody>
</table>

Table 10 – CWE/SANS Top 25 Most Dangerous Software Errors
12. **Conclusion**

It is no surprise that web applications are one of the causes for a significant number of security breaches. Ensuring that an application is secure is a complex and daunting task as it involves many different areas and it requires a good understanding of platform security, secure design, coding, implementation and deployment; hopefully, this paper may facilitate conducting an audit by providing guidance and references to resources that can be used during the audit process.
13. **Acknowledgments**

Special thanks to the following members of the Verizon Business professional services practice:

- Mark Goudie, Managing Principal Investigative Response Team in Australia, for providing technical and content review.
- Charles D. Hylender, Risk Intelligence Team in United States for providing technical and content review.
- Markus Feichtner, Principal Consultant in Germany for suggesting additional content for the paper.
- Rodney Caudle, SANS advisor, for reviewing the document and ensuring that it met SANS paper submission requirements.
14. **References**


How to Audit ASP.NET Applications for PCI DSS Compliance


Microsoft. (2007). "How to use the ASP.NET utility to encrypt credentials and session state connection strings". Retrieved October 19, 2011 from support.microsoft.com Website: http://support.microsoft.com/kb/329290


Christian J. Moldes, christian_moldes@hotmail.com


Christian J. Moldes, christian_moldes@hotmail.com
How to Audit ASP.NET Applications for PCI DSS Compliance


PCI SSC. (2010). "Can a payment application that uses cryptographic keys hard-coded by the vendor be PA-DSS compliant if they cannot be changed by the customer?". Retrieved October 21, 2011 from selfservice.kb.net Website: http://selfservice.kb.net/article.aspx?article=11938&p=81


PCI SSC. (2011c). "Are hashed Primary Account Numbers (PAN) considered cardholder data that must be protected in accordance with PCI DSS?". Retrieved from October 21, 2011 www.pcisecuritystandards.org Website: http://selfservice.kb.net/article.aspx?article=8718&p=81


Christian J. Moldes, christian_moldes@hotmail.com

© 2012 The SANS Institute

Author retains full rights.


Christian J. Moldes, christian_moldes@hotmail.com
Upcoming SANS App Sec Training