November 2018 CONFIDENTIALITY: INTERNAL

APPLICATION PRODUCTS DEPARTMENT

Secure System Development life cycle
Secure coding guidelines

Control

Inquiries	Application Products 'Security Team
Document Sensitivity	Internal
Validity	Immediate
Version	2.0
Last update	November 12th , 2018
Document Location	ONE / Security
Review Frequency	Yearly

PLAN

1. Purpose

This technology agnostic document defines a set of general software security coding practices, that can be integrated into the software development lifecycle. Implementation of these practices will mitigate most common software vulnerabilities.

Generally, it is much less expensive to build secure software than to correct security issues after the software package has been completed, not to mention the costs that may be associated with a security breach.

When using this guide, development teams should start by assessing the maturity of their secure software development lifecycle and the knowledge level of their development staff. Since this guide does not cover the details of how to implement each coding practice, developers will either need to have the prior knowledge or have sufficient resources available that provide the necessary guidance.

A set of documents covering how to implement some of these requirements are provided at the end of this documents in Appendix: External references.

This guide provides coding practices that can be translated into coding requirements without the need for the developer to have an in depth understanding of security vulnerabilities and exploits. However, other members of the development team should have the responsibility, adequate training, tools and resources to validate that the design and implementation of the entire system is secure.

A glossary of important terms in this document, including section headings and words shown in italics, is provided in Definitions.

The Appendix: Free source code security analysis tools provides a list of free and open source tools that can be used to analyze source code to find some security vulnerabilities. This list is provided only for information to the developers and is not endorsed in any way by AXA IT – Application Products Departments. Indeed, an official source code analysis will be delivered in 2018 by the Central Security Team.

For the reader: The IS requirements on application and projects lifecycles are defined in the Application IS Policy.

Important Note: This document is based on the OWASP (Open Web Application Security Project) Secure Coding Practices.

2. Scope

This document applies to:

AXA IT – Application Products Department

AXA IT – Application Products Department external providers, providing IT services.

This document covers requirements on:

Development of Software on end-points *(desktops, laptops, mobile devices)*.

Development of Software on servers and network devices.

3. Exception Management

All exceptions to this document shall be assessed and tracked as per the IS Exception Management Process.

4. Definitions

Abuse Case	Describes the intentional and unintentional misuses of the software. Abuse cases should challenge the assumptions of the system design.
Access Control	A set of controls that grant or deny a user, or other entity, access to a system resource. This is usually based on hierarchical roles and individual privileges within a role, but also includes system to system interactions
Authentication	A set of controls that are used to verify the identity of a user, or other entity, interacting with the software.
Availability	A measure of a system's accessibility and usability.
Canonicalize	To reduce various encodings and representations of data to a single simple form.
Communication Security	A set of controls that help ensure the software handles the sending and receiving of information in a secure manner.
Confidentiality	To ensure that information is disclosed only to authorized parties.
Contextual Output Encoding	Encoding output data based on how it will be Used by the application. The specific methods vary depending on the way the output data is used. If the data is to be included in the response to the client, account for inclusion scenarios like : the body of an HTML document, an HTML attribute, within JavaScript, within a CSS or in a URL. You must also account for other use cases like SQL queries, XML and LDAP.
Cross Site Request Forgery	An external website or application forces a client to make an unintended request to another application that the client has an active session with. Applications are vulnerable when they use known, or predictable, URLs and parameters; and when the browser automatically transmits all required session information with each request to the vulnerable application. (This is one of the only attacks specifically discussed in this document and is only included because the associated vulnerability is very common and poorly understood.)
Cryptographic Practices	A set of controls that ensure cryptographic operations within the application are handled securely.
Data Protection	A set of controls that help ensure the software handles the storing of information in a secure manner.
Database Security	A set of controls that ensure that software interacts with a database in a secure manner and that the database is configured securely.

Error Handling and Logging	A set of practices that ensure the application handles errors safely and conducts proper event logging.
Exploit	To take advantage of a vulnerability. Typically this is an intentional action designed to compromise the software's security controls by leveraging a vulnerability.
File Management	A set of controls that cover the interaction between the code and other system files.
General Coding Practices	A set of controls that cover coding practices that do not fit easily into other categories.
Hazardous Character	Any character or encoded representation of a character that can affect the intended operation of the application or associated system by being interpreted to have a special meaning, outside the intended use of the character. These characters may be used to ▪ Altering the structure of existing code or statements ▪ Inserting new unintended code ▪ Altering paths ▪ Causing unexpected outcomes from program functions or routines ▪ Causing error conditions ▪ Having any of the above effects on downstream applications or systems
HTML Entity Encode	The process of replacing certain ASCII characters with their HTML entity equivalents. For example, encoding would replace the less than character "<" with the HTML equivalent "<". HTML entities are 'inert' in most interpreters, especially browsers, which can mitigate certain client side attacks.
Impact	A measure of the negative effect to the business that results from the occurrence of an undesired event; what would be the result of a vulnerability being exploited.
Input Validation	A set of controls that verify the properties of all input data matches what is expected by the application including types, lengths, ranges, acceptable character sets and does not include known hazardous characters.
Integrity	The assurance that information is accurate, complete and valid, and has not been altered by an unauthorized action.

Log Event Data	This should include the following 1. Time stamp from a trusted system component 2. Severity rating for each event 3. Tagging of security relevant events, if they are mixed with other log entries 4. Identity of the account/user that caused the event 5. Source IP address associated with the request 6. Event outcome (success or failure) 7. Description of the event
Memory Management	A set of controls that address memory and buffer usage.
Mitigate	Steps taken to reduce the severity of a vulnerability. These can include removing a vulnerability, making a vulnerability more difficult to exploit, or reducing the negative impact of a successful exploitation.
Multi-Factor Authentication	An authentication process that requires the user to produce multiple distinct types of credentials. Typically this is based on something they have (e.g., smartcard), something they know (e.g., a pin), or something they are (e.g., data from a biometric reader).
Output Encoding	A set of controls addressing the use of encoding to ensure data output by the application is safe.
Parameterized Queries (prepared statements)	Keeps the query and data separate through the use of placeholders. The query structure is defined with place holders, the SQL statement is sent to the database and prepared, and then the prepared statement is combined with the parameter values. This prevents the query from being altered, because the parameter values are combined with the compiled statement, not a SQL string.
Personal Data	Personal data means any information relating to an identified or identifiable individual; an identifiable person is one who can be identified, directly or indirectly, in particular by reference to an identification number (e.g. social security number) or one or more factors specific to his physical, physiological, mental, economic, cultural or social identity (e.g. name and first name, date of birth, biometrics data, fingerprints, DNA…)
Sanitize Data	The process of making potentially harmful data safe through the use of data removal, replacement, encoding or escaping of the characters.
Security Controls	An action that mitigates a potential vulnerability and helps ensure that the software behaves only in the expected manner.
Security Requirements	A set of design and functional requirements that help ensure the software is built and deployed in a secure manner.
Sequential Authentication	When authentication data is requested on successive pages rather than being requested all at once on a single page.

Session Management	A set of controls that help ensure web applications handle HTTP sessions in a secure manner.
State Data	When data or parameters are used, by the application or server, to emulate a persistent connection or track a client's status across a multi-request process or transaction.
System	A generic term covering the operating systems, web server, application frameworks and related infrastructure.
System Configuration	A set of controls that help ensure the infrastructure components supporting the software are deployed securely.
Threat Agent	Any entity which may have a negative impact on the system. This may be a malicious user who wants to compromise the system's security controls; however, it could also be an accidental misuse of the system or a more physical threat like fire or flood.
Trust Boundaries	Typically a trust boundary constitutes the components of the system under your direct control. All connections and data from systems outside of your direct control, including all clients and systems managed by other parties, should be consider untrusted and be validated at the boundary, before allowing further system interaction.
Vulnerability	A weakness that makes the system susceptible to attack or damage.

5. Secure Coding Practice Requirements

5.1. Input Validation

5.1.1. Coding Best Practices

- All data from untrusted sources *(e.g., Databases, file streams, etc.)* shall be validated.
- A centralized input validation routine for the application shall be implemented.
- A **secure encoding library** shall be used. Examples:
  - .NET Microsoft AntiXSS Library
  - Java OWASP ESAPI
- For all sources of input **a proper character sets**, such as **UTF-8**, shall be specified.
- Data shall be encoded to a common character set before validating (Canonicalize).

```
// String s may be user controllable
// \uFE64 is normalized to < and \uFE65 is normalized to > using the NFKC normalization
form

String s = "\uFE64" + "script" + "\uFE65";

// Canonicalize
s = Normalizer.normalize(s, Form.NFKC);

// Validate
Pattern pattern = Pattern.compile("[<>]");
Matcher matcher = pattern.matcher(s);
if (matcher.find()) {
  // Found blacklisted tag
  throw new IllegalStateException();
} else {
  // ...
}
```

Code snippet 1 : Java canonicalization example

- **All client provided data shall be validated** before processing, including all parameters, URLs and HTTP header content (e.g. Cookie names and values). Be sure to include automated post backs from JavaScript, Flash or other embedded code.
- **Data from redirects shall be validated** (an attacker may submit malicious content directly to the target of the redirect, thus circumventing application logic and any validation performed before the redirect).
- **Data range and data length shall be validated for expected data types.**

```
public class Secu {
    private Pattern pattern;
    private Matcher matcher;
    // Pattern
    private static final String NUM_PATTERN =
 "^[12][- ]?([0-9]{2}[- ]?)(0[1-9]|1[012][- ]?)(2[AB]|[0-9]{2}[- ]?)([0-9]{3}[- ]?)([0-
9]{3}[- ]?)([0-9]{2})$";
     public Secu() {
        pattern = Pattern.compile(NUM_PATTERN);
    }
    // Length validation
    public boolean validate(final String secuNum) {
       if (secuNum.length() != 15 && secuNum.length() != 21  ) {
            return false;
        }
        matcher = pattern.matcher(secuNum);
   // Validation against pattern
        if (matcher.matches()) {
            return true;
        }
        return false;
    }
}
```

Code snippet 2: Java french social security number validation example

```
public bool isSecuNumber(String secuNumber)
 {
      // Pattern
            Regex myRegex = new Regex(
"^[12][- ]?([0-9]{2}[- ]?)(0[1-9]|1[012][- ]?)(2[AB]|[0-9]{2}[- ]?)([0-9]{3}[- ]?)([0-
9]{3}[- ]?)([0-9]{2})$");
            if (secuNumber == null) {
                return false;
            }
      // Length validation
            if ((secuNumber.Length !=15 && secuNumber.Length != 21)) {
                return false;
            }
            // Validation against pattern
            if (myRegex.IsMatch(secuNumber)) {
                return true;
            }
            else {

                return false;
            }
 }
```

Code snippet 3: .Net french social security number validation example

- **All input against a "white" list of allowed characters shall be validated.**
- If any potentially Hazardous Character must be allowed as input, additional controls like output encoding, secure task specific APIs and accounting for the utilization of that data throughout the application shall be implemented. Examples of common hazardous characters include: < > " ' % ( ) & + \ \' \"
- If your standard validation routine cannot address the following inputs, then they shall be checked:
    - Check for null bytes (%00);
    - Check for “dot-dot-slash" (../ or ..\) path alterations characters. In cases where UTF-8 extended character set encoding is supported, address alternate representation like: %c0%ae%c0%ae/ (Use canonicalization to address double encoding or other forms of obfuscation attacks);
    - Check for new line characters (%0d, %0a, \r, \n).

5.2. Output Encoding

5.2.1. Coding best practice

- **All characters** shall be encoded unless they are known to be safe for the intended interpreter.
- All output data returned to the client that originated outside the application's **trust boundaries** shall be **contextually encoded**. HTML Entity Encode is one example, but does not work in all cases. For more details refer to Output Encoding Rules Summary in OWASP XSS cheat sheet.
- All data output of un-trusted data to queries for SQL, XML, LDAP and operating system commands shall be contextually sanitized.
- A standard, tested routine for each type of outbound encoding shall be used.

5.3. Authentication and Password Management

5.3.1. Application logic

- **User authentication shall be performed against a corporate centralized directory** *(PassAXA, Active Directory)*. If not possible, rely on a centralized directory validated by the Central Security Team.
- Authentication mode and user profile management **shall be reviewed and validated by OpCo security team** before delivering the building permit.
- **Authentication shall be required for all pages and resources**, except those specifically intended to be public.
- Multi-factor authentication (MFA) shall be used **for privileged accounts** (see the User IS Policy regarding password settings below)
- **Users shall be re-authenticated** prior to performing critical operations, e.g. : payment transaction, etc.

5.3.2. Local authentication

- Authentication data shall be validated only on completion of all data input, especially for sequential authentication implementations.

  For example: Instead of "Invalid username" or "Invalid password", just use "Invalid username and/or password" for both. Error responses must be truly identical in both display and source code. Authentication failure responses should not indicate which part of the authentication data was incorrect

- **Password entry shall be obscured on the user's screen.** On web forms use the input type "password".
- **Authentication credentials** for accessing services external to the application shall be encrypted and stored in a protected location on a trusted system.

  For the reader: The source code is NOT a secure location!

- To transmit authentication credentials, only HTTP POST shall be used.
- If using **third party code** for authentication, **code shall be inspect** to ensure it is not affected by any malicious code.
- **Segregate authentication logic /process from the resource being requested** and use redirection to and from the centralized authentication control.
- If your application manages a credential store, it shall ensure that **only cryptographically strong one-way salted hashes of passwords are stored** and that the table/file that stores the passwords and keys is write-able only by the application. A unique salt per entry shall be used. An **algorithm stronger or equivalent to SHA256** shall be used as hashing functions.

  In particular, MD5 or SHA1 algorithms shall not be used for hashing functions

- **Non-temporary passwords shall be sent over an encrypted connection or as encrypted data**, e.g. in an encrypted email. Temporary passwords associated with email resets may be an exception.

- **The following password policy shall be enforced:**

	user types (all systems and endpoint including externally hosted)				Mobile devices	
Password Rules	non-privileged	privileged (1)	service	support & Fire-ID	pre boot authentication	handheld (3)
maximum password age (days)	60	60	365 (if controlled by a tool)	90 (2)	unlimited	unlimited
minimum password age (days)	1	1	NA	1	NA	NA
minimum password length (characters)	8	15	as long as system allows	15	4	6
reuse of n most recently used passwords (cou	13	13	13	13	NA	NA
Account Lockout Threshold (# invalid attempts	5	5	0	5	10	10
Account Lockout Duration (minutes)	30	30	0	30	until admin intervention	data wiping
Reset account lockout threshold (minutes)	30	30	0	30	NA	NA
inactivity lock (minutes)	15	15	NA	15	NA	15
	All settings mandatory where technically possible, configuration baseline compliance checks will raise non compliance.					
	(1) multi factor preferred, where passwords are used, they need to be distinct on each system and not reused.					
	(2) Password of Fire-ID to be changed each time it is used.					
	(3) For handheld devices addtion security reuirements are llisted in 11.7.1.b					

5.3.3. Password reset and temporary passwords

- Password reset and changing operations require the same level of controls as account creation and authentication.
- Password reset questions should be avoided. In case of exceptions, then Password reset questions should support sufficiently random answers.

  For example, "favorite book" is a bad question because “The Bible” is a very common answer

- If using email based resets, email shall only be sent to a pre-registered address with a temporary link/password.
- Temporary passwords and links shall have a short expiration time.
- Changing of temporary passwords shall be enforced on the next use.

5.4. Session Management

5.4.1. Session identifiers

- Session identifiers shall not be exposed in URLs, error messages or logs. **Session identifiers shall only be located in the HTTP cookie header**.

  For example: do not pass session identifiers as GET parameters

- Domain and path for cookies containing authenticated session identifiers shall be set to an appropriately restricted value for the site.
- The server or the related framework's session management controls shall be used. The application shall only recognize the defined session identifiers as valid.
- Server side session data shall be protected from unauthorized access, by other users of the server, by implementing appropriate access controls on the server.

5.4.2. Session termination

- Logout functionality shall:
  - Fully terminate the associated session or connection;
  - Be available from all pages protected by authorization.
- **Persistent logins** are forbidden and **periodic session terminations shall be enforced**, even when the session is active. Especially for applications supporting rich network connections or connecting to critical systems.
- **A user's session shall be automatically locked after a period of inactivity** no longer than 15 minutes and a password required for unlock.
- Regardless of user activity, user session shall terminate after a specific delay.
- Termination times shall support business requirements and the user shall receive sufficient notification to mitigate negative impacts.
- **Concurrent logins** with the same user ID are forbidden.

5.4.3. Re-authentication

- **A new session identifier shall be generated on any re-authentication.**
- **When new session identifier are generated, old session identifier shall be disabled**. This can mitigate certain session hijacking scenarios where the original identifier was compromised.
- Standard session management for sensitive server-side operations, like account management, shall be replaced by **per-session strong random tokens or parameters**. This method can be used to prevent Cross Site Request ForgeryErreur ! Source du renvoi introuvable. attacks.
- Standard session management for highly sensitive or critical operations shall be replaced by per-request, as opposed to per-session, strong random tokens or parameters.

5.4.4. HTTP Settings

- If the connection security changes from HTTP to HTTPS, as can occur during authentication, a new session identifier shall be generated. **Within an application, it is recommended to consistently use HTTPS rather than switching between HTTP to HTTPS.**

  Set the "secure" attribute for cookies transmitted over a TLS connection.

  Set cookies with the “HttpOnly” attribute, unless you specifically require client-side scripts within your application to read or set a cookie's value.

5.4.5. Captcha

- To prevent brute force attack, **a captcha mechanism shall be implemented after a limited number of failed authentication attempts.**

5.5. Access Control

5.5.1. Coding best practice

- **A single site-wide component shall be used to check access authorization.** This includes libraries that call external authorization services.

  Use the "referer" header as a supplemental check only, it should never be the sole authorization check, as it is can be spoofed

- **Authorization controls shall be enforced on every request.**

  This includes requests made by server side scripts, "includes" and requests from rich client-side technologies like AJAX and Flash

- Access shall be restricted to only authorized users for:
  - Files or other resources, including those outside the application's direct control
  - Protected URLs
  - Protected functions
  - Direct object references
  - Services
  - Application data
  - User and data attributes and policy information used by access controls
  - Security-relevant configuration information
- If State data must be stored on the client, encryption and integrity checking on the server side shall be used to catch state tampering.

5.5.2. Application logic

- Access controls shall fail securely.
- All access shall be denied if the application cannot access its security configuration information.
- Privileged logic shall be segregated from other application code.
- Service accounts or accounts supporting connections to or from external systems shall have the least privilege possible.
- Account auditing shall be implemented and the disabling of unused accounts shall be enforced (e.g., After no more than 30 days from the expiration of an account’s password.).

5.6. Cryptographic Practices

5.6.1. Cryptographic algorithms

- For the use of specific cryptographic mechanism refer to Group Security Instructions related to Encryption
- All random numbers, random file names, random GUIDs, session identifiers and random strings shall be generated using the cryptographic module's approved random number generator when these random values are intended to be un-guessable.

```
import java.util.Random;
// ...

Random number = new Random(123L);
//...
for (int i = 0; i < 20; i++) {
  // Generate another random integer in the range [0, 20]
  int n = number.nextInt(21);
  System.out.println(n);
}
```

Code snippet 4: Use of unsecure predictable random number generator class

```
import java.security.SecureRandom;
import java.security.NoSuchAlgorithmException;
// ...

public static void main (String args[]) {
   try {
     SecureRandom number = SecureRandom.getInstance("SHA1PRNG");
     // Generate 20 integers 0..20
     for (int i = 0; i < 20; i++) {
       System.out.println(number.nextInt(21));
     }
   } catch (NoSuchAlgorithmException nsae) {
     // Forward to handler
   }
}
```

Code snippet 5: Use of compliant unpredictable random number generator class

5.6.2. Application logic

- **Master secrets shall be protected** from unauthorized access
- Cryptographic modules shall fail securely. In case of error, access shall be denied and no information shall be disclosed.

5.7. Error Handling and Logging

5.7.1. Error handling

- **Sensitive information shall not be disclosed in error responses**, including system details, session identifiers or account information.
- Error handlers that do not display debugging or stack trace information shall be used.
- **Generic error messages shall be implemented and custom error pages shall be used**. The application shall handle application errors and not rely on the server configuration.
- Error handling logic associated with security controls shall **deny access by default.**
- **Allocated memory** when error conditions occur shall be properly freed.

5.7.2. Logging

- **Master routine for all logging operations shall be used.**
- **Logging controls shall support both success and failure** of specified security events.
- **Log entries** that include un-trusted data shall be blocked **from being executed as code in the intended log viewing interface or software.**
- **Sensitive information shall not be stored in logs**, including unnecessary system details, session identifiers or passwords.
- **All events data shall be timestamped** based on a reliable synchronized **UTC** time source.
- All following events shall be logged and make sure logs are collected and centralized in the corporate SIEM solution:
    - Input validation failures
    - Authentication attempts, especially failures
    - Access control failures
    - Apparent tampering events, including unexpected changes to state data
    - Attempts to connect with invalid or expired session tokens
    - System exceptions
    - Administrative functions, including changes to the security configuration settings
    - Backend TLS connection failures
    - Cryptographic module failures
- The logging mechanisms and collected event data shall be protected against tampering in transit, and unauthorized access, modification and deletion once stored.

For the reader: The IS requirements in terms of logging activity and log retention period are defined in the Security Log Management Policy.

5.7.3. Operational controls

- Access to logs shall be restricted to only authorized individuals.
- A mechanism shall be available to conduct log analysis.
- Cryptographic hash function shall be used to validate log entry integrity.

5.8. Data Protection

5.8.1. Coding best practice

- **Highly sensitive stored information shall be encrypted**, like authentication verification data, even on the server side. Always use well vetted algorithms, as defined in the "Cryptographic Practices" section for additional guidance.
- Appropriate access controls shall be implemented for sensitive data stored on the server.

  This includes cached data, temporary files and data that should be accessible only by specific system users

- **All cached or temporary copies of sensitive data stored on the server shall be protected** from unauthorized access and those temporary working files shall be purged a soon as they are no longer required.
- Passwords, connection strings or other sensitive information shall not be stored in clear text or in any non-cryptographically secure manner on the client side.

  This includes embedding in insecure formats like: MS viewstate, Adobe flash or compiled code.

- The application shall support the removal of sensitive data when that data is no longer required.

  For example, personal information or certain financial data may require deletion after specific period of time.

- Auto complete features on forms expected to contain sensitive information, including authentication, shall be deactivated.
- Server-side source-code shall be protected from being downloaded by a user.

5.8.2. Personal Data specific requirements

- Environments that are not production environments shall not contain personal data , but anonymized or pseudonymized data.
- Application shall support the removal of personal data when data is no longer required, or on request.

  For example, personal information may require deletion after specific period of time, or when a user requests the deletion of its data.

- Application shall support the removal of personal data from any processing when it is no longer required, or on request, without needing data deletion.

  For example, personal information may require deletion after specific period of time, or when a user requests the deletion of its data or when the user requests that its data stops being processed

- Application shall support the export of personal data in a structured format on request, per person.
- All personal data export or extraction outside of its system shall be tracked and documented.

For example, personal information could be exported to a csv file somewhere on the filesystem, or some personal data could appear in a log file for debugging purpose.

These locations shall be documented as containing personal data and monitored for unauthorized access.

5.8.3. HTTP Specifics

- Sensitive information shall not be included in HTTP GET request parameters.
- **Client side caching on pages containing sensitive information shall be disabled.** Cache-Control: no-store, may be used in conjunction with the HTTP header control "Pragma: no-cache", which is less effective, but is HTTP/1.0 backward compatible.

5.9. Communication Security

5.9.1. SSL/TLS

- By default, TLS encryption shall be enforced on all network data flow. In particular, all web application shall use HTTPS protocol only.
- Transmission of sensitive information shall be encrypted. This should include TLS for protecting the connection and can be completed by discrete encryption of sensitive files or non-HTTP based connections.

  For TLS, ensure that version is at least TLS 1.2.

  For additional detail on the use of specific cryptographic mechanism refer to Group Security Instructions related to Encryption.
- Failed TLS connections shall not fall back to an insecure connection.
- TLS shall be used for all content requiring authenticated access and for all other sensitive information.
- TLS shall be used for connections to external systems that involve sensitive information or functions.

5.9.2. Certificates and CRL

- TLS certificates shall be valid and have the correct domain name, not be expired, and be installed with intermediate certificates when required.
- Certificates validation shall include controls of non-revocation (by looking at the CRL), non-expiry and that the CN (Common Name) matches the expected subject. If check fails, the user shall be provided with adequate information about the nature of the problem and how to proceed.
- CRL (Certificate Revocation List) signature shall be controlled before trusting it.
- Chain of trust of a certificate shall be rooted in a reputable trusted entity (a trusted Certificate Authority).
- Certificate status for each sensitive transaction (e.g. payment transactions) shall be verified and previously cached status shall not be used.

5.10. Database Security

5.10.1. Coding best practice

- **Strongly typed Parameterized requests shall be used.**

```
try {
      String pwd = hashPassword(password);
      String sqlString = "select * from db_user where username=?  and password=?";
      PreparedStatement stmt = connection.prepareStatement(sqlString);
      stmt.setString(1, username);
      stmt.setString(2, pwd);
      ResultSet rs = stmt.executeQuery();
      if (!rs.next()) {
        throw new SecurityException("User name or password incorrect");
     }
```

Code snippet 6: Java prepared statement example

- **Stored procedures shall be used** to abstract data access and allow for the removal of permissions to the base tables in the database
- **Input validation and output encoding shall be implemented,** including meta characters. If these fail, do not run the database command
- **Connection strings shall not be hard coded within the application**. **Connection strings** shall be stored in a separate configuration file on a trusted system and they **shall be encrypted**. Access to this configuration file shall be restricted only to the account managing the database system.
- Connections shall be closed as soon as possible.

```
public final Connection getConnection() throws SQLException {
  // Hardcoded username and password
  return DriverManager.getConnection(
      "jdbc:mysql://localhost/dbName",
      "username", "password");
}
```

Code snippet 7: Hardcoded username and password are forbidden

```
public final Connection getConnection() throws SQLException {
  String username;
  String password;
  // Username and password are read at runtime from a secure config file
  return DriverManager.getConnection(
      "jdbc:mysql://localhost/dbName", username, password);
}
```

Code snippet 8: example of how to get username and password

5.10.2. Application logic

- The **application shall use the lowest possible level of privilege** when accessing the database. These privileges shall be documented.
- The **application shall connect to the database with different credentials** for every trust distinction (e.g., user, read-only user, guest, administrators).
- Credentials shall be secured for database access.

5.11. File Management

5.11.1. Coding best practice

- **User supplied data shall not be passed directly** to any dynamic include function
- **User supplied data shall not be passed into a dynamic redirect**. If this must be allowed, then the redirect shall accept only validated, relative path URLs.
- **Directory or file paths shall not be passed**, index values mapped to pre-defined list of paths shall be used. Types of files that can be uploaded shall be limited to the ones needed for business purposes
    - The uploading of files that may be interpreted by the web server, shall be forbidden or restricted to business needs.
- **Uploaded files type shall be controlled by checking file headers.** Checking for file type by extension alone is not sufficient.
- When referencing existing files, a white list of allowed file names and types shall be implemented. The value of the parameter being passed shall be validated. If it does not match one of the expected values, it shall be rejected or a hard coded default file value shall be used.
- The absolute file path shall never be sent to the client.

5.11.2. Application logic

- Authentication shall be required before allowing a file to be uploaded.
- **Files shall not be saved in the same web context as the application.** Files shall either go to the content server or in the database.

5.12. Memory Management

5.12.1. Coding best practice

- Known vulnerable functions *(e.g., printf, strcat, strcpy etc.)* shall not be used.
- Buffer overflow safe libraries *(e.g., The Better String Library, Vstr, Erwin)* shall be used.
- Buffer boundaries shall be checked if calling the function in a loop, including the capability of writing past the allocated space.
- Input strings shall be truncated to a reasonable length before passing them to the copy and concatenation functions.
- Non-executable stacks shall be used when available.

  For Microsoft Windows, enable Data Execution Prevention for all processes.

5.12.2. Memory deallocation

- Release of resources shall be specified, **do not rely on garbage collection**. *(e.g., connection objects, file handles, etc.)*.
- **Allocated memory shall be freed** upon the completion of functions and at all exit points.

5.13. General Coding Practices

5.13.1. Coding best practice

- **Tested and approved managed code** shall be used rather than creating new unmanaged code for common tasks.
- Specific built-in APIs shall be used to conduct operating system tasks. **The application shall not issue commands directly to the Operating System**, especially through the use of application initiated command shells.
- Checksums or hashes shall be used to **verify the integrity of interpreted code**, libraries, executables, and configuration files.
- Locking shall be implemented to prevent multiple simultaneous requests or when required a synchronization mechanism shall be used to prevent race conditions.
- All variables and other data stores shall be initialized, either during declaration or just before the first usage.
- If the application must run with elevated privileges, privileges shall be raised as late as possible, and shall be dropped as soon as possible.
- Users shall not be able to generate new code or altering existing code.
- Updating shall be secured. If the application uses automatic updates, then cryptographic signatures for the code shall be used and download clients shall verify those signatures. Code transfer from the host server shall be encrypted.
- Code shall be tested by building test suites composed of unit tests and integration tests.
- All success and failure conditions shall be tested.

5.13.2. External libraries

- **External libraries shall be hosted** in AXA trusted servers *(e.g. javascript files shall not be included from the editor's website)*
- An inventory of all trusted libraries shall be validated and maintained on a regular basis.
- Trusted libraries shall be downloaded from AXA repository instead of the internet.
- All secondary applications, third party code and libraries shall be reviewed to determine business necessity and validate safe functionality, as these can introduce new vulnerabilities.

5.13.3. Coding development environments best practice

- Compilers that check, that the stack has not been altered when a function returns, shall be used, i.e. provide developers with compilers that implement "Buffer overflow protection" thanks to compiler protection technologies *(e.g. GCC with StackShield / StackGuard / ProPolice flags, Microsoft Visual Studio with "/GS " parameter…)*.

6. Infrastructure Best Practices Checklist

6.1. System Configuration

- **Test code** or any functionality not intended for production, **shall be removed prior to deployment in production.**
- **Development environments shall be isolated from the production network** and only authorized development and test groups shall be granted access. Development environments are often configured less securely than production environments and attackers may use this difference to discover shared weaknesses or as an avenue for exploitation.
- All systems and web servers deployed shall be compliant with the relevant hardening procedures or hardened templates. The hardening procedures and/or hardened templates shall be validate by the security team prior to being used.

6.1.1. Web Server Configuration

- Disclosure of the directory structure in the robots.txt file shall be prevented by placing directories not intended for public indexing into an isolated parent directory. Then the entire parent directory shall be disallowed in the robots.txt file rather than disallowing each individual directory.
- Directory listings shall be turned off.
- Supported HTTP methods (e.g. GET, POST) shall be defined and whether it will be handled differently in different pages in the application.
- Unnecessary HTTP methods shall be disabled, such as WebDAV extensions. If an extended HTTP method that supports file handling is required, Use a well-vetted authentication mechanism.
- If the web server handles both HTTP 1.0 and 1.1, both shall have the same configuration or ensure at least the difference that may exist shall be known and accepted *(e.g. handling of extended HTTP methods)*.
- Unnecessary information from HTTP response headers related to the OS, web-server version and application frameworks shall be removed.

6.1.2. System Hardening

- Web server, process and service accounts shall be restricted to the least privileges possible.
- All unnecessary functionality and files shall be removed.

6.1.3. Operational security

- An asset management system shall be implemented, including system components and software.
- A software change control system shall be implemented to manage and record changes to the code both in development and production.
- Servers, frameworks and system components shall have all patches issued for the version in use.

6.1.4. Logging

- Access to logs shall be restricted to only authorized individuals.
- A mechanism shall exist to conduct log analysis.
- Log entire integrity shall be validated with hash functions.

For the reader: The IS requirements in terms of logging activity and log retention period are defined in the Security Log Management Policy.

6.2. Database Security

6.2.1. Database management system configuration

- All default database administrative passwords shall be removed or changed. Strong passwords/phrases or multi-factor authentication shall be used.
- All unnecessary database functionality shall be turned off *(e.g., unnecessary stored procedures or services, utility packages, install only the minimum set of features and options required (surface area reduction)).*
- Unnecessary default vendor content shall be removed (e.g., sample schemas).
- Any default accounts that are not required to support business requirements shall be removed / disabled.

6.3. File Management

6.3.1. Server configuration

- Execution privileges shall be turned off on file upload directories
- Safe uploading shall be implemented in UNIX by mounting the targeted file directory as a logical drive using the associated path or the chrooted environment.
- Application files and resources shall be read-only.
- All uploaded files shall be scanned for viruses and malware.

6.4. Firewalling

6.4.1. Server configuration

- Web Application Firewalls shall be implemented in order to detect, block and alert Web attacks such as XSS.

7. Mobile Code - Secure Coding guidelines

7.1. Presentation and protection against Mobile Applications vulnerabilities (OWASP)

7.1.1. Improper Platform Usage

- Authentication for every sensitive transaction / transaction shall be requested
- Security features built into the Development Framework (anti-CSRF token, XSS blocking, etc.) should be enabled
- Coherent entitlement management ("does the identified individual have the right to access the resource?") should be performed,
- Injection attempts through the validation of entries, canonization, encoding outputs, etc. (find out about injections, in the top 10 OWASP Web) should be blocked
- APIs (services, Frameworks, configuration, etc.) as well as the platform (server) components should be kept secure and up-to-date
- Residual mobile-specific risks in existing architectures should be Clearly identified
- Recommendations for Web Services should be applied

7.1.2. Insecure Data storage

- The nature of the data stored at the client level (Mobile device) should be identified
- Data should be classified in terms of sensitivity according to the CID criteria (Confidentiality, Integrity, Availability)
- Data should be stored on the mobile only if necessary,
- Sensitive data (login / password pair for example) should never be stored on the phone, prefer storage at the server level.
- Sensitive data should never be stored at unsecured media (SD cards, ...)
- "SQLite Cipher" should be used to encrypt BDDs on mobile
- Data should be erased from the memory after application shutdown
- Sensitive information should not be stored in Log or cache

Example of encryption libraries:

- Spongy Castle for Android (enhanced version of the standard Bouncy Castle encryption Android library)
- Common Crypto and Keychain Services for iOS (Encryption APIs)
- System.Security.Cryptography for Windows Phone (Bouncy Castle is available in C #)
- Storage of cached data (e.g. Session cookies and authentication tokens) should be limited in time
- The application should only require access to a minimum of functions / data from the device during installation.

7.1.3. Insecure communication

- Sensitive data should be encrypted as soon as it leaves the device
- Strong encryption algorithm must be used.
- All exchanges must be secured with SSL (not just the authentication phase)
- All types of channels: DSL, Wireless, 3G / UMTS, NFC, etc. should be secured

- Sensitive data should not be transmitted through alternative channels: SMS, MMS or via notifications
- Used certificates should be signed by a trusted Certificate Authority
- Self-signed certificates should not be allowed
- Exceptions thrown after a certificate check for SSL should not be thrown
- Do not return to unencrypted mode after an error
- When an invalid certificate is detected, the user should be alerted

7.1.4. Insecure Authentication

- All exchanges during which the password is sent must be protected: use SSL / TLS
- For some sensitive actions (transfer for example), the user should be re-authenticated.
- Machine ID or Operator ID should never be used as the sole authentication element, use multi factor authentication (e.g. "user ID + password" + OTP)
- The password must be stored on the server (no mobile storage)
- The application must comply with the AXA group's password security policy
- For password management / protection and cryptography in general, existing solutions (ex: Keychain, ...) should be used rather than implementing a custom mechanism.

For additional detail on Authentication requirements refer to Application IS Policy

7.1.5. Insufficient cryptography

- Encryption keys should not be stored in the source code.
- Encryption of application should be tested and analyzed
- Encoding (such as Base64) encoding is not encryption!
- Encryption keys should be rotated
- standard libraries approved by AXA should be used over unknown/untested libraries
- Standard and robust algorithms (AES, SHA-256, RSA, etc.) should be used and implemented correctly
- Keys for storage and transmission should be encrypted
- Passwords chosen by "human" should not be used, use robust PNRG algorithms instead.
- Low PRNG (ex: rand () in C language, java.util.Random () in java ...) should not be used

For additional detail on the use of specific cryptographic mechanism refer to Group Security Instructions related to Encryption.

7.1.6. Insecure Authorization

- All access to resources hosted at the server level should be secured,
- The role of the user should not be included in the request sent to the server, for example role = admin, role = user

- Upon receipt of each request, verification that the user is authorized to access the requested resource or execute the invoked function must be performed,
- The device identifier should not be used as a session token,
- Ability to quickly revoke the tokens in the event of theft / loss of the device should be secured,
- Means of protection against the theft of sessions (activate "HttpOnly" and "secure" attributes for session cookies must be implemented, only secure connections to communicate with the server should be used.

7.1.7. Code Tampering

Check the build.prop file, if the build tags contain "test-keys" (ro.build.tags = test-keys) then the ROM is not official. This control alone is not a way to overcome the risk of code corruption, but it is part of a defense-in-depth approach.

Check the presence of APKs known to be rooted as for example:

- com.noshufou.android.su
- com.thirdparty.superuser
- eu.chainfire.supersu
- com.koushikdutta.superuser

Check the presence of SU binaries

- /system/bin/su
- /system/xbin/su
- /sbin/su
- /system/su
- /system/bin/.ext/.su

Use the SU command directly and check if the user ID is returned, if the user ID is 0, then the OS is rooted.

8. Referenced IS documents

Document	Type	Details	Location	Version
Technical infrastructure security baseline	AXA Group Instruction	Group Security Information Security Requirements on Technical infrastructure security baseline	ONE	2018
Encryption	AXA Group Instruction	Group Security Information Security Requirements on Encryption	ONE	2018
IS Global Policy	Local Policy	Global IS policy of AXA IT – Application Products Department	ONE APD Security	2018
IS User policy	Local Policy	Security requirements related to Users (access to applications, handling of assets, security usages).	ONE APD Security	2018
IS Application policy	Local Policy	Security requirements related to Applications in project mode and maintenance.	ONE APD Security	2018
SDLC Policy	Local Policy	Mandatory Information Security controls required to be applied throughout the lifecycle of all IT systems / applications development, testing and maintenance	ONE APD Security	2018

		Customization of a IS group ISO template		
IS Network policy	Local Policy	Security requirements related to Networks (segregation).	ONE APD Security	2018
Security Log Management Policy	Local Policy	Security requirements related to security logs management	ONE APD Security	2018
IS Data Classification & Protection Policy	Local Policy	Security requirements related to data classification & protection	ONE APD Security	2018

9. Documentation management process

9.1. Acronyms list

Item	Definition
FQDN	Fully Qualified Domain Name
IS	Information Security
ISMS	Information Security Management system
KPI	Key Performance Indicator
KRI	Key Risk Indicator
LAN	Local Area Network
VPN	Virtual Private Network

9.2. Approval and contacts

Approval

Name	Role	Date	Observation
Sébastien MAHIEUX	Chief Information Security Officer	November 12th , 2018	

Contact

Contact	Role	AXA IT – Application Products Department
Sébastien MAHIEUX	Chief Information Security Officer	
Bachir IBROUCHENE	Security BAU manager	
Elona DERVISHI	Security Program manager	
Mehdi BOUAYAD	Security expert	

9.3. Revision history

Owner	Version	Approved Date	Comment
C.PROUST	0.3	18/10/2016	Added: ▪ §Coding development environments best practice ▪ AND in the annex for OWASP link ▪ §Firewalling for WAF configuration on infrastructure server side ▪ Compliance Cookbooks 2016
R.BEAUFILS	0.4	07/06/2018	Update to new template Added: ▪ Personal data section ▪ Minor edits
A.GABSI	0.5	10/10/2018	Added Mobile secure coding guidelines
Mehdi BOUAYAD	0.6	29/10/2018	Review of document
Yassine BOUSSOUIS	0.7	5/11/2018	Review of document
Bachir IBROUCHENE	0.8	8/11/2018	Review of document

10. Appendices

10.1. Attack prevention

Category	Link
OWASP Proactive Controls	https://www.owasp.org/index.php/OWASP_Proactive_Controls
SQL Injection Prevention Cheat Sheet	https://www.owasp.org/index.php/Injection_Prevention_Cheat_Sheet https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
Query Parameterization Cheat Sheet	https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet
LDAP Injection Prevention Cheat Sheet	https://www.owasp.org/index.php/LDAP_Injection_Prevention_Cheat_Sheet
XSS (Cross Site Scripting) Prevention Cheat Sheet	https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet
DOM based XSS Prevention Cheat Sheet	https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet
CSRF Prevention Cheat Sheet	https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

10.2. Security Best practices

Category	Link
Cryptographic Storage Cheat Sheet	https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet
Input Validation Cheat Sheet	https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet
Logging Cheat sheet	https://www.owasp.org/index.php/Logging_Cheat_Sheet
Password Storage cheat Sheet	https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet
Forgot Password Cheat Sheet	https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet

Category	Link
Session Management Cheat Sheet	https://www.owasp.org/index.php/Session_Management_Cheat_Sheet
Transaction Authorization Cheat Sheet	https://www.owasp.org/index.php/Transaction_Authorization_Cheat_Sheet
Unvalidated_Redirects_and_Forwards_Cheat_Sheet	https://www.owasp.org/index.php/Unvalidated_Redirects_and_Forwards_Cheat_Sheet
User_Privacy_Protection_Cheat_Sheet	https://www.owasp.org/index.php/User_Privacy_Protection_Cheat_Sheet

10.3. Specific technology documents

Category	Link
.NET OWASP Security Cheat Sheet	https://www.owasp.org/index.php/.NET_Security_Cheat_Sheet
.NET Writing Secure Code	https://msdn.microsoft.com/en-us/security/aa570401.aspx
.NET Security Warnings	https://msdn.microsoft.com/en-us/library/ms182296.aspx
Oracle Java Secure coding Guidelines for Java SE	http://www.oracle.com/technetwork/java/seccodeguide-139067.html
SEI CERT Oracle Coding Standard for Java	https://www.securecoding.cert.org/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java
JAAS Cheat Sheet	https://www.owasp.org/index.php/JAAS_Cheat_Sheet
Ruby On Rails cheat Sheet	https://www.owasp.org/index.php/Ruby_on_Rails_Cheatsheet
REST Security Cheat Sheet	https://www.owasp.org/index.php/REST_Security_Cheat_Sheet
SAML Security Cheat Sheet	https://www.owasp.org/index.php/SAML_Security_Cheat_Sheet
Transport Layer Protection Cheat Sheet	https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet
Web_Service_Security_Cheat_Sheet	https://www.owasp.org/index.php/Web_Service_Security_Cheat_Sheet
Web_Service_Security_Testing_Cheat_Sheet	https://www.owasp.org/index.php/Web_Service_Security_Testing_Cheat_Sheet

10.4. Mobile

Category	Link

IOS_Developer_Cheat_Sheet	https://www.owasp.org/index.php/IOS_Developer_Cheat_Sheet

10.5. Free source code security analysis tools

10.5.1. Multi languages

Category	Link
SonarQube (C, C++, Java, Objective-C)	SonarQube is already in use in Asia by both AXA Asia IT and AXA Group Solutions Asia. The SonarQube security module, though a convenient and easy to use, is a tactical solution and might be replaced in the future http://www.sonarqube.org/
VisualCodeGrepper (C++, C#, VB, PHP, Java and PL/SQL)	http://sourceforge.net/projects/visualcodegrepp/ https://github.com/nccgroup/VCG
Yasca (Java, C/C++, HTML, JavaScript, ASP, ColdFusion, PHP, COBOL, .NET)	www.yasca.org/ https://github.com/scovetta/yasca
RATS (C, C++, Perl, PHP, Python)	https://code.google.com/p/rough-auditing-tool-for-security/ https://security.web.cern.ch/security/recommendations/en/codetools/rats.shtml
JavaScript Scanjs, ruleset for the eslint. tool	https://github.com/mozfreddyb/eslint-config-scanjs http://eslint.org/
JSprime	http://dpnishant.github.io/jsprime/ https://github.com/dpnishant/jsprime
DOM-XSS scanner , a burp suite plugin	https://www.codemagi.com/downloads/dom-xss-scanner-checks https://portswigger.net/burp/
PHP	***RIPS*** http://rips-scanner.sourceforge.net/ ***DevBug*** http://www.devbug.co.uk
Java	***LAPSE+, Eclipse Helios plugin*** http://evalues.es/projects/lapse+/lapse+.html ***Find Security Bugs*** http://h3xstream.github.io/find-sec-bugs/
.NET	***FxCop*** https://msdn.microsoft.com/en-us/library/bb429476.aspx http://www.microsoft.com/en-us/download/details.aspx?id=6544 ***CAT.NET*** http://www.microsoft.com/en-us/download/details.aspx?id=5570
Ruby	***Brakeman*** http://brakemanscanner.org/
C/C++	***Flawfinder*** http://www.dwheeler.com/flawfinder/ ***Cppcheck*** http://cppcheck.sourceforge.net/
Python	***Bandit*** https://wiki.openstack.org/wiki/Security/Projects/Bandit

Category	Link
	https://github.com/openstack/bandit

10.5.2. Coding Development environment

Category	Link
Coding Development environment)	https://www.owasp.org/index.php/Buffer_Overflows