

**HCTM Application Security Standards**

***Version 3.01***

1 Background 4

2 Document Structure 4

3 Security Standards 5

3.1 Authentication 5

3.2 Session Management 7

3.3 Access Control 8

3.4 Input Handling 9

3.5 Error Handling and Logging 11

3.6 Cryptography and Data Protection 12

3.7 Secure Communication 15

3.8 HTTP Security 16

3.9 Operational Security 17

4 Glossary 18

Document Revision History

| Version | Modification | Date | Author |
| --- | --- | --- | --- |
| 1 | Version 3.01 (Web application security standards) | October 1, 2014 | Koen Buyens  Chandu Ketkar  Carl Schwarcz |
|  |  |  |  |

# Background

This document enumerates security standards for software applications that are either already developed or currently being developed by the HCTM group at Bosch Healthcare. In the first release of this document, the focus is on developing security standards for web applications. In the next version, these security standards will be enhanced to include standards pertaining to embedded systems security.

These security standards are sourced from various community sources (such as OWASP, WASC), Cigital’s expertise, and informed by Bosch’s security context – including the types of applications issues Cigital has repeatedly encountered while assessing HCTM applications.

Whereas each HCTM application (or release) may or may not have its own set of security requirements, the security standards in this document are mandatory and will act as the default security requirements for all HCTM applications and their releases. All HCTM applications are expected to comply with these security standards and exceptions should be filed when it is not feasible for technical or business reasons.

# Document Structure

The security standards are categorized by security concerns such as authentication, authorization, and session management.

Most terms used in this document are well-known to a security practitioner, however, a comprehensive glossary is provided at the end to help with a common understanding of the security related terms used in this document.

# Security Standards

This section lists the security standards. Each security standard contains the following descriptive elements:

* **Description**: lists the security standard itself.
* **External references**: provides references to external sources that describe the security standard in more detail.

## Authentication

Authentication is the process of determining whether someone or something is, in fact, who or what it is claiming to be. Authentication is a security control and allows an application to identify a user who is trying to access the application’s functionality. Authentication should be required before accessing important system functionality, data, and resources.

Currently, the most commonly used method to authenticate a user to a system consists of using a shared secret, such as a password. A user presents a user ID or a username and then provides a secret value, the password that only the user (and the system to which the user is authenticating) knows. The major threats to knowledge-based authentication are that passwords can be guessed, sniffed from the wire, captured and re-used, or stolen from a compromised machine (e.g. an application database that stores the password). Therefore, the authentication scheme not only relies on the protection of the systems and network handling the secrets, but also on the strength of the secrets itself: how difficult the password is to guess. Strong secrets are required to protect user accounts from various types of brute force attacks – where an attacker can simply run through all possible passwords to find a correct one.

The following authentication security standards protect against common threats:

1. Authentication controls shall be enforced on the server side.  
   **More Information**: *https://www.owasp.org/index.php/Authentication\_Cheat\_Sheet*
2. All resources shall require authentication before access except those specifically intended to be public.  
   **More Information**: *https://www.owasp.org/index.php/Authentication\_Cheat\_Sheet*
3. Applications shall respond with a generic error message when the user enters a wrong username/password combination. The error message shall not provide an indication of the status of an account. An example is: ‘*Login Failed: Invalid username or password.*’  
   **More Information**: *https://www.owasp.org/index.php/Authentication\_Cheat\_Sheet*
4. Account lockout mechanisms shall be implemented. These should lock out an account if more than a number of unsuccessful login attempts, typically three, have been made.  
   **More Information**: *https://www.owasp.org/index.php/Authentication\_Cheat\_Sheet*
5. Account unlock mechanisms shall be employed. These should unlock an account after it has been locked. The exact mechanism depends on the business needs of the application, but commonly used methods include the following:
   1. The accounts can unlock automatically after a predefined period of time – e.g. 20 minutes.
   2. A call must be made before an administrative employee unlocks the account.

**More Information**: *https://www.owasp.org/index.php/Authentication\_Cheat\_Sheet*

1. Identity information handled by the application(s) (including passwords, credentials, and session ids) shall only traverse encrypted links, i.e. TLS v1.1or TLS v1.2 with
   1. a cipher length of at least 128 bit,
   2. disabled TLS compression,
   3. disabled client-side renegotiation, and
   4. a certificate per host.

**More Information:** *https://www.owasp.org/index.php/Transport\_Layer\_Protection\_Cheat\_Sheet*

1. Account passwords shall be stored in protected form at the server-side:   
   [protected form] = [salt] + hash([hash func], [salt] + [password]);   
   The protected form consists of a salt and a salted hash of the password. A salt is a cryptographically secure random value of at least 32 bits. The password (or password phrase) should be appended to the salt. This combination should be used as input to a hash function. The salt shall be unique to each password.  
   **More Information:**[*https://www.owasp.org/index.php/Password\_Storage\_Cheat\_Sheet*](https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet)
2. Account passwords shall never be stored at the client-side in insecure storage such as cookies. If an application requires storing credentials client-side then the application shall use secure storage offered by the operating system.  
   **More Information:**[*https://www.owasp.org/index.php/Password\_Storage\_Cheat\_Sheet*](https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet)
3. Shared secrets (such as passwords) used in direct authentication shall have sufficient entropy (at least 80 bits).  
   **More Information:**[*http://en.wikipedia.org/wiki/Kolmogorov\_complexity*](http://en.wikipedia.org/wiki/Kolmogorov_complexity)
4. The system shall enable users to use complex passwords or passphrases: These passwords shall be between 8 and 128 or more characters. The passwords shall contain characters from three of the following five categories:
   1. Uppercase letters (A-Z),
   2. Lowercase letters (a-z),
   3. Numbers (0-9),
   4. Non-alphanumeric characters ~!@#$%^&\*\_-+=’`<>,.?/|\{}[]():;
   5. Any Unicode character that is categorized as an alphabetic character but is not uppercase or lowercase. This includes Unicode characters from Asian languages.

**More Information**: *http://technet.microsoft.com/en-us/library/cc786468(v=ws.10).aspx*

1. The system shall protect against the use of commonly chosen passwords by verifying the chosen passwords against (password) dictionary lists.   
   **More Information**: *https://owasp.org/index.php/Password\_length\_%26\_complexity*
2. All authentication decisions shall be logged. The logs shall contain the time of the authentication, the time of logging, user ID, whether authentication succeeded, and a description. **More Information**: *https://www.owasp.org/index.php/Logging\_Cheat\_Sheet*
3. All authentication credentials for accessing services external to the application shall be encrypted and stored in a protected location (not in source code), typically configuration files that contain the encrypted shared secret. Access to the configuration files and encryption keys shall be controlled by operating system access. Keys may be stored by a (dedicated) hardware security module.  
   **More Information**: *https://www.owasp.org/index.php/Authentication\_Cheat\_Sheet* and *https://www.owasp.org/index.php/Key\_Management\_Cheat\_Sheet  
   http://en.wikipedia.org/wiki/Hardware\_security\_module*
4. The *‘forgot password*’ functionality shall send a link including a time-limited activation token rather than the password itself. For applications with a critical risk (as defined in the risk rating document), additional authentication using a second factor (e.g. something the user has such as SMS token or hardware token) shall be required before the link is sent.  
   **More Information:** *https://fishnetsecurity.com/6labs/resource-library/white-paper/best-practices-secure-forgot-password-feature*
5. Users shall be required to re-authenticate to the web application before executing any sensitive transaction. Sensitive transactions are transactions that either directly manipulate sensitive data or accesses sensitive business functionality. Examples of sensitive transactions are password changes, viewing patient data, permission changes, or changes to user roles (see glossary for a more elaborate definition).  
   **More Information**: [*https://www.owasp.org/index.php/Authentication\_Cheat\_Sheet*](https://www.owasp.org/index.php/Authentication_Cheat_Sheet)
6. If the application supports account deletion, then the application shall prohibit users from creating accounts with user IDs of deleted accounts.  
   **More Information**: [*https://www.owasp.org/index.php/Authentication\_Cheat\_Sheet*](https://www.owasp.org/index.php/Authentication_Cheat_Sheet)

## Session Management

Web applications use the HTTP protocol for transporting requests and responses. The HTTP protocol is stateless, which means that each HTTP request and response carries with it no context regarding previous requests. Consequently, web applications require the ability on part of the web application to retain state (such as user identity) across user interactions. A session is a sequence of request and response transactions.

There are many methods of managing state within a session; some are more secure than the others. One could store state in a hidden HTML field (e.g. VIEWSTATE in .NET applications) or one could store the state at the server and pass its handle (commonly referred to as the Web Session ID or simply Session ID) in request/responses. The latter is the best practice for most web applications. In order to keep the state and track the users progress within the web application, applications provide users with a session ID that is assigned at session creation time (i.e. during authentication), and is shared and exchanged between the user and the web application for the duration of the session (it is sent on every request). The ‘*state*’ is stored server-side and the session ID is used to access the state.

If an attacker can get access to a session ID, an attacker can bypass authentication controls and still spoof the user during the session.

The major threats to session management are that session IDs can be guessed, sniffed from the wire, captured and re-used, or stolen from a compromised machine. Therefore, session management not only relies on the protection of the systems and network handling the session IDs, but also on the strength of these IDs; i.e. the difficulty of guessing the session IDs.

The session ID must have various properties in order to be secure, as described by the following security standards.

1. The session IDs shall be at least 128 bits and random (64 bits of entropy). The session IDs shall be created through the usage of cryptographically secure random number generation functions. These IDs shall be regenerated during login and re-authentication.  
   **More Information**: *https://www.owasp.org/index.php/Insufficient\_Session-ID\_Length* and *https://www.owasp.org/index.php/Session\_Management\_Cheat\_Sheet*
2. Sessions (including session data) shall be invalidated when a user logs out, or after a specified period of inactivity, typically after 15 to 60 minutes, depending on the application business context.  
   The session timeout management and expiration shall be enforced on the server-side.  
   After invalidating the session, the user must be forced to (re)authenticate to the web application and establish a new session.  
   **More Information**: *https://www.owasp.org/index.php/Session\_Management\_Cheat\_Sheet*
3. Session IDs shall be renewed or regenerated by the web application after any privilege level change within the associated user session. An example is the authentication process, where the privilege level of the user changes from the unauthenticated (or anonymous) state to the authenticated state. Other scenarios that should be considered, are password changes, permission changes, or switching from a regular user role to an administrator role within the web application. For all these web application critical pages, previous session IDs have to be ignored, a new session ID must be assigned to every new request received for the critical resource, and the old or previous session ID must be destroyed.  
   **More Information**: *https://www.owasp.org/index.php/Session\_Management\_Cheat\_Sheet*
4. Identity information (including session IDs and credentials) shall never be disclosed; particularly in URLs, HTML-bodies, error messages, or logs.  
   **More Information**: *https://www.owasp.org/index.php/Session\_Management\_Cheat\_Sheet*
5. Cookies containing authenticated session tokens shall be only sent to a restricted number of paths on the origin server. This can be achieved by setting the *Domain* and the *Path* attributes. The ‘*Domain*’ attribute should not be set (restricting the cookie just to the origin server) and the ‘*Path*’ attribute should be set as restrictive as possible to the web application path that makes use of the session ID. This ensures that webpages outside that path are not able to use the cookie.  
   **More Information**: *https://www.owasp.org/index.php/Session\_Management\_Cheat\_Sheet*
6. Cookies containing authenticated session tokens shall only be communicated on a secure channel, such as SSL. This can be achieved by setting the ‘*Secure’* property.  
   **More Information**: *https://www.owasp.org/index.php/Session\_Management\_Cheat\_Sheet*
7. Cookies containing authenticated session tokens shall not be accessible by JavaScript. This can be achieved by setting the ‘*HttpOnly*’ property.  
   **More Information**: *https://www.owasp.org/index.php/Session\_Management\_Cheat\_Sheet*
8. Cookies containing authenticated session tokens shall not be persisted by setting the properties ‘*Max-Age’* and ‘*Expire’*.  
   **More Information**: *https://www.owasp.org/index.php/Session\_Management\_Cheat\_Sheet*
9. Session IDs shall only be a cryptographically random ID (or a random token) and never include sensitive information or Personal Identifiable Information. The meaning and business or application logic associated to the session ID must be stored on the server-side, and specifically, in session objects or in a session management database or repository. The stored information can include the client IP address, User-Agent, e-mail, username, user ID, role, privilege level, access rights, language preferences, account ID, current state, last login, session timeouts, and other internal session details.  
   **More Information**: *https://www.owasp.org/index.php/Session\_Management\_Cheat\_Sheet*

## Access Control

Authorization is the process of verifying that an authenticated user has sufficient privileges to access resources controlled by the system. It should be noted that authorization is not equivalent to authentication - as these terms and their definitions are frequently confused.

In any access-control model, the entities that request access to resources are called subjects, while the entities representing resources to which access may need to be controlled are called objects.

Acommon access control model is **Role Based Access Control.** This model assigns users to roles based on their organizational functions and determines authorization based on those roles.[[1]](#footnote-1)

Common issues for access control include:

* Applications do not consistently perform access control checks on the server-side for all functionality.
* Applications accidentally expose internal resources or directories.

The following access control security standards protect against common vulnerabilities and flaws:

1. The access control rules implied by the presentation layer (client-side) shall be enforced on the server-side.  
   **More Information**: *https://www.owasp.org/index.php/Access\_Control\_Cheat\_Sheet*
2. Access control shall be enforced on the server-side for all resources (functions, services, data files, and other URLs). Examples include data describing which role has access to what resources (including roles); and data attributes of resources; i.e. metadata such as read-only or its creation time.  
   **More Information**: *https://www.owasp.org/index.php/Access\_Control\_Cheat\_Sheet*
3. Directory browsing shall be disabled unless deliberately required.  
   **More Information**: *https://www.owasp.org/index.php/OWASP\_Periodic\_Table\_of\_Vulnerabilities\_-\_Directory\_Indexing*
4. The application shall generate strong random anti-CSRF tokens unique to the transaction as part of any transaction that modifies the application’s data or internal state. This token shall be part of the form (as a hidden field) and not be stored in the cookie. The application shall verify the presence of this token with the proper value for the current user when processing these requests. If the tokens match, the request can be processed; otherwise, the application shall issue an error indicating the request is invalid. The tokens shall have sufficient entropy (same as session ID).   
   **More Information**: *https://www.owasp.org/index.php/Cross-Site\_Request\_Forgery\_(CSRF)\_Prevention\_Cheat\_Sheet*
5. All access control decisions (including failed decisions) shall be logged. Logs shall contain time of check, time of logging, user ID wanting to access the resource, resource being accessed, whether access was granted, the source IP address, and a description.  
   **More Information**: *https://www.owasp.org/index.php/Logging\_Cheat\_Sheet*
6. An application shall not provide direct access to internal implementation objects based on user-supplied input, but rather verify whether the user should have access to the requested internal object.  
   **More Information**: *https://www.owasp.org/index.php/Testing\_for\_Insecure\_Direct\_Object\_References\_(OTG-AUTHZ-004)*

## Input Handling

Software systems and components commonly make assumptions about data they operate on. It is important to explicitly ensure that these assumptions hold: vulnerabilities frequently arise from implicit assumptions about data, which can be exploited if an attacker can subvert and invalidate them.

As such, it is important to design software systems to ensure that comprehensive data validation actually takes place and that all assumptions about data have been validated when they are used. Such data validation is usually called input validation.

The two main approaches for input validation are whitelisting and blacklisting. A whitelist approach defines what data should be accepted by the application for a given input point. In other words, it defines a set of “*known good inputs*”. The blacklist approach will attempt to do the opposite by defining a set of “*known bad inputs*”. Whitelisting shall be used, as it is considered good security practice. Blacklisting may be used as a fallback approach if whitelisting is not technically feasible.

The following input handling security standards protect against common vulnerabilities and flaws:

1. All input validation or encoding routines shall be performed on the server side. Preferably framework features shall be used when possible. Input validation can also be performed client-side for usability reasons., A separate input validation control shall be used by the application for each type of data that is accepted. **More Information**: *https://www.owasp.org/index.php/Input\_Validation\_Cheat\_Sheet*
2. Input validation shall not only be executed close to the entry point (e.g. web service, web page), but also across each trust boundary. When data crosses a trust boundary, its trust level should go to zero and the data must be revalidated.  
   **More Information**:[*https://www.owasp.org/index.php/Input\_Validation\_Cheat\_Sheet*](https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet)and *http://cybersecurity.ieee.org/center-for-secure-design/define-an-approach-that-ensures-all-data-are-explicitly-validated.html*
3. All user-controlled data (parameters, cookies, file names and path data) obtained from untrusted sources shall be canonicalized, input validated, and output encoded. Additionally, there shall be an upper bound on the data size of each data item.  
   **More Information**:   
   *http://cybersecurity.ieee.org/center-for-secure-design/define-an-approach-that-ensures-all-data-are-explicitly-validated.html* *https://www.owasp.org/index.php/XSS\_(Cross\_Site\_Scripting)\_Prevention\_Cheat\_Sheet#Output\_Encoding\_Rules\_Summary*
4. Input validation failures shall result in input rejection. For instance, an application that receives an input string that is invalid shall reject that input and return a user-understandable error message.  
   **More Information**: *http://cybersecurity.ieee.org/center-for-secure-design/define-an-approach-that-ensures-all-data-are-explicitly-validated.html*
5. The application shall use (framework) controls to avoid injection vulnerabilities, such as SQL injection, LDAP injection, Cross-site scripting, XML injection, or OS command injection.  
   **More Information**: *https://www.owasp.org/index.php/SQL\_Injection\_Prevention\_Cheat\_Sheet  
   https://www.owasp.org/index.php/Injection\_Prevention\_Cheat\_Sheet*
6. All untrusted data that is output to the user (e.g. HTML including HTML elements, HTML attributes, JavaScript data values, CSS blocks, and URI attributes) shall be properly escaped/encoded for the applicable context. For instance, the system shall use HTML Entity encoding for HTML, HTML Attribute Encoding for HTML attributes, URL encoding for URLs, JavaScript encoding for JavaScript, and CSS Hex Encoding for CSS-files.  
   **More Information**: *https://www.owasp.org/index.php/XSS\_(Cross\_Site\_Scripting)\_Prevention\_Cheat\_Sheet#Output\_Encoding\_Rules\_Summary*
7. Input validation failures shall be logged. The log shall contain the time of validation, the time of logging, the username, the input data, a description, and the source IP address. The input data itself shall only be logged after output encoding.  
   **More Information**: *https://www.owasp.org/index.php/Logging\_Cheat\_Sheet* and *http://cybersecurity.ieee.org/center-for-secure-design/define-an-approach-that-ensures-all-data-are-explicitly-validated.html*
8. Input handling libraries shall be configured to use secure-defaults, e.g. XML parsing libraries shall be configured to not expand external entities.  
   **More Information**: *https://www.owasp.org/index.php/XML\_External\_Entity\_(XXE)\_Processing*
9. The application shall use implementation-language-level types when available to capture assumptions of data validity. For example, an application that receives as an input a date and time in string representation should validate that this input indeed consists of a well-formed string representation of a date and time (e.g., in ISO 8601 format). It is desirable to implement validation by parsing the input into a typed representation (such as a "*date and time*" type provided in many programming language's standard libraries), and to use that typed representation (and not the original input string) throughout the program.  
   **More Information**: *http://cybersecurity.ieee.org/center-for-secure-design/define-an-approach-that-ensures-all-data-are-explicitly-validated.html*

## Error Handling and Logging

Applications should strive to handle 100% of the errors thrown within an application for three reasons:

* Motivated attackers can use information in error messages to execute targeted attacks.
* Error messages may leak privacy related information.
* An attacker may be able to exploit an application that is in an unknown state to access unauthorized functionality, or to create, modify, or destroy data.

Applications use a logging mechanism for debugging, forensic evidence in case of a breach, and attack detection amongst others. Logs can contain different kinds of data. The selection of the data used is normally affected by the motivation leading to the logging. In general, the logging features include appropriate debugging information such as when, where, who, and what. The properties of these depend on the class of application, but often include the following:

When:

* Log date and time
* Event data and time
* Interaction identifier

Where:

* Application identifier
* Application address
* Service
* URL
* Code location

Who:

* User identity (e.g. user id)
* Source address

What:

* Type of event
* Severity of event
* Security relevant
* Description

The following logging security standards need to be followed:

1. The application shall maintain a separate security log in which security related events shall be logged.  
   **More Information**: *https://www.owasp.org/index.php/Logging\_Cheat\_Sheet*
2. The application shall not display to users error messages or stack traces containing sensitive data that could assist an attacker, including session ID, passwords, and personal identifiable information (PII). Sensitive data is defined in the risk-rating document.  
   **More Information**: *https://www.owasp.org/index.php/Logging\_Cheat\_Sheet*
3. All logging controls shall be implemented on the server. Logging shall be performed before and after executing sensitive transactions (see glossary). Examples of sensitive transactions include payment transactions, transactions that manage user provisioning, and transactions that manage health programs. If logging fails, the transaction fails.  
   **More Information**: *https://www.owasp.org/index.php/Logging\_Cheat\_Sheet*
4. Each security log event shall include: a timestamp, severity level of the event, an indication that this is a security relevant event (if mixed with other logs), the identity of the user that caused the event (if there is a user associated with the event), the source IP address of the request associated with the event, whether the event succeeded or failed, and a description of the event.  
   **More Information**: *https://www.owasp.org/index.php/Logging\_Cheat\_Sheet*
5. A security log analysis tool shall be implemented. This tool shall allow the analyst to search for log events based on combinations of search criteria across all fields in the log record format supported by the system.  
   **More Information**: *https://www.owasp.org/index.php/Logging\_Cheat\_Sheet*
6. Logs shall be collected by a single centralized application-level logging component as centralized logging protects the integrity, availability, and confidentiality of application event and audit logs. Moreover, centralized logging improves time-to-recover in the case of an outage or security incident. For this reason, many security operations standards such as PCI DSS, the Center for Internet Security's system configuration benchmarks, and many SANS institute guides require it for compliance purposes.  
   **More Information**: *https://www.owasp.org/index.php/Logging\_Cheat\_Sheet*
7. All non-printable symbols and field separators shall be output encoded to the appropriate context (i.e. depending on the log format).  
   **More Information**: *https://www.owasp.org/index.php/Logging\_Cheat\_Sheet  
   https://www.owasp.org/index.php/Log\_injection*

## Cryptography and Data Protection

This section focuses on protection of stored data by cryptographic techniques. It is meant for developers and architects who are using cryptographic solutions and will not focus on the design of cryptographic algorithms.

The data handled by the web application must be protected when stored both at the client side as well as at the server side. Protecting stored data – also called data at rest – and protecting data over a communication channel – also called data in transit - are done by applying cryptographic algorithms. Section 3.8 enumerates security standards for data in transit.

Cryptographic algorithms can provide one or more of the following four services. For protection of stored data, one typically uses integrity and confidentiality.

* **Authentication**: establishes the identity of a remote user or an entity. A typical example is the SSL certificate of a web server providing proof to the user that he or she is connected to the correct server.
* **Non-Repudiation**: proves that a unique user has made a transaction request. It must not be possible for the user to refute his or her actions.
* **Confidentiality:** keeps information private. Whether it is passwords sent during a log on process, or storing confidential medical records in a database, encryption can assure that only users who have access to the appropriate key will get access to the data.
* **Integrity:** ensures that data is not viewed or altered during storage or transmission.

Cryptographic techniques are symmetric cryptography, asymmetric cryptography, hashes, message authentication code, digital signatures, and key exchange algorithms.

* **Symmetric Cryptography:** in a symmetric cryptosystem,the involved parties share a common secret (password, pass phrase, or key). Data is encrypted and decrypted using the same key. Common examples of symmetric algorithms are 3DES and AES.
* **Asymmetric Cryptography**: Asymmetric algorithms use two keys, one to encrypt the data, and either key to decrypt. These inter-dependent keys are generated together. One is labeled the Public key and is distributed freely. The other is labeled the Private Key and must be kept secret. Example algorithms include RSA.
* **Hash Functions:** take some data of an arbitrary length and generate a fixed-length hash based on this input. Hash functions used in cryptography have the property that it is easy to calculate the hash, but computationally infeasible to re-generate the original input if only the hash value is known. Common example of hash functions are SHA-256.
* **Key Exchange Algorithms:** enables a party to safely exchange encryption keys with an unknown party.
* **Message authentication Codes (MAC):** MAC algorithms are used to authenticate a message exchanged between two entities. HMAC or CMAC are common examples of MAC algorithms.
* **Digital Signatures:** Just like the MAC algorithms, the digital signatures are used to authenticate messages exchanged between two entities. The additional protection provided by the digital signature algorithms is the non-repudiation.

For data at rest, one typically uses symmetric cryptography and hash functions. For data in transit, one typically uses a secure communication protocol, such as TLS, that combines many of the above cryptographic techniques (see Section 3.8). This section enumerates security standards for data at rest, while section 3.8 enumerates standards for data in transit.

The following cryptography security standards need to be understood and followed when using cryptography to protect data at rest:

1. All cryptographic functions used to protect server-side application secrets from the user shall be implemented server side, preferably by a central component.  
   **More Information**: *https://www.owasp.org/index.php/Cryptographic\_Storage\_Cheat\_Sheet*
2. The system shall use cryptographic libraries to perform cryptographic operations. For systems/products that are to be used by US government agencies, the system shall use a cryptographic library implementation that is FIPS 140-2 certified.  
   **More Information**: *https://www.owasp.org/index.php/Cryptographic\_Storage\_Cheat\_Sheet*
3. Cryptographic keys shall not be hardcoded in the source code, but stored in a protected location, typically configuration files containing the encrypted shared secret. Access to the configuration files and encryption keys shall be controlled by operating system access controls.  
   **More Information**: *https://www.owasp.org/index.php/Cryptographic\_Storage\_Cheat\_Sheet*
4. All random numbers, random file names, random GUIDs, and random strings shall be generated using the cryptographically secure random number generation algorithms.  
   **More Information**: *https://www.owasp.org/index.php/Cryptographic\_Storage\_Cheat\_Sheet*
5. The system shall allow for cryptographic algorithm adaption and evolution. As weaknesses are discovered in cryptographic algorithms over time, the system shall support the ability to change the algorithm parameters such as the key, the key length or the ability to swap old (insecure) algorithms with different (more secure) algorithms.   
   **More Information**: *https://www.owasp.org/index.php/Cryptographic\_Storage\_Cheat\_Sheet*
6. All cached or temporary copies of sensitive data stored on the server shall be protected from unauthorized access or purged/invalidated after the authorized user accesses the sensitive data. Protection means that the sensitive data is encrypted and integrity protected. The keys shall not be stored together with the data. Operating system controls shall be enforced for access to the key data. Additionally, PCI DSS requirements shall be followed if the data is credit card-data.  
   **More Information**: *https://www.owasp.org/index.php/Cryptographic\_Storage\_Cheat\_Sheet*
7. All cached or temporary copies of sensitive data sent to the client shall be protected from unauthorized access or purged/invalidated after the authorized user accesses the sensitive data: the proper ‘*no-cache*’ and ‘*no-store*’ Cache-Control headers should be set and all forms that contain sensitive information such as passwords shall have autocomplete disabled.  
   **More information**: *https://www.owasp.org/index.php/OWASP\_Application\_Security\_FAQ#What.27s\_the\_difference\_between\_the\_cache-control\_directives:\_no-cache.2C\_and\_no-store.3F*
8. If the system requires hash functions, then it shall be one of the following: SHA-256, SHA-384, SHA-512, or RIPEMD-160.  
   **More Information**: *https://www.ethical-hacking.de/pdf/Crypto\_Cheat\_Sheet-Best\_Practices-en.pdf*
9. If the system requires MAC methods, then it shall be one of the following: HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512, MAC-3DES-CBC, or AES-CMAC-256. Each MAC function uses a shared secret. The secret should be generated using cryptographically secure random numbers generation algorithms.  
   **More Information**: *https://www.ethical-hacking.de/pdf/Crypto\_Cheat\_Sheet-Best\_Practices-en.pdf*
10. If the system requires symmetric key algorithms, then it shall be one of the following: AES -128, AES-192, AES-256. The initialization vector shall be a secure cryptographically random number and shall not be reused. These initialization vectors do not need to be secret, but the application may limit their exposure to the encrypting and decrypting instances. The application preferably uses CBC mode and PKCS #7 padding. Care must be taken that both the sender and the recipient use the same character encoding (e.g. UTF-8).  
    **More Information**: *https://www.ethical-hacking.de/pdf/Crypto\_Cheat\_Sheet-Best\_Practices-en.pdf*
11. When CBC mode of encryption is used, the system shall not output any padding errors (this is the root cause of padding oracle attack on the CBC mode of encryption).
12. If the system requires asymmetric key algorithms, then it shall be one of the following: RSA with key size of at least 2048 bit. Note that asymmetric encryption should not be used to encrypt large blocks of data. It is better to encrypt the data with a symmetric key and then encrypt the symmetric key with asymmetric encryption.  
    **More Information**: *https://www.ethical-hacking.de/pdf/Crypto\_Cheat\_Sheet-Best\_Practices-en.pdf*
13. If the system requires secure signature algorithms, then it shall be one of the following: RSA with at least 2048 bit key size, DSA-1024, ECDSA-256, ECDSA-384, or ECDSA-512.   
    **More Information**: *https://www.ethical-hacking.de/pdf/Crypto\_Cheat\_Sheet-Best\_Practices-en.pdf*
14. If the system requires a key-exchange algorithm, then it shall be Diffie-Hellman.   
    **More Information**: *https://www.ethical-hacking.de/pdf/Crypto\_Cheat\_Sheet-Best\_Practices-en.pdf*

## Secure Communication

Sensitive data should be protected when it is communicated between two entities over a network. Communication between two parties is considered secure when the communication

* Cannot be sniffed, captured, and/or intercepted,
* Cannot be tampered in transit,
* Cannot be replayed,
* Originator can be ascertained or authenticated.

There exist two major options to protect data in transit, namely SSL/TLS and Virtual Private Networks. An architect selects an option based on the business needs of a particular organization. For example, a VPN connection may be the best design for a partnership between two companies that includes mutual access to a shared server over a variety of protocols. Conversely, an Internet facing enterprise web application would likely be best served by a SSL/TLS model.

This section focuses on the usage of SSL/TLS, as this option is frequently used for publically accessible web applications.

The following standards for secure communication need to be understood and followed:

1. TLS shall be used for all connections (including both external and backend connections) that are authenticated or that involve sensitive data or functions (as defined by the risk-rating document). TLS v1.1or TLS v1.2 shall be used with
   1. a cipher of at least 128 bit,
   2. disabled TLS compression,
   3. disabled client-side renegotiation,
   4. a certificate per host.

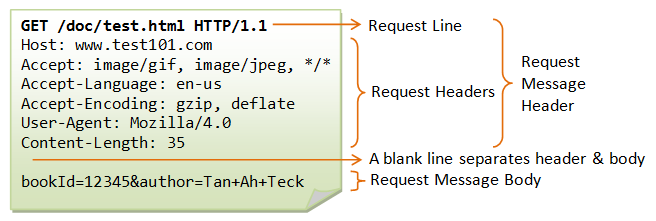
**More Information***: https://www.owasp.org/index.php/Transport\_Layer\_Protection\_Cheat\_Sheet  
https://www.owasp.org/index.php/User\_Privacy\_Protection\_Cheat\_Sheet*

1. It is important that a path is built from a trusted CA to each TLS server certificate, and that each server certificate shall be valid.  
   **More Information**: *https://www.owasp.org/index.php/Transport\_Layer\_Protection\_Cheat\_Sheet  
   https://www.owasp.org/index.php/User\_Privacy\_Protection\_Cheat\_Sheet*
2. Backend TLS connection failures shall be logged. Logs shall contain time of failure, time of logging, host wanting to access the server, server being accessed, the reason of failure, and a description.  
   **More Information**: *https://www.owasp.org/index.php/Logging\_Cheat\_Sheet  
   https://www.owasp.org/index.php/User\_Privacy\_Protection\_Cheat\_Sheet*
3. All sensitive data shall be sent to the server in the HTTP message body (not parameters). The meaning of sensitive data is application-specific and determined by the risk rating document.  
   **More Information**: *https://www.owasp.org/index.php/User\_Privacy\_Protection\_Cheat\_Sheet*

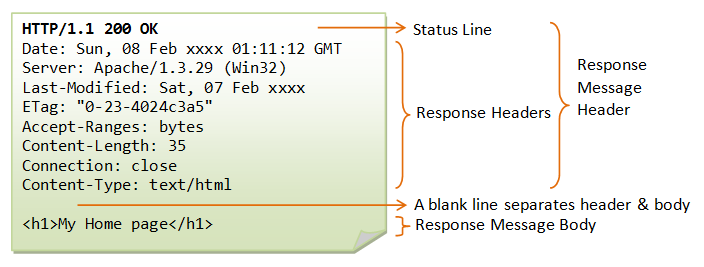
## HTTP Security

HTTP (Hypertext Transfer Protocol) is a protocol used by various web applications. The protocol works as follows. An HTTP client sends a request message to an HTTP server. The server, in turn, returns a response message. Messages typically have a message header and an optional message body.

The request message header consists of a request line and optional request headers. The request line defines the request method name, such as GET, POST, HEAD followed by the URI that specifies the request resource, and the HTTP version.



The response message header consists of a status line followed by optional response headers. The status line contains the HTTP version, the status code, and a short explanation of the status code. The response headers are in the form of key value pairs.



Several intermediate components such as a web application firewall may add request headers to the request before it reaches the web server.

Typical issues with respect to the usage of the HTTP protocol include:

* Disclosure of sensitive information in the headers
* Spoofing of headers added by front-end components such as web application firewalls
* Accepting request headers and request bodies without proper validation

The following HTTP security standards need to be understood and followed in order to protect against the common issues:

1. The application shall only accept a defined set of HTTP request methods, such as GET and POST and unused methods are explicitly blocked.  
   **More Information**: *https://www.owasp.org/index.php/Testing\_for\_HTTP\_Methods\_and\_XST\_%28OWASP-CM-008%29#Arbitrary\_HTTP\_Methods*
2. Every HTTP response shall contain a content type header specifying a safe character set (e.g., UTF-8) of the response body.   
   **More Information**: *https://www.owasp.org/index.php/OWASP\_Testing\_Guide\_Appendix\_D:\_Encoded\_Injection#Output\_Encoding\_.E2.80.93\_Server\_.26\_Browser\_Consensus*
3. HTTP headers added by an application frontend such as an intercepting proxy or a web application firewall, and used by the (web) application, shall not be able to be spoofed by the end user; i.e. the frontend shall remove any such headers originating from the user before re-adding them.
4. The HTTP header, ‘*X-Frame-Options*’ shall be used for sites where content should not be viewed in a 3rd-party X-Frame. The system shall preferably use ‘*DENY*’, but a common middle ground is to send ‘*SAMEORIGIN*’, meaning only websites of the same origin, i.e. the current site, may frame it.  
   **More Information**: *https://www.owasp.org/index.php/List\_of\_useful\_HTTP\_headers  
   https://www.owasp.org/index.php/Clickjacking\_Defense\_Cheat\_Sheet*
5. HTTP headers shall not expose detailed version information of system components. Examples include the name and version number of the server software or the application framework.  
   **More Information**: *http://www.ipponusa.com/owasp-top-10-a5/*

## Operational Security

IT security covers all the processes and mechanisms by which computer-based equipment, information and services are protected from unintended or unauthorized access, change or destruction. IT security is paramount for application security as secure applications will not be secure if they are running in an insecure environment. The rest of this section elaborates on some security standards that are critical for credentials and log files. However, this section is not meant to be exhaustive.

1. Access to server application code, configuration files and encryption keys shall be controlled by operating system access controls and access shall be monitored. Keys may be stored by a (dedicated) hardware security module.  
   **More Information**: *https://www.owasp.org/index.php/Authentication\_Cheat\_Sheet* and *https://www.owasp.org/index.php/Key\_Management\_Cheat\_Sheet  
   http://en.wikipedia.org/wiki/Hardware\_security\_module*
2. Secure Log files should be in append-only mode  
   **More Information**: *https://www.owasp.org/index.php/Logging\_Cheat\_Sheet*
3. Copies of log files should be made at regular intervals depending on volume and size (daily, weekly, monthly, etc.). A common naming convention should be adopted with regards to logs, making them easier to index.  
   **More Information**: *https://www.owasp.org/index.php/Logging\_Cheat\_Sheet*
4. Log files should be copied and moved to permanent storage and incorporated into the organization's overall backup strategy.  
   **More Information**: *https://www.owasp.org/index.php/Logging\_Cheat\_Sheet*
5. Log files and media should be deleted and disposed of properly and incorporated into an organization's shredding or secure media disposal plan. Reports should be generated on a regular basis, including error reporting and anomaly detection trending.  
   **More Information**: *https://www.owasp.org/index.php/Logging\_Cheat\_Sheet*
6. All logging components should be synced with a timeserver so that all logging can be consolidated effectively without latency errors. This time server should be hardened and should not provide any other services to the network.  
   **More Information**: *https://www.owasp.org/index.php/Logging\_Cheat\_Sheet*

# Glossary

Most definitions are taken from: *http://www.sans.org/security-resources/glossary-of-terms/*

**Access Control** is the degree to which the system limits access to its resources only to its authorized externals (e.g., human users, programs, processes, devices, or other systems).

**Access Control List:** see ACL.

**Access Control Service** is a security service that provides protection of system resources against unauthorized access. The two basic mechanisms for implementing this service are ACLs and tickets.

**ACL:** A mechanism that implements access control for a system resource by listing the identities of the system entities that are permitted to access the resource.

**Advanced Encryption Standard (AES):** a symmetric encryption algorithm.

**Asset:** anything of value that should be protected from harm. Assets can be people, properties (e.g., data, hardware, software, and facilities), and services.

**Asymmetric Cryptography**: Public-key cryptography; A modern branch of cryptography in which the algorithms employ a pair of keys (a public key and a private key) and use a different component of the pair for different steps of the algorithm.

**Attack** is an attacker’s unauthorized attempt to cause harm to an asset (i.e., violate the security of the system, bypass security mechanisms). An attack may be either successful or unsuccessful.

**Attacker** is an agent (e.g., humans, programs, processes, devices, or other systems) that causes an attack due to the desire to cause harm to an asset.

**Availability:** The probability that an item will operate satisfactorily at a given point in time when used in an actual or realistic operating and support environment. It includes logistics time, ready time, and waiting or administrative downtime, and both preventive and corrective maintenance downtime.

**Authentication** establishes the identity of a remote user or an entity. A typical example is the SSL certificate of a web server providing proof to the user that he or she is connected to the correct server.

**Authentication Service**: A service that offers authentication controls to applications or servers.

**Authorization** is the degree to which access and usage privileges of authenticated externals are properly granted and enforced.

**Brute force** is a cryptanalysis technique or other kind of attack method involving an exhaustive procedure that tries all possibilities, one-by-one.

**Canonicalization[[2]](#footnote-2):** is a process for converting [data](http://en.wikipedia.org/wiki/Data) that has more than one possible representation into a standard, normal, or [canonical form](http://en.wikipedia.org/wiki/Canonical_form). This can be done to compare different representations for equivalence, to count the number of distinct data structures, to improve the efficiency of various [algorithms](http://en.wikipedia.org/wiki/Algorithm) by eliminating repeated calculations, or to make it possible to impose a meaningful [sorting](http://en.wikipedia.org/wiki/Sorting) order.

**Client:** A system entity that requests and uses a service provided by another system entity, called a "server." In some cases, the server may itself be a client of some other server.

**Confidentiality:** the degree to which sensitive information is not disclosed to unauthorized parties (e.g., individuals, programs, processes, devices, or other systems).

**Dictionary Attack**: An attack that tries all of the phrases or words in a dictionary, trying to crack a password or key. A dictionary attack uses a predefined list of words compared to a brute force attack that tries all possible combinations.

**Diffie-Hellman:** A key agreement algorithm published in 1976 by Whitfield Diffie and Martin Hellman. Diffie-Hellman does key establishment, not encryption. However, the key that it produces may be used for encryption, for further key management operations, or for any other cryptography.

**Digital Signature:** A digital signature is a hash of a message that uniquely identifies the sender of the message and proves the message hasn't changed since transmission

**Direct Authentication:**

**Encryption:** Cryptographic transformation of data (called "plaintext") into a form (called "cipher text") that conceals the data's original meaning to prevent it from being known or used.

**Hash:** the output of a hash function.

**Hash function**: are used to generate a one way "check sum" for a larger text, which is not trivially reversed. The result of this hash function can be used to validate if a larger file has been altered, without having to compare the larger files to each other. Frequently used hash functions are MD5 and SHA1.

**Harm:** is a negative impact associated with an asset due to an attack.

**HTTP[[3]](#footnote-3)**: The Hypertext Transfer Protocol (HTTP) is an [application protocol](http://en.wikipedia.org/wiki/Application_protocol) for distributed, collaborative, [hypermedia](http://en.wikipedia.org/wiki/Hypermedia) information systems. HTTP is the foundation of data communication for the [World Wide Web](http://en.wikipedia.org/wiki/World_Wide_Web).

**HTTPS**: When used in the first part of a URL (the part that precedes the colon and specifies an access scheme or protocol), this term specifies the use of HTTP enhanced by a security mechanism, which is usually SSL/TLS.

**Identity:** Identity is whom someone or what something is, for example, the name by which something is known.

**Input Validation[[4]](#footnote-4)**: is the process of ensuring that a program operates on clean, correct and useful data. It uses routines, often called [validation rules](http://en.wikipedia.org/wiki/Validation_rule) that check for correctness, meaningfulness, and security of data that are input to the system. The rules may be implemented through the automated facilities of a [data dictionary](http://en.wikipedia.org/wiki/Data_dictionary), or by the inclusion of explicit [application program](http://en.wikipedia.org/wiki/Application_program) validation logic.

**Integrity:** the need to ensure that information has not been changed accidentally or deliberately, and that it is accurate and complete.

**Key Exchange Algorithms:** enables a party to safely exchange encryption keys with an unknown party.

**Message authentication Codes (MAC):** MAC algorithms are used to authenticate a message exchanged between two entities. HMAC or CMAC are common examples of MAC algorithms.

**Nonrepudiation** is the degree to which a party to an interaction (e.g., message, transaction, transmission of data) is prevented from successfully repudiating (i.e., denying) any aspect of the interaction.

**RBAC**: Role based access control assigns users to roles based on their organizational functions and determines authorization based on those roles.

**Response**: information sent as a response to some stimulus.

**Risk**: the product of the level of threat with the level of vulnerability. It establishes the likelihood of a successful attack.

**Role Based Access Control**: see RBAC.

**Salt**: A fixed-length (32 to 64 bit) cryptographically-strong random value.

**Security** is the degree to which malicious harm to a valuable asset is prevented, reduced, and properly responded to. Security is thus the quality factor that signifies the degree to which valuable assets are protected from significant threats posed by malicious attackers.

**Security Mechanism** (a.k.a., countermeasure) is an architectural mechanism (i.e., strategic decision) that helps fulfill one or more security requirements and/or reduces one or more security vulnerabilities. Security mechanisms can be implemented as some combination of hardware or software components, manual procedures, training, etc.

**Security Requirement** is a quality requirement that specifies a required amount of security (actually a quality subfactor of security) in terms of a system-specific criterion and a minimum level of an associated quality measure that is necessary to meet one or more security policies.

**Security Vulnerability** is any weakness in the system that increases the likelihood of a successful attack (i.e., cause harm).

**Sensitive Transaction** is a transaction that either directly manipulates sensitive data or accesses sensitive business functionality. The exact meaning of sensitive is application-specific and thus out-of-scope for this document. Examples of sensitive transactions include application functionality that reads or modifies non-public personally identifiable information (NPPI) such as patient information and healthcare programs data, functionality that modifies access control information such as user roles or permissions, and functionality that handles payment transactions.

**Server:** system entity that provides a service in response to requests from other system entities called clients.

**Session:** a sequence of request and response transactions.

**Session ID:** token to identify a session.

**SQL Injection:** SQL injection is a type of input validation attack specific to database-driven applications where SQL code is inserted into application queries to manipulate the database.

**Symmetric Cryptography:** A branch of cryptography involving algorithms that use the same key for two different steps of the algorithm (such as encryption and decryption, or signature creation and signature verification). Symmetric cryptography is sometimes called "secret-key cryptography" (versus public-key cryptography) because the entities that share the key.

**Threat**: A potential for violation of security, which exists when there is a circumstance, capability, action, or event that could breach security and cause harm.

**Virtual Private Network**: see VPN.

**VPN**: A restricted-use, logical (i.e., artificial or simulated) computer network that is constructed from the system resources of a relatively public, physical (i.e., real) network (such as the Internet), often by using encryption (located at hosts or gateways), and often by tunneling links of the virtual network across the real network. For example, if a corporation has LANs at several different sites, each connected to the Internet by a firewall, the corporation could create a VPN by (a) using encrypted tunnels to connect from firewall to firewall across the Internet and (b) not allowing any other traffic through the firewalls. A VPN is generally less expensive to build and operate than a dedicated real network, because the virtual network shares the cost of system resources with other users of the real network.

1. Ferraiolo, D.F. and Kuhn, D.R. (October 1992). “Role-Based Access Control”. 15th National Computer Security Conference. pp 554-563. [↑](#footnote-ref-1)
2. http://en.wikipedia.org/wiki/Canonicalization [↑](#footnote-ref-2)
3. http://en.wikipedia.org/wiki/Hypertext\_Transfer\_Protocol [↑](#footnote-ref-3)
4. http://en.wikipedia.org/wiki/Data\_validation [↑](#footnote-ref-4)