PRODUCT

Security Architecture and Design Guidelines

Document History

Version	Date	Author / Changed By	Change Description
0.1	04/02/2015	Vinod Raghavan	First Draft. Further sections on Web/HTTP Security, Mobile security to be added. Also reference guide for sources of more information on various topics mentioned.
0.2	01/03/2015	Vinod Raghavan	Updated after review comments. Added mobile security and data integrity sections and added external resources.
0.3	18/03/2015	Vinod Raghavan	Updated after comments from Prema Varadhan
1.0	23/03/2015	Vinod Raghavan	Baselined Version – Approved by Mark Winterburn
1.1	14/05/2015	Santhosh Kumar K	Minor updates and additions from further reviews: - Added links to references in the Appendix (3.2.5, 5.2.1, 5.2.7) - Added guideline 2.2.11, 2.2.12, 2.3.5 - Additional Mobile security requirements.
2.0	15/05/2015	Vinod Raghavan	Changes approved

Table of Contents

1. Overview

This guideline document aims to give an understanding of the elements that should be covered in order to ensure a good level of security for Temenos products. The aim is that this will be a reference point for product and development management when creating requirements for, and designing new products, modules or applications. It should also give understanding of where improvements can be made in existing products and will be the basis for security reviews by the Security Analysis Team. These reviews will be undertaken either by the security team or in conjunction with Internal Audit and will be performed on the basis of risk assessments (E.g. Channels products are usually considered most high risk)

1.1 Scope

This document describes the functionality that should be considered and implemented in order to ensure security of a product. It does not enforce specific technologies or how they should be implemented as this depends on the platform and technology guidelines for individual products. The aim of the document is to raise the issues and guidance can be sought from industry standards to address any product issues. The security analysis team can also assist with sources of information for various technologies.

Each section deals with an area related to security that should be addressed in every product and SI development and can be used as the basis of security requirements. It is acknowledged that not all of the areas raised here will have been dealt with in the past, but we should endeavour to gradually improve existing products according to criticality of missing features. To this end, the Security Analysis team is going through a program of Security Risk Analysis assessments in order to find rate the criticality of missing features in various products. The risk assessment are carried out with the product teams and are similar in nature to a technical security audit.

2. Authentication

2.1 Overview

The process of validating an entity's claim is authentication. The entity may be a person, a process or a hardware device. The common means by which authentication occurs is that the entity provides identity claims and/or credentials which are validated and verified against a trusted source holding those credentials.

This section specifies architecture and design elements that are mandatory and best practice.

2.2 Mandatory

2.2.1 The system should verify a user's access to all pages and resources unless they are explicitly identified as being public resources.

2.2.2 All password fields should be masked

2.2.3 All authentication should take place on the server, not on the client machine

2.2.4 Any authentication control should fail securely, i.e. if authentication fails, prohibit access by displaying a generic error message which prevents identification of valid usernames.

2.2.5 It must be possible to mandate strong passwords, i.e. minimum length, mixture of uppercase, lowercase, numeric and special characters.

2.2.6 The processes for password management must be specified, e.g. user registration, changing passwords, changing passwords on expiry, forgotten passwords and lost tokens (in the case of 2-factor authentication)

2.2.7 All successful/failed authentication attempts should be logged along with IP and timestamp for auditing purposes.

2.2.8 Users must be able to change their passwords as well as changing them on expiry

2.2.9 Passwords must be hashed when stored using a secure algorithm (e.g. SHA-256, and not SHA-1 or MD5). The process for hashing should also use a salt (random element) to ensure that the hashed value cannot be determined using rainbow tables.

2.2.10 Passwords and other user credentials should not be sent over unencrypted channels

2.2.11 Random or pseudo-random mechanisms should be used when generating system generated passwords.

2.2.12 Hard coded credentials or cryptographic keys in clear text must not be included in inline code, or in configuration files.

2.3 Best Practice

2.3.1 Forgotten password mechanisms must not reveal the existing password, but should securely enable the user to reset the password.

2.3.2 There should be no default passwords for administration accounts that could potentially be left in place in production systems. Such passwords should ideally be set during installation.

2.3.3 The system should remember a certain number of the user's previous passwords in order to prevent regular reuse of passwords.

2.3.4 For sensitive transactions it should be possible to "step-up" authentication to a higher level. This should mean adding an additional authentication request using a different factor (e.g. One-time password token, Smartcard certificate or biometric)

2.3.5 When setting passwords, a password strength checker should be included which measures the effectiveness of a password in resisting guessing, dictionary and brute-force attacks.

3. Session Management

3.1 Overview

Session management ensure that once a session is established, it remains in a state that it will not compromise the security of the software. All user interfaces to a system must have a session management mechanism in order to control access to the system once a user has been authenticated.

3.2 Mandatory

3.2.1 Session management control must be used by the application. Any information related to the session must be isolated from other user sessions

3.2.2 Sessions must be invalidated after the user logs out, or timeout after a set (configurable) period of inactivity.

3.2.3 It must be possible to logout from any page of the application (the logout link must always be visible)

3.2.4 Web session Id's must never be disclosed other than in secured cookie headers.

3.2.5 Session tokens must be generated in random or pseudo-random mechanism. See OWASP guideline.

3.2.6 Web session cookies over HTTP must be marked with the "secure" and "HttpOnly" attributes. Most web servers will encrypt the contents of the cookie once it is marked as secure.

3.2.7 Web session ids should not include credentials or claims that can be easily spoofed and replayed. E.g. IP address, MAC address, DNS or reverse-DNS lookups or referrer headers

3.2.8 Session Id's should be changed after authenticating to the system. This prevents a malicious user gaining information from an unauthenticated session and using it to gain access to an authenticated session.

3.3 Best Practice

3.3.1 Applications should not allow concurrent sessions for a single user as they encourage multiple users to use single accounts. In addition they causes confusion on license use.

4. Access Control (Authorization)

4.1 Overview

Access Control confirm that an authenticated entity has the needed rights and privileges to access and perform actions on a requested resource. This section deals with best practices for access control to systems. It must be possible to limit what a user can do once authenticated to a system.

4.2 Mandatory

4.2.1 The system must have the ability to specify functions that a role or user can perform.

4.2.2 The system must have the ability to specify data that a role or user can view or modify.

4.2.3 The system must be able to limit access to downloadable files stored in the system (e.g. account statements, application forms etc). Users should only be able to download such information if they have the correct permissions.

4.2.4 All access control rules and permissions must be implemented on the server side.

4.2.5 It must not be possible for system users (business users) to modify access control rules and permissions.

4.2.6 Restrict user access to system-level resources, System level resources include files, folders, registry keys, Active Directory objects, database objects, event logs, and so on.

4.2.7 If access control fails, it must fail securely, i.e. by prohibiting access and log for audit trails.

4.2.8 It must be possible to review logs for access control decisions.

4.2.9 The system must not expose a reference to an internal implementation object, such as a file, directory or database key. Instead use an index of the value or a reference map so that direct parameter manipulation of internal objects are prohibited.

4.2.10 The system must not expose internal objects directly via URLs or form parameters to the end user.

4.3 Best Practice

4.3.1 Any access control implemented in the presentation layer must also be implemented on the server to prevent circumvention of access control rules.

4.3.2 Access control rules must be specified in a centralized system in order that the rules are kept consistent.

5. Data Input Verification

5.1 Overview

All input to a system must be verified as valid. There are several types of malicious input that could be attempted by a rogue user, and these should be mitigated against.

5.2 Mandatory

5.2.1 The runtime environment must not be susceptible to buffer overflows. Security controls should protect against buffer overflows.

5.2.2 Any input validation must take place on the server. Client side validation must only ever be an additional validation for usability purposes.

5.2.3 Any failure in input validation must be logged for audit purposes.

5.2.4 All decoding of input data must take place before the verification. If data is encoded it poses a risk that a malicious user is trying to hide dangerous scripts by encoding.

5.2.5 All input data must be checked for SQL injection attacks. These types of attacks have a risk of allowing unintended and unauthorized database queries and operations.

5.2.6 All input data must be checked for OS command injection attacks. These types of attacks have a risk of allowing unintended and unauthorized access to restricted OS files and operations.

5.2.7 All input data must be checked for XML injection attacks. These types of attacks have a risk of allowing unintended and unauthorized data into XML structures that could potentially be executed as a script processed by the system or by a web browser.

5.2.8 Any non-validated data that is output to HTML must be escaped in order to prevent malicious scripts being executed. This can happen if data entered by a user is directly sent to be displayed by a web application.

5.2.9 If input data is to accept file names to file paths and URLs, which are prone to canonicalization attack, they should be strictly formed and before making security decisions such as granting or denying access to the specified file.

5.3 Best Practice

5.3.1 All input data must be checked for LDAP injection attacks. These types of attacks have a risk of allowing unintended and unauthorized LDAP queries and operations if the system uses an LDAP as a user and/or permissions repository.

5.3.2 If the source of data stored in a system cannot be guaranteed to have been validated on input, then the system must encode data to prevent any malicious scripts being executed at the user interface.

6. Data Cryptography

6.1 Overview

One of the most important ways of protecting sensitive data is to encrypt it to avoid unauthorized users and systems from gaining access. As well as ensuring proper encryption strength is used based on the sensitivity of data, it is also important to ensure that encryption keys are kept securely.

Deciding what encryption algorithms depends on many considerations such as the purpose of encryption (data in transit/ or stationary data), how long the data will be encrypted for, how frequently the data will be encrypted/decrypted, performance requirements for high volumes of data etc. This should be discussed at design time to make an informed choice.

6.2 Mandatory

6.2.1 All cryptographic operations must take place on the server. If this is not done it may be possible to bypass cryptographic processes and keys may be compromised.

6.2.2 If any cryptographic process fails, then it must do so securely. E.g. if encryption of data fails then it must not be stored in an unencrypted form.

6.2.3 Ensure that any keys or secrets used to protect other keys or configuration files, is protected from unauthorized access.

6.2.4 If using any form of key based cryptography there must be key management processes in place to determine how keys can be changed either regularly or if keys have been compromised.

6.2.5 All sensitive data must be cryptographically secure when it is stored

6.2.6 The use of a weak or custom developed unvalidated cryptographic algorithm for encryption and decryption must be avoided.

6.2.7 The use of older cryptographic application programming interfaces (APIs) such as CryptoAPI must be avoided when encrypting sensitive data.

6.2.8 The design of the application must take into account the ability to quickly swap cryptographic algorithms as needed.

6.3 Best Practice

6.3.1 Any random numbers generated in the system should be generated using cryptographic modules.

6.3.2 Sensitive should be separated from non-sensitive data (if feasible) using naming conventions and strong types. This makes it easier to detect code segments where data used is unencrypted when it needs to be.

7. Data Integrity

7.1 Overview

It must be possible to prove the integrity of critical data in the system especially when the data has been input by users. If there is a chance that a user may contest that they input the transaction then mechanisms must be put in place to provide the content of a transaction.

7.2 Mandatory

7.2.1 It must be possible for user facing interfaces to generate digital signature of critical or sensitive transactions based on encryption with a unique and personal encryption key held by the user. The key must be a private key with the public key held by the server. Private keys can be held within a user certificate in the users Browser, or more securely on a Smartcard or secure USB key. More recent innovations also allow these to be held on Mobile devices and accessed via technologies such as NFC, but the appropriate design decision should be taken for the product.

7.2.2 It must be possible for the server to validate the digital signatures of transactions to show that a particular transaction was signed using the private key owned by a specific user.

7.2.3 Digital signature of transactions must be stored as part of a read-only audit record to ensure the integrity of digital signature.

7.3 Best Practice

7.3.1 Ensure that the developed code is signed by the author's digital signature. Code signing is the process of digitally signing the code (executables, scripts, etc.) with the digital signature of the code author. When code is signed using the code author's digital signature, a cryptographic hash of that code is generated. This hash is published along with the software when it is distributed.

8. Communications Encryption and Security

8.1 Overview

Any communication involving sensitive data must be encrypted when passed between systems or between separate components of systems. This will prevent the interception of messages by unauthorised users. Often these users are administrators of machines that have access to administer systems, but should not see the data flowing or at rest.

Please note that TLS is mentioned in this section for encrypting channels. It is the recommended mechanism and supersedes SSL.

8.2 Mandatory

8.2.1 It must be possible for communications between systems or components to be configured with either one way or two way (mutual) TLS based on customer preferences. This should be the case for all sensitive and authenticated communications.

8.2.2 When configuring TLS it must be possible for each system or component to verify the Certificate Authority (CA) contained in the TLS certificates.

8.2.3 Any TLS connection failures should be logged

8.2.4 Any connections to Temenos systems from 3rd parties should only allow permissions to perform the task required.

8.3 Best Practice

8.3.1 Any system to system communication should be authenticated.

9. Error Handling, Logging and Auditing

9.1 Overview

This section deals with issues surrounding data that appears in error messages and logs, event logging and audit logging. The aim is to ensure that transactions and security events appear in logs and can be audited. Also that sensitive data should not appear in logs or at least it should be masked to ensure confidentiality and privacy.

9.2 Mandatory

9.2.1 Error messages and stack traces in logs must not contain sensitive information

9.2.2 All logging must take place on the server

9.2.3 All error handling must take place on the server or other trusted devices. Any errors on client devices could be manipulated by a malicious user

9.2.4 All error logs and audit logs must contain a reliable timestamp, severity rating, and an indication that it is a security event if that is the case. Event logging must also contain the source IP or other identifier of the source of the request, success or failure of the event and description of the event.

9.2.5 Security and audit logs must not be modifiable and only viewable by authorized users. Audit logs should preferably reside on a separate server from the application in order to maintain integrity from administration users.

9.2.6 In the event of a failure, sensitive information such as system data or information about the system should not be exposed. For example, stack trace details that include function names etc., instead, generic error messages should be returned to the client.

9.3 Best Practice

9.3.1 Ideally there should be a single logging mechanism for an entire application in order that event logging is coordinated and can be monitored easily.

9.3.2 There should be a logging mechanism for GUI navigation audit trails to keep track of navigation and data accessed.

9.3.3 An index or reference mapping to an error should be used instead of detailed information to prevent sensitive information disclosure. An example of using indices would be to use a Globally Unique Idenfiers (GUID) that map to internal errors but it is the GUID alone that is displayed to the user, informing the user to contact the support line for more assistance. This way, the internal error details are not directly revealed to the end user or to the attacker who could be using them in their reconnaissance efforts as they plan to launch other attacks.

9.3.4 Failure to protect the audit log configuration interfaces can result in an attack going undetected. (If the configuration interfaces to turn logging on or off is left unprotected, an attacker may be able to turn logging off, perform their attack and turn it back on once they have completed their malicious activities)

10. Data Privacy

10.1 Overview

This section deals with ensuring that sensitive and personal information stored in systems is kept securely. There are many worldwide and banking specific regulations that insist on privacy when it comes to Personally Identifiable Information (PII). This is not only information directly about a customer, but also is often interpreted as being data that could indirectly identify a user.

10.2 Mandatory

10.2.1 Sensitive data must never be cached on a client device. This includes any autocomplete features in web browsers.

10.2.2 Sensitive data stored within a system should be identified and policies put in place to control access and encryption.

10.2.3 Any sensitive data sent to a client should be controlled from further access after the session has ended. In a web interface this should be done using Cache-Control headers

10.2.4 There must be mechanisms to securely archive or delete data from a system after a specified period of time in order to comply with data retention regulations.

10.2.5 Removing of sensitive data securely, as sensitive data can still be recovered, even when deleted from workstation. Using secure wiping utilities data should be removed, and make sure old equipment is thoroughly destroyed/de-magnetized to prevent data from being recovered.

10.3 Best Practice

10.3.1 There should be mechanisms available to identify abnormally high numbers of requests and events, especially if the events are high value transactions or system administration tasks.

10.3.2 Data masking mechanism should be used to secure sensitive data, it reduces the risk exposure by preserving data confidentiality.

10.3.3 Sensitive data should be protected using proper data anonymization techniques like Replacement, Suppression and Generalization.

11. Configuration

11.1 Overview

This section deals with ensuring Web application's configuration files security, as securing configuration files which contains credentials such as database connection strings, user accounts, user ids, password. The consequences of a security breach can be severe, because the attacker can get access and ends up running with administrator privileges and has direct access to the entire application.

11.1.1 It is important that configuration files are accessible only by authorized users and administrators.

11.1.2 Encrypt the configuration files which contains sensitive information such as database connection strings or credentials.

11.1.3 Avoid Installation or configuration of unneeded services, ports and protocols, unused pages, and unprotected files and directories

11.1.4 Disable directory listing in the application server

11.1.5 Handle errors explicitly using redirects and error messages so that breach upon any misconfiguration does not result in the disclosure of more information than is necessary.

11.2 Best Practice

N/A

12. Mobile

12.1 Overview

The nature of mobile devices means that specific security constraints have to be considered. The fact that a device can easily be picked up or stolen means that additional safeguards are necessary.

12.2 Mandatory

12.2.1 Applications on mobile devices must validate SSL certificates.

12.2.2 The device id must never be used as a security credential as it can be spoofed.

12.2.3 Sensitive data should not be stored in shared areas of the device, and ideally should not be stored or cached at all.

12.2.4 Sensitive data should not be stored in an on-device SQLLite database.

12.2.5 Encryption keys and passwords must not be hard-coded into applications.

12.2.6 It should not be possible for Applications to be executed on Jailbroken devices.

12.2.7 It should not be possible to leak sensitive data by taking a snapshot of the state of a device (e.g. the snapshot feature in iOS).

12.2.8 All apps should only request the required permissions from the device.

12.2.9 Any crash logs should not contain sensitive data from the applications

12.2.10 Sensitive data must not be logged to device system logs

12.2.11 The application must implement certificate pinning to prevent proxy's intercepting the data.

12.2.12 HTTPS traffic must not be cached on the device as it is likely to contain sensitive data.

12.2.13 Any personal information must be truncated and/or masked to avoid the need to send sensitive personal information to devices.

12.2.14 Any sensitive data that must be stored or cached by the application must be held in a cryptographically secure manner.

12.2.15 The files created on internal storage should be accessible only by that application, this protection is implemented by Android platform

12.2.16 One should avoid using the MODE_WORLD_WRITEABLE or MODE_WORLD_READABLE modes for IPC files because they do not provide the ability to limit data access to particular applications.

12.2.17 If data needs to be shared with other app processes could be achieved using a content provider, which offers read and write permissions to other apps and can make dynamic permission grants on a case-by-case basis.

12.2.18 File/Data should not be created on external storage(as SD cards) are globally readable and writable as its can be removed by the user and also modifiable by any application. So one should not store sensitive information using external storage.

12.2.19 If data needs to be shared with other application, then content providers should be used, as it offers a structured storage mechanism by setting the android:exported=true in the application manifest.

12.2.20 Minimize the number of permissions app requests, not having access to sensitive permissions reduces the risk of inadvertently misusing those permissions, and makes app less vulnerable for attack. If a permission is not required for app to function, then do not request it.

12.2.21 The runtime environment must not be susceptible to buffer overflows. Security Controls should protect against buffer overflows by careful handling pointers and managing buffers. Any input validation must take place on the server. Client side validation must only ever be an additional validation for usability purposes

12.2.22 All input data must be checked for SQL injection attacks. These types of attacks have a risk of allowing unintended and unauthorized database queries and operations.

12.2.23 If access to sensitive data is required, evaluate whether that information must be transmitted to a server, or whether the operation can be performed on the client. Consider running any code using sensitive data on the client to avoid transmitting user data.

12.2.24 While Handling Credentials, minimize the frequency of asking for user credentials – to make phishing attacks more conspicuous, make use of authorization token and refresh it.

12.3 Best Practice

12.3.1 The application binaries should be obfuscated to prevent reverse engineering and information gathering about the application.

12.3.2 Anti-debugging and reverse engineering protection measures should be implemented.

12.3.3 As with data from any untrusted source, should perform input validation when handling data from external storage

12.3.4 Do not store executable or class files on external storage prior to dynamic loading. If app does retrieve executable files from external storage, then the files should be signed and cryptographically verified prior to dynamic loading.

12.3.5 Minimize the number of permissions that app requests, not having access to sensitive permissions reduce the risk of inadvertently misusing those permissions.

12.3.6 Use type safe languages, which reduces the likelihood of input validation issues

12.3.7 Use parameterized queries and limit the permission to read-only or write only, which can reduce the potential harm related to SQL injection.

12.3.8 Minimize the use of APIs that access sensitive or personal user data.

12.3.9 Be careful when writing to on-device logs, in Android, logs are a shared resource and are available to an application with the READ_LOGS permission.

13. Appendix A – Reference Information

This section will be updated regularly.

13.1 Information about potential application vulnerabilities

13.1.1 Open Web Application Security Project – OWASP

https://www.owasp.org/

Contains many security resources including top ten web vulnerabilities, development and testing guides and tutorials.

13.1.2 SANS Institute – Top 25 Most Dangerous Software Errors

http://www.sans.org/top25-software-errors/

13.2 Glossary of Terms

13.2.1 **Buffer overflow**: A buffer overflow occurs when a program or process tries to store more data in a memory buffer (temporary data storage area) than it was intended to hold.

13.2.2 **XML injection attack** occurs when the application does not properly filter or quote special characters or reserved words that are used in XML, allowing an attacker to modify the syntax, contents or commands before execution

http://www.sans.org/security-resources/glossary-of-terms/