City of Chicago

Innovation and Technology

Application Security Policy and Standards

From the Office of the DoIT Chief Information Officer

Table of Contents

Application Security Policy and Standards

Revisions

Author/Editor	Description	Date	Revision	Approved By
Jonathan Villa	Initial Version	02/23/2007	1.0	Tav
Thomas Vari	Removed reference to Clinton. Renamed from Web Application Security Policy to Application Security Policy. Referred to Information Security Standards in the Data Classification Section.	10/31/07	1.1	Tav
Thomas Vari	Changed BIS to DoIT	01/11/08	1.2	Tav
Thomas Vari	Added Brandon	02/15/08	1.3	Tav
Thomas Vari	Renamed Document to from Application Security Policy to Application Security Standards. Updated reference to the Confidentiality and Acceptable Use Agreement. Updated the HTTPS section regarding the “Secure” Attribute.	03/05/08	1.4	Tav
Jonathan Villa	Added additional information regarding the use of hidden fields and filtering data.	07/08/08	1.5	
Jonathan Villa	Added information on use of IP addresses	04/03/09	1.6	Tav
Jonathan Villa	Added captcha section and vulnerability scans	07/30/09	1.7	
Thomas Vari	-Added footer -Added reference to posted Vulnerability Policy. -Added reference to OWASP. -Added Appendix A: PCI supplement - Renamed from ‘standards’ to ‘Policy and standards’ -changed ‘should’ to ‘must’ regarding the use of https	02/15/10	1.8	Tav
Thomas Vari	Added 7.2 per QSA	02/23/2010	1.9	Tav
Thomas Vari	Added 3.1, 9.1 and, 12.6 per QSA	06/23/2010	2.0	Tav and Sec team
Thomas Vari	Added 8.3 and 12.3	06/23/2010	2.1	Tav and Sec team
Thomas Vari	Modified 12.6.1	06/23/2010	2.2	Tav and Sec team
Thomas Vari	Added Appendix B	07/12/2010	2.3	Tav
Jonathan Villa	Added NOTE to App Security Scan	09/15/2010	2.4	JV/TAV
Thomas Vari	Removed Hardy added CWE/SANS Top 25	11/04/2010	2.5	TAV
Thomas Vari	Appendix A was updated with PCI 2.0 specifications. Added reference to HITECH. Modified glossary.	04/13/2011	3.0	Tav and Sec team
Jonathan Villa	Added Insecure cryptographic storage, improper access control, CSRF, and insecure direct object references	4/15/2011	3.2	Tom and Doug

Introduction

The purpose of this appendix is to provide a reference for developers working on City of Chicago Application projects. This appendix will not provide the details required to exploit any particular vulnerability but rather insight into the concepts needed to defend against these attacks. Furthermore, the information contained in this appendix assists the City of Chicago in ensuring the confidentiality, integrity, and availability of its information systems and data. Some examples include:

- Confidentiality – protecting administrative data that has been made available on the web as well as citizen data in transmission and storage;
- Integrity – users of an application's administrative interface continue to see valid and useful data;
- Availability – avoiding database crashes due to web based buffer overflow attacks or storage or performance issues due to automated attacks.

The information in this appendix is designed to protect against both isolated events—as in the case of XSS and SQL Injection—as well as designed to work as part of a larger strategy to defend against Application reconnaissance that is a requirement for an attacker. For example, information gathered from the errors generated by one project may allow an attacker to compromise another application.

This appendix will be updated as new vulnerabilities are discovered and more practical ways of defense can be implemented, for example when vendors release patches that assist in the defense of Applications allowing the web administration team to implement the fix across the enterprise environment.

Purpose

The information technology industry has established several standards in regard to securing private information including patient data, personal information, and payment cards information. The City of Chicago conforms to these requirements and standards as described in the Illinois Personal Information Act, HIPAA privacy and security rules, and the Payment Card Industry Data Security Standard.

Data Classification

All data must be classified as Confidential, Internal, or Public as defined in the City of Chicago's Information Security Policy which is posted on the City's Intranet Site.

General Concerns

The following is a dynamic list of common web based attacks as well as points of interest for attackers and malicious users. This list will be updated as needed.

SQL Injection

SQL Injection is an attack made against a database by means of an application or any other program where dynamic SQL is used by modifying the parameters in order to change SQL statements. This can range from returning data to dropping tables or altering a table structure. Modern enterprise databases and frameworks have taken steps to guard against these attacks however a defense in depth approach is preferred.

Care must be taken to ensure that data transmitted as what often times are assumed to be *nothing more than* Strings or *pure data* are not found to contain metacharacters that may have adverse affects once processed. In order to ensure this, developers should convert any metacharacters contained within a String into their appropriate character entity. Reference: Character entity references.

A popular method used by an attacker to begin a SQL injection attack is to produce an error message by passing invalid data to a form. If a proper error reporting system has not been implemented within the application, an attacker may gain access to table or column names or database version information.

Insecure Cryptographic Storage

As required by the City of Chicago's Information Security Policy, data transmitted, processed, or stored by an application must be encrypted using an encryption algorithm approved by the City of Chicago's policy if the data falls under a classification requiring encryption. Examples of Confidential information that must be protected include Social Security numbers and credit card numbers. Many existing encryption algorithms have become obsolete or have been reclassified as weak algorithms. It is the responsibility of the developer to avoid using insecure cryptographic algorithms while processing, transmitting, or storing data. Examples of insecure cryptographic algorithms for both encryption and hashing data include DES, WEP, MD5 and RC4. This list should not be considered complete therefore referring to the City of Chicago's Cryptographic Policy and Standards document is required.

Cross-Site Request Forgery

Cross-Site Request Forgery (CSRF) allows an attacker who has gained control over a website to successfully execute HTTP requests to another website where the user has already authenticated. Once the initial Phishing or trickery has been successful, CSRF is successful because it utilizes HTTP GET and POST. To protect against CSRF, an application should implement methods that it can rely on to guarantee that every request has been initiated from within a valid session and not that of a simple HTTP GET or POST using the an authenticated session ID. A recommended approach to protecting against CSRF is the use of a unique-per-user token submitted with every form. Furthermore, critical business logic should be initiated on a GET request versus an HTTP POST.

Improper Access Control

City of Chicago web applications that have been designed to separate tasks or privileges between guest users, users with rights to manage data, and administrators should take due care to properly implement a secure access control design, e.g. Role Based Access Control (RBAC). Permissions should not rely on values not under the constant control of the application's business logic. Storing access control values in cookies, hidden fields, or a GET parameter (query string) is considered insecure. Storing or relying on values used by the business logic to determine access is also considered insecure.

Insecure Direct Object References

Applications are often developed to use identifying data as primary keys, for example account numbers or order numbers. These values are then used within the application to help a user navigate throughout the application, e.g. clicking on an order id to see details of the order. While safeguards to protect against parameter tampering via GET parameters may be implemented, an attacker may still be able to POST an HTTP request with manipulated data fields causing the application to response with data from a different record. Developers should avoid exposing direct object references within their applications.

Another example of insecure direct object reference is the use of URL parameters used to access generated files on a file system. For example, generating a file or graphic and referring to it in the following manner, file=12345.txt or graph=6789.png.

Administrators should secure web and applications servers against insecure direct object references by disabling indexing of file system directories. Not doing so allows for directory traversal which allows an attacker to browse the servers file system through a web browser.

Cross Site Scripting

Cross Site Scripting (XSS) is an attack against the integrity of an application as the main target is the end user. If an application is vulnerable and an end user is victimized, the integrity or reliability of the application—and perhaps applications in the same environment—comes into question. XSS is typically used to steal session information and then contact the site on behalf of the victim. This can have adverse effects when the end users data contains personal or financial information.

An application becomes vulnerable when it lacks proper output filtering. Properly handling the data during input will help safeguard against this attack.

Example to test for XSS:
1. User enters in the characters "><script>alert('test')</script> into the name field on a form and submits the page.
2. Some other validation fails and the user is presented with the same form & data.
3. The first two characters, "> will be interpreted by the web browser and will close out the name field's value and generate a JavaScript alert.
   <input type="text" name="" value=""><script>alert('test')</script>">

If the data was filtered, the value of the field would be value="&quote;&gt;alert &#40;&quote;test &quote;&#41;&lt;&#47;script&gt;". This would be rendered after the web browser has completed interpreting the markup language. Otherwise, when not filtered, the first two characters will be interpreted by the web browser as markup language.

Certain development frameworks provide data filtering capabilities that provide data filtering within the framework's library. Developers should ensure that all form fields and user input mediums are being filtered properly and not rely solely on the framework to provide the filtering functionality.

Unhandled Exceptions

Every effort must be made to avoid displaying any information relating to source code, database connections, City of Chicago package naming structures (org.cityofchicago.dept.webapp), etc. This aligns with our defense against Application reconnaissance. Errors printed to the browser may contain database or table names, a hint of database credentials, or other information. Every error should be caught and a DoIT approved system error page displayed.

Errors may also reveal versioning information regarding libraries, server versions, etc. and may be recorded for later user when that resource has been to be vulnerable.

Comments

Pages that are displayed to the user should contain a very minimal set of comments. Comments can reveal usernames or emails of developers as well as old functionality that is no longer needed and only provides the attacker with more tools for reconnaissance. Comments should be kept to indicating very basic HTML structural information and should not inform about loops, conditionals, or included files.

Validation

In a security best case scenario, validation of user input would be performed on the server. There is a debate about performance in this area where administrators prefer to avoid making repeating calls to the application server to validate user input. Developers have opted to introduce client-side validation using JavaScript to avoid these repeated calls.

JavaScript—while great for enhancing the user experience—has now been called to help the server rather than user. The security aspect of validating user data has now been pushed onto the client where the application and its security layer have lost control. A malicious user can easily utilize a proxy server to bypass any client-side validation. Often times an application is relying on the client-side validation process and when not performed, may cause an unhandled exception to occur. This may provide the malicious user with information to be used later in an attack.

Developers should **always** implement a defense in depth approach when validating user input, i.e. **always** validate the user input on the server as well. If possible, avoid using client-side validation which is quickly becoming a security standard.

Forms

Form fields should have maximum length values corresponding to the limit defined in the database. Setting the appropriate maximum length values will assist in defending against web based buffer overflow attacks as well as displaying database or application information in the event of an unhandled exception, e.g. "*value too large for column".*

Form names should be unique, i.e. they should not contain the exact name of the database column. For example, if the database column used to store a person's first name is firstName, then the form field for first name should not be firstName but rather something unique such as deptFirstName.

While developers may not always have control over the names of the form fields, this approach should be used whenever possible.

Invalid data should always be treated as invalid data. Trying to convert the data into valid data may cause unexpected results. Developers should always ensure that the data being supplied by the client/end user is valid.

When validating data, a whitelist approach should be implemented by validating for expected values rather than attempting to deny certain values.

Captchas

In an effort to deter the use of automated tools and scripts used to perform validation of email accounts within City web applications, developers should implement the use of captchas. A scenario in where an application would be vulnerable to an email harvester would be found on a password reset page. When a user submits their email address in an effort to get a new password, it is assumed that the application will present an informational message indicating that the email will be sent with a new password. This results in a positive test for an email harvester in that the email address used has just been confirmed as valid within the system.

While an attacker may conduct this test manually, the purpose of the captcha is deter the attacker and prevent them from user automated scripts and tools to validate email addresses.

Information on captcha implementations at the City can be requested by contacting the Department of Innovation and Technology.

Hidden Fields

Hidden form fields are normally used to store server generated data. As is the case with the client-side validation, this puts the security—or integrity of the data—on the web browser thus losing the control on the server. A malicious user can easily use a proxy server to manipulate any POST data that is in transmission between the web browser and the server.

Developers should exercise caution when using hidden form fields and ensure that the logic of the application does not rely on the data stored in hidden form fields for functionality that drives the flow of the application. The following are examples of what not to do:

- storing the price of an item in a commerce application
- storing the user id or role of an authenticated user and using that data to determine access control for the application
- state of an authenticated session, e.g. loggedIn=true

Username Enumeration

Keeping in line with maintaining the integrity of the applications and confidentiality of data, developers should make additional efforts to avoid disclosing any information relating to the end user. One of the ways that this can be implemented is in how application generated error messages and password recovery methods are implemented and displayed. For example, messages displayed on failed login attempts should not inform the end user that the password was incorrect only. Messages indicating the password was incorrect inform an attacker that the username was correct. An attacker can now begin a dictionary attack once the username is known.

Password recovery processes should also require additional information rather than just a username prior to presenting the user with the secret question. This also allows an attacker to assume that the username was correct.

Information Disclosure

From time to time developers may need to interact with vendors or the public via a forum/bulletin to troubleshoot an issue. City of Chicago infrastructure information such as server names, IP addresses, or detailed layouts should not be shared with these outside sources unless approved by the DoIT administration team. Furthermore, it should be a standard practice to use hostnames and not IP address when distributing test URLS

Developers will often post configuration files such as web.xml, properties files, or similar on forums for review by others. The process of *scrubbing* should be applied in these cases. Scrubbing involves removing information that pertains to an organization from the file while still maintaining enough configuration information to properly diagnose the problem. For example, IP addresses, server names, database credentials or versioning information, and the like should never be posted as is. IP addresses should be replaced by 12.34.56.78 or xx.x.x.x or similar. Server names and database credentials should be replaced with generic information such as username=username, password=password, serverOne, serverTwo, etc.

Information—such as logs files and configuration files—supplied to vendors such as BEA, FileNet, or others should be carefully reviewed in order to determine if any personal customer information is contained within the file. Credentials for QA and Production systems should always be removed from any information submitted to vendors.

If there is any doubt over the classification of data, please direct any concerns to one of the DoIT contacts listed in the Contacts section of this appendix.

Page Caching

Developers must ensure that pages that process or display private or sensitive information contain the appropriate no-cache settings to avoid saving any private or sensitive information on the customers computer.

Examples include purging form field data so that private and sensitive data is not available when a browsers back button is used or when temporary internet files are viewed on a computer.

Defaults

Default configuration parameters and values should not be used for any application that transmits or processes private or sensitive data.

HTTPS

Any page that handles Confidential Information including pages that require a user to authenticate MUST be processed under a Secure Sockets Layer. Many people use the same password for various applications and the compromising of one application may facilitate the compromising of another more important application. Furthermore, applications that handle Confidential information must enable their cookie's "Secure" attribute so that cookies used by the application are not sent in clear text. Setting this attribute prevents the browser from sending the cookie to the server over an unencrypted link.

End users will be on both public and private networks and may be susceptible to a packet sniffing program controlled by an attacker. A packet sniffing program captures network data in transmission on the home network to its final destination. When this data is passed in plain text, a packet sniffer will a result similar to the following example.

```
0000   00 13 02 d1 eb 89 00 12 17 04 f2 71 08 00 45 00  ...........q..E.
0010   00 4d 58 5b 40 00 36 06 42 d2 42 b4 a3 bb c0 a8  .MX[@.6.B.B.....
0020   02 66 00 15 08 dc 9f b5 b1 74 7c 42 53 6d 50 18  .f.......t|BSmP.
0030   16 d0 b0 76 00 00 33 33 31 20 55 73 65 72 20 77  ...v..331 User w
0040   68 73 66 20 4f 4b 2e 20 50 61 73 73 77 6f 72 64  hsf OK. Password
0050   20 72 65 71 75 69 72 65 64 0d 0a                 required..
```

The above example is a capture of an FTP connection with the username of whsf (the password has been withheld).

Invalid Filenames

During development, it is common to start working on a new version of a file and renaming the original file with an extension of .bak, .txt, .old, etc. This should be avoided at all times. Attackers often employ tools capable of file type probes. A file type probe crawls a website and then attempt to render the recorded pages with extensions of .bak, .txt, .old, etc. If successful, a JSP page that contained server side comments and other information has now been rendered in plain text over a web browser.

Compliance

Illinois Personal Information Protection Act

Under the Illinois Personal Information Protection Act the City of Chicago is defined as a “data collector” and conforms fully to this Act.

HIPAA and HITECH Acts

Any project that stores, processes, or transmits patient data must meet the standards as described in the Health Insurance Portability and Accountability Act (HIPAA) and the Health Information Technology for Economic and Clinical Health (HITECH) Act. Data owners and data custodians must ensure that secure procedures are used during the creation, transmission, storage, processing, and disposing of patient data.

PCI

Projects handling payment card information must meet the standards set forth by the Payment Card Industry Data Security Standard. The PCI DSS has established twelve requirements for transmission, processing, and storage of payment card information. Developers must meet and ensure that the applications being developed adhere to the following requirements when working with payment card information:

- Requirement 2: Do not use vendor-supplied defaults for system passwords and other security parameters.
- Requirement 3: Protect stored cardholder data by masking all but the last four credit card numbers in any display or printout
- Requirement 4: Encrypt transmission of cardholder data across open, public networks.
- Requirement 6: Develop and maintain secure systems and applications *See Appendix A: PCI Supplement for detailed requirements.*

The PCI DSS can be downloaded as a read and copy version only for study purposes by visiting the following link:
https://www.pcisecuritystandards.org/

External Vulnerability Assessment (Ethical Hacks)

The City of Chicago conducts various external and internal vulnerability assessment audits in order to identify vulnerabilities that may compromise the confidentiality, integrity, or availability of data. External audits in a typical year include, at a minimum, an audit of financial systems, an audit of PCI systems, at least one ethical hack test, and monthly vulnerability scans. Internal vulnerability audits in a typical year include, at a minimum, monthly network/server vulnerability scans, internal HIPAA audits, responses to IDS/IPS alerts and event-driven application vulnerability scans.

Refer to the Vulnerability and Remediation Policy and Procedures in the Security Intranet Portal for the latest policy regarding vulnerability assessments and remediation.

Most of these audits produce a vulnerability risk rating. It is expected that remediation deployment for vulnerabilities with a risk rating above "High" must be addressed as soon as possible but no more than 30 days from the date the vulnerability was found. Those with a risk rating of "Medium" must be addressed within 90 days. Since identification of a vulnerability cannot always be anticipated, your project manager must be informed if remediation of vulnerabilities will impact your project. By following coding practices in the Application Security Standards Manual and by pro-actively obtaining vulnerability scan reports as part of your SDLC, you will dramatically reduce delays related to the required remediation of vulnerabilities identified with your application after it is migrated to production.

Application Security Scan

As part of the City of Chicago Development Standards, developers must request an application security scan for applications being deployed to the production environment prior to deployment. This should coincide with other performance testing such as load tests. The administration team will request the results of the security scan prior to deploying the new or updated application to production. Applications must not have any vulnerabilities or possible vulnerabilities with a severity level greater than medium. The severity levels are: info, low, medium, high, and critical. A sample report is available upon request from the administration team.

NOTE: Certain application frameworks provide “catch all” responses for system or user generated error. While this is a not only a security but a development best practice, it does provide a conflict with the automated application security scan. If such configuration exists, it should be disabled during the scanning process in QA. Developers should make note to enable it prior to going to production.

.NET example: <customErrors mode="Off" />(file: web.config)

Confidentiality and Acceptable Use Agreement

Each developer and/or company is responsible for reading and signing the City of Chicago Confidentiality and Acceptable Use Agreement. Developers are required to abide by this Statement throughout their relationship with the City of Chicago.

Developer Responsibilities

Security Awareness

Apart from ensuring that City of Chicago projects developed by City of Chicago developers are meeting security guidelines and standards, developers should be monitoring security patches to frameworks, libraries, commercial applications, and any other resource utilized by the City of Chicago. Security awareness should be practiced by implementing a defense in depth approach and not relying on the underlying framework as the only security layer.

Examples:

- Microsoft .NET vulnerability
- FileNet Workplace – XSS found by the City of Chicago
- BEA Weblogic Multiple Vulnerabilities

Developers must become familiar with the Open Web Application Security Project (OWASP) at least to the extent of understanding that many of our standards are based upon the best practices identified by the project. More information is available at www.owasp.org. In addition to highlighting the 10 ten application security risks, you will find resources such as the OWASP Developer Guide, Testing Guide, and Code Review Guide.

Developers should also become familiar with the CWE/SANS Top 25 Software Errors. Along with the list, the site provides discussions, technical details, code examples, detection methods and references. More information is available at http://www.sans.org/top25-software-errors/.

Contacts

- Douglas Hurdelbrink – douglas.hurdelbrink@cityofchicago.org
- Thomas Vari – tvari@cityofchicago.org
- Jonathan Villa – jvilla@cityofchicago.org

Glossary

- Developer – any employee or consultant serving as a development, design, administration, or management resource for any City of Chicago information technology project.
- Vendor – Any commercial solution provider such as BEA, FileNet, Microsoft, Oracle, Red Hat, etc.
- Data custodian – any resource responsible for the administration or development of storage devices that persist data such as a database, network storage, or local disk as well as transmit or processes data.
- Developer – any employee or consultant serving as a development, design, administration, or management resource for any City of Chicago information technology project.
- Vendor – Any commercial solution provider such as BEA, FileNet, Microsoft, Oracle, Red Hat, etc.
- Data custodian – any resource responsible for the administration or development of storage devices that persist data such as a database, network storage, or local disk as well as transmit or processes data.
- Defaults – predetermined entries an application will use to populate fields with no source input data or replace null values.

Appendix A: PCI Supplement

The following procedures (as identified in the PCI DSS Requirements and Security Assessment Procedures v 2.0) must be adhered to:

3.1 Cardholder data storage is not allowed unless the storage solution was reviewed by a QSA or approval has been obtained from the DoIT Manager of Custom Development and the DoIT Security Architect. In the case where storage of cardholder data is approved:

- **Cardholder data must not be retained if it is not needed for legal, regulatory, or business requirements. In some isolated cases, cardholder data may be needed to be held for 2 days for resolving conflicts.**
- **When cardholder data is no longer needed, it must be disposed of using proper techniques to make sure that the cardholder data cannot be made available. (Also see 9.10)**
- **The application support team must verify that cardholder data does not exceed business retention requirements no less than quarterly.**

3.4 The storage of PAN data is not allowed unless the storage solution was reviewed by a QSA or approval has been obtained from the DoIT Manager of Custom Development and the DoIT Security Architect. In the case where storage of the PAN is approved, the PAN must be rendered unreadable using one of the following methods:

- **One-way hashes based on strong cryptography**
- **Truncation**
- **Index tokens and pads, with pads being securely stored**
- **Strong cryptography, with associated key-management processes and procedures**

6.3 Develop software applications in accordance with PCI DSS and based on industry best practices, and incorporate information security throughout the software development lifecycle. The processes must include the following:

6.3.1 Custom application accounts, user IDs and/or passwords are removed before system goes into production or is released to customers.

6.3.2a All custom application code changes must be reviewed (using either manual or automated processes) as defined in PCI DSS 2.0.

- Code changes are reviewed by individuals other than the originating code author, and by individuals who are knowledgeable in code review techniques and secure coding practices.

-Code reviews ensure code is developed according to secure coding guidelines (See PCI DSS requirement 6.5).

-Appropriate corrections are implemented prior to release.

-Code review results are reviewed and approved by the Manager of Development (or his designate) prior to release.

Appendix A: PCI Supplement (cont'd)

6.4 Follow change control processes and procedures for all changes to system components. The processes must include the following:

6.4.1 Non-production environments must be separate from the production environment and access controls must be in place to enforce the separation.

6.4.2 There must be a separation of duties between personnel assigned to the non-production environment and those assigned to the production. environments unless an alternative is approved by a QSA.

6.4.3 Production PANs must not be used in non-production environments. If an explicit exception is made by the Manager of Custom Development, production PANs must be removed prior to migration to production.

6.4.4 Furthermore, test data and accounts must be removed from a production environment before the production system becomes active.

6.4.5 Change control procedures for the implementation of security patches and software modifications must contain the following:

- 6.4.5.1 Documentation of impact.
- 6.4.5.2 Documented change approval by authorized parties.
- 6.4.5.3 Functionality testing to verify that change does not adversely impact the security of the system.
- 6.4.5.4 Back-out procedures.

6.5 Develop all applications based on secure coding guidelines such as the Open Web Application Security Standard, the SANS CWE Top 25 or CERT Secure Coding. Safeguards must be in place to explicitly address, at a minimum, the following vulnerabilities:

6.5.1 Injection flaws, particularly SQL injection
(E.g. validate input to verify user data cannot modify meaning of commands and queries, utilize parameterized queries, etc.)

6.5.2 Buffer overflow
(E.g. Validate buffer boundaries and truncate input strings)

6.5.3 Insecure cryptographic storage (E.g. Prevent cryptographic flaws)

6.5.4 Insecure communications (E.g. Properly encrypt all authenticated and sensitive communications)

6.5.5 Improper error handling
(E.g. Do not leak information via error messages or other means.)

6.5.6 All “High” vulnerabilities as identified in PCI DSS Requirement 6.2.

6.5.7 Cross-site scripting (XSS)
(E.g. Validate all parameters before inclusion, utilize context-sensitive escaping, etc.)

6.5.8 Improper Access Control, such as insecure direct object references, failure to restrict URL access, and directory traversal
(E.g. Properly authenticate users and sanitize input, Do not expose internal object references to users.)

6.5.9 Cross-site request forgery (CSRF)
(E.g. Do not reply on authorization credentials and tokens automatically submitted by browsers.)

7.2 All system components within PCI scope must have access control systems in place.

Appendix A: PCI Supplement (cont'd)

8.1 All users must have unique IDs for access to system components that are within PCI scope. Furthermore, the accounts their associated passwords must not be shared.

8.3 All remote access by administrators must use two-factor authentication for remote access. (Also see 12.3.x)

9.10 Media containing cardholder data must be destroyed when it is no longer needed for business or legal reasons as follows:

- 9.10.1 Hard-copy materials must be cross-cut shredded, incinerated or pulped such that there is reasonable assurance that the hard-copy materials cannot be reconstructed. Storage containers used for information to be destroyed must have a lock to prevent access to its contents.
- 9.10.2 Cardholder data on electronic media must be rendered unrecoverable via a secure wipe program in accordance with industry-accepted standards for secure deletion such as NIST or otherwise physically destroying the media (for example, degaussing.)

12.3 The project manager for all PCI related applications must assure that the following arrangements are made with respect to their PCI project's infrastructure:

- 12.3.4 All PCI components must include proper asset management including the proper labeling of devices.
- 12.3.6 All PCI components must be installed in approved PCI network security zones.
- 12.3.8 All remote-access technologies must use software approved by DoIT and must have an automatic disconnect after 30 minutes of inactivity.
- 12.3.9 Remote-access technologies for vendors that are not part of the City Workforce (i.e. employees/consultants) must be activated only when needed by vendors, with deactivation after use.

12.6 All employees and consultants who work with PCI data must participate in the City's Security Awareness program as follows:

- 12.6.1 Attending security awareness training upon hire and at least annually is required. Additional security training awareness methods are also available. They include the City's Intranet Security Portal (where policies, procedures, standards, and training materials are posted), a Security Reminder Page, articles in the DoIT newsletter and event driven emails regarding security.
- 12.6.2 Acknowledging (in writing or electronically) at least annually that they have read and understand the City's Information Security Policy.

Appendix B: HIPAA Supplement

1.0 HIPAA data is classified as Confidential therefore all City policies related to the protection of Confidential data reflect the minimum requirements for protecting HIPAA data. In many cases, the policies and procedures documented to protect PCI data reflect best practices that should be applied to protecting any Confidential data.

2.0 All project managers for HIPAA related projects must be familiar with HIPAA regulations and the City's HIPAA related policies. (These polices are available on the City's Security Intranet Site). A CDPH HIPAA checklist (which may be obtained from CDPH) has been created to document many of the implementation requirements. All project managers for HIPAA related projects must complete the CDPH HIPAA checklist. An approval of the document (or a granted extension) from CDPH HIPAA Officers is required before moving an application to production.

3.0 All project managers for HIPAA related projects must assure that the infrastructure components for HIPAA related projects are placed in DoIT approved HIPAA security zones that are protected with approved firewalls and monitoring techniques. The project manager must assure that each tier (Web, Application, DB, and Network) has a person assigned to address vulnerabilities and patching requirements according to the City's policy.

4.0 All rules for accessing HIPAA data must use access control procedures which are auditable. Furthermore, remote access must be consistent with methods approved for access to Confidential data.

5.0 All employees and consultants who work with PCI Data must participate in the City's Security Awareness program as follows:

3.1 Attending HIPAA security awareness training upon hire (for a HIPAA project) and at least annually are required. Additional security training awareness methods are also available. They include the City's Intranet Security Portal (where policies, procedures, standards, and training materials are posted), a Security Reminder Page, articles in the DoIT newsletter and event driven emails regarding security.

3.2 Acknowledging (in writing or electronically) at least annually that they have read and understand the City's Information Security Policy.