Standard Bank Interface Security Verification Result

October 2017

Detailed Verification Requirements 5

V1: Architecture, design and threat modelling 6

Control objective 6

Requirements 6

References 7

V2: Authentication Verification Requirements 8

Control objective 8

Requirements 8

References 10

V3: Session Management Verification Requirements 12

Control objective 12

Requirements 12

References 13

V4: Access Control Verification Requirements 14

Control objective 14

Requirements 14

References 15

V5: Malicious input handling verification requirements 16

Control objective 16

Requirements 16

References 18

V6: Output encoding / escaping 20

V7: Cryptography at rest verification requirements 21

Control objective 21

Requirements 21

References 22

V8: Error handling and logging verification requirements 23

Control objective 23

Requirements 23

References 24

V9: Data protection verification requirements 25

Control objective 25

Requirements 25

References 27

V10: Communications security verification requirements 28

Control objective 28

Requirements 28

References 29

V11: HTTP security configuration verification requirements 31

Control objective 31

Requirements 31

References 32

V12: Security configuration verification requirements 33

V13: Malicious controls verification requirements 34

Control objective 34

Requirements 34

References 34

V14: Internal security verification requirements 35

V15: Business logic verification requirements 36

Control objective 36

Requirements 36

References 36

V16: Files and resources verification requirements 37

Control objective 37

Requirements 37

References 38

V17: Mobile verification requirements 39

Control objective 39

Requirements 39

References 40

V18: Web services verification requirements 41

Control objective 41

Requirements 41

References 42

V19. Configuration 43

Control objective 43

Requirements 43

References 44

Appendix A: What ever happened to… 45

Appendix B: Glossary 51

Appendix C: References 55

Appendix D: Standards Mappings 56

# Detailed Verification Requirements

V1. Architecture, design and threat modelling

V2. Authentication

V3. Session management

V4. Access control

V5. Malicious input handling

V7. Cryptography at rest

V8. Error handling and logging

V9. Data protection

V10. Communications

V11. HTTP security configuration

V13. Malicious controls

V15. Business logic

V16. File and resources

V17. Mobile

V18. Web services (NEW for 3.0)

V19. Configuration (NEW for 3.0)

# V1: Architecture, design and threat modelling

## Control objective

Ensure that a verified application satisfies the architecture and design is in place, in use, and effective.

## Requirements

1. Verify that all application components are identified and are known to be needed **PASSED**

**Standard Bank Interface** has list of components as following:

|  |  |  |
| --- | --- | --- |
| **No.** | **Component** | **Description** |
| 1 | Request Receiver | Process all payment requests, check authentication then forwards to Request Processor or queue. |
| 2 | Request Processor | Process payment logic of the requests. |
| 3 | Tokenization Service | Process all tokenization logics. |
| 4 | Bank Adapter | Interface that provides all functions need to integrate to bank services. |
| 5 | Operation Tool | User interface provided to operate SBI system. |
| 6 | Configuration Service | Provide APIs to get system configuration. |
| 7 | Measurement Agent | Measure servers’ performance and update to Request Receiver. |
| 8 | Queue Service | Handle asynchronous and configuration update requests. |
| 9 | Analytics | Help to analyse users’ behaviour and SBI system’s performance. |

1. Verify that all components, such as libraries, modules, and external systems, that are not part of the application but that the application relies on to operate are identified **PASSED**

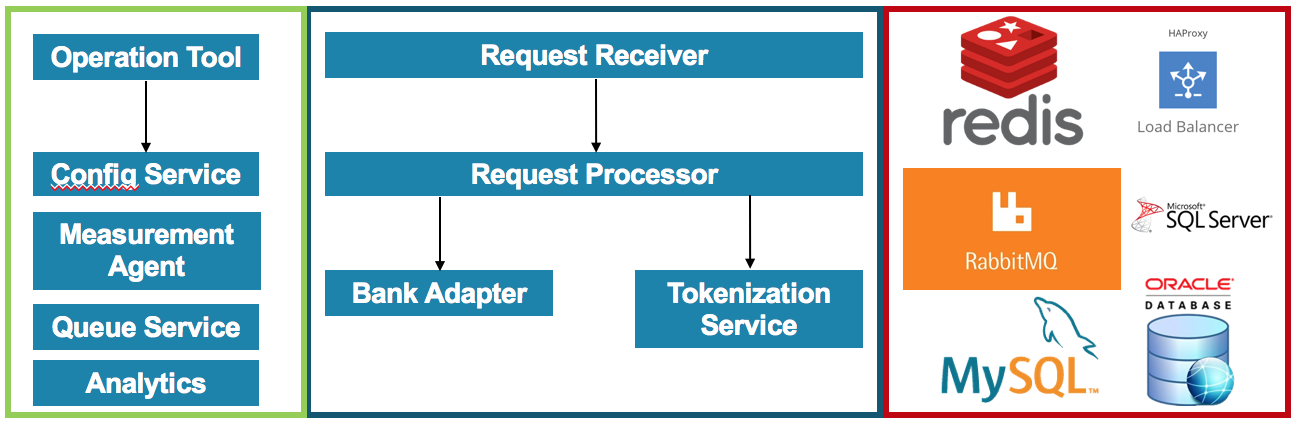
**Standard Bank Interface** requires to integrate with bank components as following:

|  |  |  |
| --- | --- | --- |
| **No.** | **Component** | **Description** |
| 1 | Bank Services | APIs provided by bank to integrate with core bank/core card. |
| 2 | Core Card | Card management system of bank. |
| 3 | Core Bank | Provide functions to access their bank accounts and perform basic transactions from any of the member branch offices. |

**Standard Bank Interface** requires a list of libraries as following: **NhanNV2**

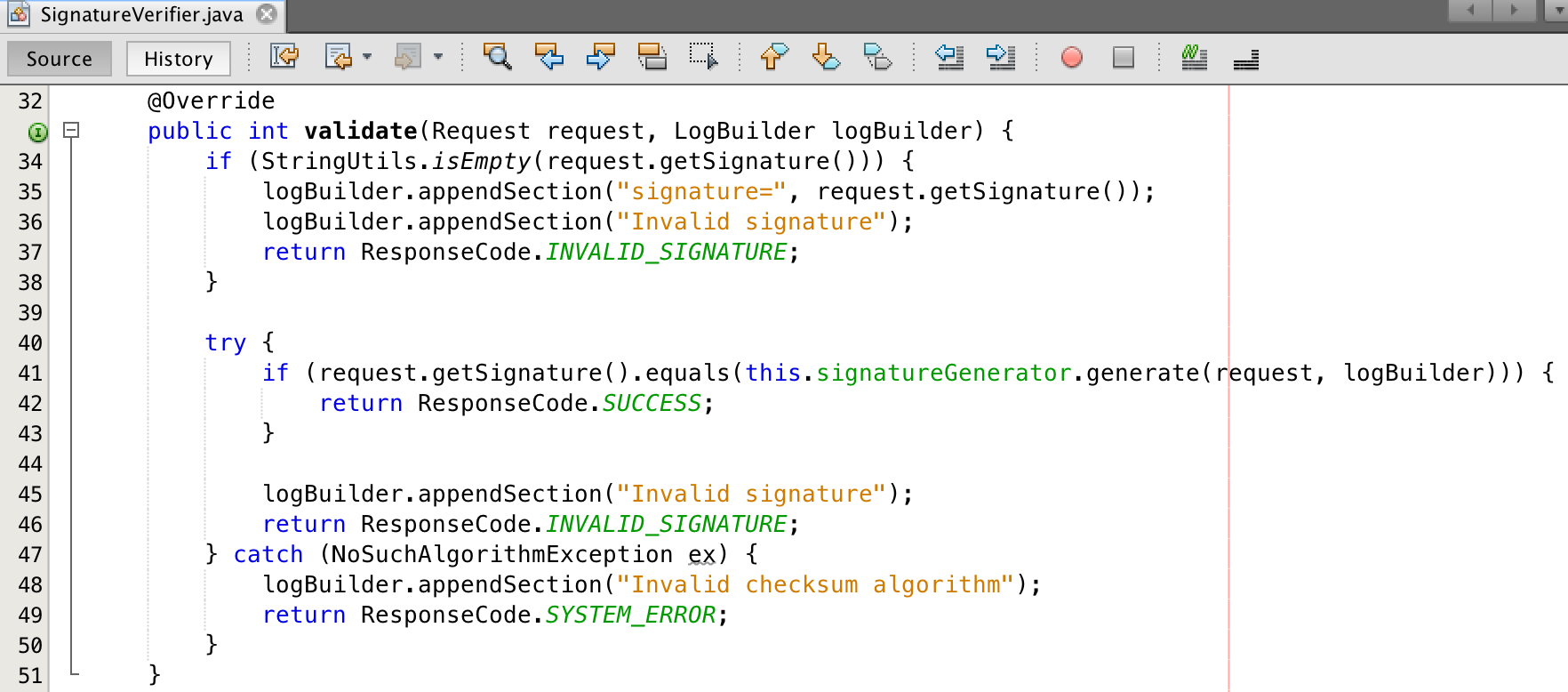
|  |  |  |
| --- | --- | --- |
| **No.** | **Library** | **Description** |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |

1. Verify that a high-level architecture for the application has been defined **PASSED**

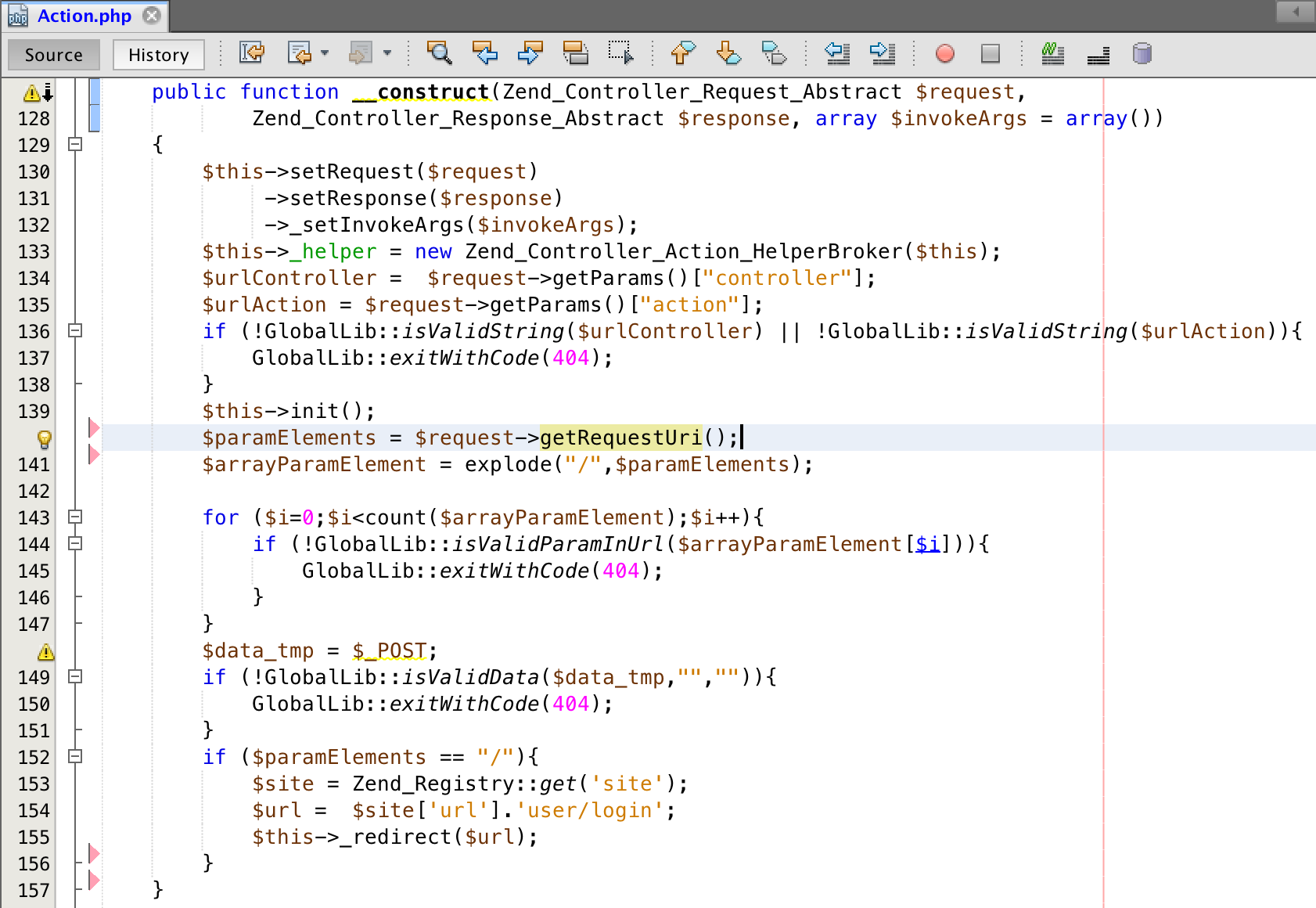


1. Verify that all application components are defined in terms of the business functions and/or security functions they provide **PASSED** (refer to 1.1)
2. Verify that all components that are not part of the application but that the application relies on to operate are defined in terms of the functions, and/or security functions, they provide **PASSED** (refer to 1.2)
3. Verify that a threat model for the target application has been produced and covers off risks associated with Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of privilege (STRIDE) **LongHV**
4. Verify all security controls (including libraries that call external security services) have a centralized implementation **PASSED**

* For **Request Receiver**, we centralize signature verification in SignatureVerifier class



* For **Operation Tool**, we centralize input sanitize in Action.php of Zend framework



1. Verify that components are segregated from each other via a defined security control, such as network segmentation, firewall rules, or cloud based security groups **TungND3**
2. Verify the application has a clear separation between the data layer, controller layer and the display layer, such that security decisions can be enforced on trusted systems **PASSED**

**Operation Tool** is using Zen Framework MVC architecture with **Config Service** as Data Layer.

1. Verify that there is no sensitive business logic, secret keys or other proprietary information in client side code **QuangLCN**
2. Verify that all application components, libraries, modules, frameworks, platform, and operating systems are free from known vulnerabilities **NhanNV2**

## References

For more information, please see:

* Threat Modelling Cheat Sheet <https://www.owasp.org/index.php/Application_Security_Architecture_Cheat_Sheet>
* Attack Surface Analysis Cheat Sheet: <https://www.owasp.org/index.php/Attack_Surface_Analysis_Cheat_Sheet>

# V2: Authentication Verification Requirements

## Control objective

Authentication is the act of establishing, or confirming, something (or someone) as authentic, that is, that claims made by or about the thing are true. Ensure that a verified application satisfies the following high level requirements:

* Verifies the digital identity of the sender of a communication.
* Ensures that only those authorised are able to authenticate and credentials are transported in a secure manner.

## Requirements

1. Verify all pages and resources by default require authentication except those specifically intended to be public (Principle of complete mediation) **QuangLCN**
2. Verify that forms containing credentials are not filled in by the application. Pre-filling by the application implies that credentials are stored in plaintext or a reversible format, which is explicitly prohibited **QuangLCN**
3. Deprecated
4. Verify all authentication controls are enforced on the server side **TaiNT6**
5. Deprecated
6. Verify all authentication controls fail securely to ensure attackers cannot log in **QuangLCN**
7. Verify password entry fields allow, or encourage, the use of passphrases, and do not prevent password managers, long passphrases or highly complex passwords being entered **QuangLCN**
8. Verify all account identity authentication functions (such as update profile, forgot password, disabled / lost token, help desk or IVR) that might regain access to the account are at least as resistant to attack as the primary authentication mechanism **QuangLCN**
9. Verify that the changing password functionality includes the old password, the new password, and a password confirmation **QuangLCN**
10. Deprecated
11. Deprecated
12. Verify that all authentication decisions can be logged, without storing sensitive session identifiers or passwords. This should include requests with relevant metadata needed for security investigations **QuangLCN**
13. Verify that account passwords are one way hashed with a salt, and there is sufficient work factor to defeat brute force and password hash recovery attacks **QuangLCN**
14. Deprecated
15. Deprecated
16. Verify that credentials are transported using a suitable encrypted link and that all pages/functions that require a user to enter credentials are done so using an encrypted link **QuangLCN**
17. Verify that the forgotten password function and other recovery paths do not reveal the current password and that the new password is not sent in clear text to the user **QuangLCN**
18. Verify that information enumeration is not possible via login, password reset, or forgot account functionality **QuangLCN**
19. Verify there are no default passwords in use for the application framework or any components used by the application (such as “admin/password”) **QuangLCN**
20. Verify that anti-automation is in place to prevent breached credential testing, brute forcing, and account lockout attacks **QuangLCN**
21. Verify that all authentication credentials for accessing services external to the application are encrypted and stored in a protected location **QuangLCN**
22. Verify that forgotten password and other recovery paths use a TOTP or other soft token, mobile push, or other offline recovery mechanism. Use of a random value in an e-mail or SMS should be a last resort and is known weak **QuangLCN**
23. Verify that account lockout is divided into soft and hard lock status, and these are not mutually exclusive. If an account is temporarily soft locked out due to a brute force attack, this should not reset the hard lock status **QuangLCN**
24. Verify that if shared knowledge based questions (also known as "secret questions") are required, the questions do not violate privacy laws and are sufficiently strong to protect accounts from malicious recovery **QuangLCN**
25. Verify that the system can be configured to disallow the use of a configurable number of previous passwords **QuangLCN**
26. Verify that risk based re-authentication, two factor or transaction signing is in place for high value transactions **QuangLCN**
27. Verify that measures are in place to block the use of commonly chosen passwords and weak passphrases **QuangLCN**
28. Verify that all authentication challenges, whether successful or failed, should respond in the same average response time **QuangLCN**
29. Verify that secrets, API keys, and passwords are not included in the source code, or online source code repositories **QuangLCN**
30. Deprecated
31. Verify that if an application allows users to authenticate, they can authenticate using two-factor authentication or other strong authentication, or any similar scheme that provides protection against username + password disclosure **QuangLCN**
32. Verify that administrative interfaces are not accessible to untrusted parties **QuangLCN**
33. Browser autocomplete, and integration with password managers are permitted unless prohibited by risk based policy **QuangLCN**

## References

For more information, please see:

* OWASP Testing Guide 4.0: Testing for Authentication <https://www.owasp.org/index.php/Testing_for_authentication>
* 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>
* Choosing and Using Security Questions at <https://www.owasp.org/index.php/Choosing_and_Using_Security_Questions_Cheat_Sheet>

# V3: Session Management Verification Requirements

## Control objective

One of the core components of any web-based application is the mechanism by which it controls and maintains the state for a user interacting with it. This is referred to this as Session Management and is defined as the set of all controls governing state-full interaction between a user and the web-based application.

Ensure that a verified application satisfies the following high level session management requirements:

* Sessions are unique to each individual and cannot be guessed or shared
* Sessions are invalidated when no longer required and timed out during periods of inactivity.

## Requirements

1. Verify that there is no custom session manager, or that the custom session manager is resistant against all common session management attacks **QuangLCN**
2. Verify that sessions are invalidated when the user logs out **QuangLCN**
3. Verify that sessions timeout after a specified period of inactivity **QuangLCN**
4. Verify that sessions timeout after an administratively-configurable maximum time period regardless of activity (an absolute timeout) **QuangLCN**
5. Verify that all pages that require authentication have easy and visible access to logout functionality **QuangLCN**
6. Verify that the session id is never disclosed in URLs, error messages, or logs. This includes verifying that the application does not support URL rewriting of session cookies **QuangLCN**
7. Verify that all successful authentication and re-authentication generates a new session and session id **QuangLCN**
8. Deprecated
9. Deprecated
10. Verify that only session ids generated by the application framework are recognized as active by the application **QuangLCN**
11. Verify that session ids are sufficiently long, random and unique across the correct active session base **QuangLCN**
12. Verify that session ids stored in cookies have their path set to an appropriately restrictive value for the application, and authentication session tokens additionally set the “HttpOnly” and “secure” attributes **QuangLCN**
13. Deprecated
14. Deprecated
15. Deprecated
16. Verify that the application limits the number of active concurrent sessions **QuangLCN**
17. Verify that an active session list is displayed in the account profile or similar of each user. The user should be able to terminate any active session **QuangLCN**
18. Verify the user is prompted with the option to terminate all other active sessions after a successful change password process **QuangLCN**

## References

For more information, please see:

* OWASP Testing Guide 4.0: Session Management Testing <https://www.owasp.org/index.php/Testing_for_Session_Management>
* OWASP Session Management Cheat Sheet: <https://www.owasp.org/index.php/Session_Management_Cheat_Sheet>

# V4: Access Control Verification Requirements

## Control objective

Authorization is the concept of allowing access to resources only to those permitted to use them. Ensure that a verified application satisfies the following high level requirements:

* Persons accessing resources holds valid credentials to do so.
* Users are associated with a well-defined set of roles and privileges.
* Role and permission metadata is protected from replay or tampering.

## Requirements

1. Verify that the principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege **QuangLCN**
2. Deprecated
3. Deprecated
4. Verify that access to sensitive records is protected, such that only authorized objects or data is accessible to each user (for example, protect against users tampering with a parameter to see or alter another user's account) **QuangLCN**
5. Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS\_Store, .git or .svn folders **QuangLCN**
6. Deprecated
7. Deprecated
8. Verify that access controls fail securely **QuangLCN**
9. Verify that the same access control rules implied by the presentation layer are enforced on the server side **TaiNT6**
10. Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized **QuangLCN**
11. Verify that there is a centralized mechanism (including libraries that call external authorization services) for protecting access to each type of protected resource **LongHV**
12. Verify that all access control decisions can be logged and all failed decisions are logged **QuangLCN**
13. Verify that the application or framework uses strong random anti-CSRF tokens or has another transaction protection mechanism **QuangLCN**
14. Verify the system can protect against aggregate or continuous access of secured functions, resources, or data. For example, consider the use of a resource governor to limit the number of edits per hour or to prevent the entire database from being scraped by an individual user **QuangLCN**
15. Verify the application has additional authorization (such as step up or adaptive authentication) for lower value systems, and / or segregation of duties for high value applications to enforce anti-fraud controls as per the risk of application and past fraud **QuangLCN**
16. Verify that the application correctly enforces context-sensitive authorisation so as to not allow unauthorised manipulation by means of parameter tampering **QuangLCN**

## References

For more information, please see:

* OWASP Testing Guide 4.0: Authorization <https://www.owasp.org/index.php/Testing_for_Authorization>
* OWASP Cheat Sheet: Access Control <https://www.owasp.org/index.php/Access_Control_Cheat_Sheet>

# V5: Malicious input handling verification requirements

## Control objective

The most common web application security weakness is the failure to properly validate input coming from the client or from the environment before using it. This weakness leads to almost all of the major vulnerabilities in web applications, such as cross site scripting, SQL injection, interpreter injection, locale/Unicode attacks, file system attacks, and buffer overflows.

Ensure that a verified application satisfies the following high level requirements:

* All input is validated to be correct and fit for the intended purpose.
* Data from an external entity or client should never be trusted and should be handled accordingly.

## Requirements

1. Verify that the runtime environment is not susceptible to buffer overflows, or that security controls prevent buffer overflows **LongHV**
2. Deprecated
3. Verify that server side input validation failures result in request rejection and are logged **NhanNV2**
4. Deprecated
5. Verify that input validation routines are enforced on the server side **NhanNV2**
6. Verify that a single input validation control is used by the application for each type of data that is accepted **QuangLCN**
7. Deprecated
8. Deprecated
9. Deprecated
10. Verify that all SQL queries, HQL, OSQL, NOSQL and stored procedures, calling of stored procedures are protected by the use of prepared statements or query parameterization, and thus not susceptible to SQL injection **TaiNT6**
11. Verify that the application is not susceptible to LDAP Injection, or that security controls prevent LDAP Injection **N/A**

**Standard Bank Interface** does not integrate with LDAP

1. Verify that the application is not susceptible to OS Command Injection, or that security controls prevent OS Command Injection **TaiNT6**
2. Verify that the application is not susceptible to Remote File Inclusion (RFI) or Local File Inclusion (LFI) when content is used that is a path to a file **TaiNT6**
3. Verify that the application is not susceptible to common XML attacks, such as XPath query tampering, XML External Entity attacks, and XML injection attacks **QuangLCN**
4. Ensure that all string variables placed into HTML or other web client code is either properly contextually encoded manually, or utilize templates that automatically encode contextually to ensure the application is not susceptible to reflected, stored and DOM Cross-Site Scripting (XSS) attacks **QuangLCN**
5. If the application framework allows automatic mass parameter assignment (also called automatic variable binding) from the inbound request to a model, verify that security sensitive fields such as “*accountBalance*”, “*role*” or “*password*” are protected from malicious automatic binding **QuangLCN**
6. Verify that the application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (GET, POST, cookies, headers, environment, etc.) **QuangLCN**
7. Verify that client side validation is used as a second line of defense, in addition to server side validation **QuangLCN**
8. Verify that all input data is validated, not only HTML form fields but all sources of input such as REST calls, query parameters, HTTP headers, cookies, batch files, RSS feeds, etc; using positive validation (whitelisting), then lesser forms of validation such as greylisting (eliminating known bad strings), or rejecting bad inputs (blacklisting) **QuangLCN**
9. Verify that structured data is strongly typed and validated against a defined schema including allowed characters, length and pattern (e.g. credit card numbers or telephone, or validating that two related fields are reasonable, such as validating suburbs and zip or post codes match) **QuangLCN**
10. Verify that unstructured data is sanitized to enforce generic safety measures such as allowed characters and length, and characters potentially harmful in given context should be escaped (e.g. natural names with Unicode or apostrophes, such as ねこ or O'Hara) **QuangLCN**
11. Make sure untrusted HTML from WYSIWYG editors or similar are properly sanitized with an HTML sanitizer and handle it appropriately according to the input validation task and encoding task **QuangLCN**
12. For auto-escaping template technology, if UI escaping is disabled, ensure that HTML sanitization is enabled instead **QuangLCN**
13. Verify that data transferred from one DOM context to another, uses safe JavaScript methods, such as using .innerText and .val **QuangLCN**
14. Verify when parsing JSON in browsers, that JSON.parse is used to parse JSON on the client. Do not use eval() to parse JSON on the client **QuangLCN**
15. Verify that authenticated data is cleared from client storage, such as the browser DOM, after the session is terminated **QuangLCN**

## References

For more information, please see:

* OWASP Testing Guide 4.0: Input Validation Testing  
  <https://www.owasp.org/index.php/Testing_for_Input_Validation>
* OWASP Cheat Sheet: Input Validation <https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet>
* OWASP Testing Guide 4.0: Testing for HTTP Parameter Pollution <https://www.owasp.org/index.php/Testing_for_HTTP_Parameter_pollution_%28OTG-INPVAL-004%29>
* OWASP LDAP Injection Cheat Sheet <https://www.owasp.org/index.php/LDAP_Injection_Prevention_Cheat_Sheet>
* OWASP Testing Guide 4.0: Client Side Testing <https://www.owasp.org/index.php/Client_Side_Testing>
* OWASP Cross Site Scripting Prevention Cheat Sheet <https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet>
* OWASP Java Encoding Project  
  <https://www.owasp.org/index.php/OWASP_Java_Encoder_Project>

For more information on auto-escaping, please see

* Reducing XSS by way of Automatic Context-Aware Escaping in Template Systems <http://googleonlinesecurity.blogspot.com/2009/03/reducing-xss-by-way-of-automatic.html>
* AngularJS Strict Contextual Escaping <https://docs.angularjs.org/api/ng/service/$sce>
* <https://cwe.mitre.org/data/definitions/915.html>

# V6: Output encoding / escaping

**This section was incorporated into V5 in Application Security Verification Standard 2.0. ASVS requirement 5.16 addresses contextual output encoding to help prevent Cross Site Scripting.**

# V7: Cryptography at rest verification requirements

## Control objective

Ensure that a verified application satisfies the following high level requirements:

* That all cryptographic modules fail in a secure manner and that errors are handled correctly.
* That a suitable random number generator is used when randomness is required.
* That access to keys is managed in a secure way.

## Requirements

1. Deprecated
2. Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable oracle padding **TaiNT6**
3. Deprecated
4. Deprecated
5. Deprecated
6. Verify that all random numbers, random file names, random GUIDs, and random strings are generated using the cryptographic module’s approved random number generator when these random values are intended to be not guessable by an attacker **TaiNT6**
7. Verify that cryptographic algorithms used by the application have been validated against FIPS 140-2 or an equivalent standard **TaiNT6**
8. Verify that cryptographic modules operate in their approved mode according to their published security policies **NhanNV2**
9. Verify that there is an explicit policy for how cryptographic keys are managed (e.g., generated, distributed, revoked, and expired). Verify that this key lifecycle is properly enforced **NhanNV2**
10. Deprecated
11. Verify that all consumers of cryptographic services do not have direct access to key material. Isolate cryptographic processes, including master secrets and consider the use of a virtualized or physical hardware key vault (HSM) **NhanNV2**
12. *Personally Identifiable Information* should be stored encrypted at rest and ensure that communication goes via protected channels **NhanNV2**
13. Verify that sensitive passwords or key material maintained in memory is overwritten with zeros as soon as it is no longer required, to mitigate memory dumping attacks **NhanNV2**
14. Verify that all keys and passwords are replaceable, and are generated or replaced at installation time **NhanNV2**
15. Verify that random numbers are created with proper entropy even when the application is under heavy load, or that the application degrades gracefully in such circumstances **NhanNV2**

## References

For more information, please see:

* OWASP Testing Guide 4.0: Testing for weak Cryptography <https://www.owasp.org/index.php/Testing_for_weak_Cryptography>
* OWASP Cheat Sheet: Cryptographic Storage <https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet>

# V8: Error handling and logging verification requirements

## Control objective

The primary objective of error handling and logging is to provide a useful reaction by the user, administrators, and incident response teams. The objective is not to create massive amounts of logs, but high quality logs, with more signal than discarded noise.

High quality logs will often contain sensitive data, and must be protected as per local data privacy laws or directives. This should include:

* Not collecting or logging sensitive information if not specifically required.
* Ensuring all logged information is handled securely and protected as per its data classification.
* Ensuring that logs are not forever, but have an absolute lifetime that is as short as possible.

If logs contain private or sensitive data, the definition of which varies from country to country, the logs become some of the most sensitive information held by the application and thus very attractive to attackers in their own right.

## Requirements

1. Verify that the application does not output error messages or stack traces containing sensitive data that could assist an attacker, including session id, software/framework versions and personal information **QuangLCN**
2. Verify that error handling logic in security controls denies access by default **QuangLCN**
3. Verify security logging controls provide the ability to log success and particularly failure events that are identified as security-relevant **NhanNV2**
4. Verify that each log event includes necessary information that would allow for a detailed investigation of the timeline when an event happens **NhanNV2**
5. Verify that all events that include untrusted data will not execute as code in the intended log viewing software **NhanNV2**
6. Verify that security logs are protected from unauthorized access and modification **NhanNV2**
7. Verify that the application does not log sensitive data as defined under local privacy laws or regulations, organizational sensitive data as defined by a risk assessment, or sensitive authentication data that could assist an attacker, including user’s session identifiers, passwords, hashes, or API tokens **NhanNV2**
8. Verify that all non-printable symbols and field separators are properly encoded in log entries, to prevent log injection **NhanNV2**
9. Verify that log fields from trusted and untrusted sources are distinguishable in log entries **NhanNV2**
10. Verify that an audit log or similar allows for non-repudiation of key transactions **NhanNV2**
11. Verify that security logs have some form of integrity checking or controls to prevent unauthorized modification **NhanNV2**
12. Verify that the logs are stored on a different partition than the application is running with proper log rotation **NhanNV2**
13. Time sources should be synchronized to ensure logs have the correct time **NhanNV2**

## References

For more information, please see:

* OWASP Testing Guide 4.0 content: Testing for Error Handling <https://www.owasp.org/index.php/Testing_for_Error_Handling>

# V9: Data protection verification requirements

## Control objective

There are three key elements to sound data protection: Confidentiality, Integrity and Availability (CIA). This standard assumes that data protection is enforced on a trusted system, such as a server, which has been hardened and has sufficient protections.

Applications have to assume that all user devices are compromised in some way. Where an application transmits or stores sensitive information on insecure devices, such as shared computers, phones and tablets, the application is responsible for ensuring data stored on these devices is encrypted and cannot be easily illicitly obtained, altered or disclosed.

Ensure that a verified application satisfies the following high level data protection requirements:

* **Confidentiality**: Data should be protected from unauthorised observation or disclosure both in transit and when stored.
* **Integrity**: Data should be protected being maliciously created, altered or deleted by unauthorized attackers.
* **Availability**: Data should be available to authorized users as required

## Requirements

1. Verify that all forms containing sensitive information have disabled client side caching, including autocomplete features **QuangLCN**
2. Verify that the list of sensitive data processed by the application is identified, and that there is an explicit policy for how access to this data must be controlled, encrypted and enforced under relevant data protection directives **QuangLCN**
3. Verify that all sensitive data is sent to the server in the HTTP message body or headers (i.e., URL parameters are never used to send sensitive data) **QuangLCN**
4. Verify that the application sets appropriate anti-caching headers as per the risk of the application, such as the following: **QuangLCN**

Expires: Tue, 03 Jul 2001 06:00:00 GMT

Last-Modified: {now} GMT

Cache-Control: no-store, no-cache, must-revalidate, max-age=0

Cache-Control: post-check=0, pre-check=0

Pragma: no-cache

1. Verify that on the server, all cached or temporary copies of sensitive data stored are protected from unauthorized access or purged/invalidated after the authorized user accesses the sensitive data **QuangLCN**
2. Verify that there is a method to remove each type of sensitive data from the application at the end of the required retention policy **NhanNV2**
3. Verify the application minimizes the number of parameters in a request, such as hidden fields, Ajax variables, cookies and header values **QuangLCN**
4. Verify the application has the ability to detect and alert on abnormal numbers of requests for data harvesting for an example screen scraping **QuangLCN**
5. Verify that data stored in client side storage (such as HTML5 local storage, session storage, IndexedDB, regular cookies or Flash cookies) does not contain sensitive data or PII **QuangLCN**
6. Verify accessing sensitive data is logged, if the data is collected under relevant data protection directives or where logging of accesses is required **NhanNV2**
7. Verify that sensitive information maintained in memory is overwritten with zeros as soon as it is no longer required, to mitigate memory dumping attacks **NhanNV2**

## References

For more information, please see:

* User Privacy Protection Cheat Sheet: <https://www.owasp.org/index.php/User_Privacy_Protection_Cheat_Sheet>

# V10: Communications security verification requirements

## Control objective

Ensure that a verified application satisfies the following high level requirements:

* That TLS is used where sensitive data is transmitted
* That strong algorithms and ciphers are used at all times.

## Requirements

1. Verify that a path can be built from a trusted CA to each Transport Layer Security (TLS) server certificate, and that each server certificate is valid **TungND3**
2. Deprecated
3. Verify that TLS is used for all connections (including both external and backend connections) that are authenticated or that involve sensitive data or functions, and does not fall back to insecure or unencrypted protocols. Ensure the strongest alternative is the preferred algorithm **NhanNV2**
4. Verify that backend TLS connection failures are logged **TungND3**
5. Verify that certificate paths are built and verified for all client certificates using configured trust anchors and revocation information **TungND3**
6. Verify that all connections to external systems that involve sensitive information or functions are authenticated **NhanNV2**
7. Deprecated
8. Verify that there is a single standard TLS implementation that is used by the application that is configured to operate in an approved mode of operation **TungND3**
9. Deprecated
10. Verify that TLS certificate public key pinning (HPKP) is implemented with production and backup public keys. For more information, please see the references below **TungND3**
11. Verify that HTTP Strict Transport Security headers are included on all requests and for all subdomains, such as Strict-Transport-Security: max-age=15724800; includeSubdomains **QuangLCN**
12. Verify that production website URL has been submitted to preloaded list of Strict Transport Security domains maintained by web browser vendors. Please see the references below **QuangLCN**
13. Ensure forward secrecy ciphers are in use to mitigate passive attackers recording traffic **QuangLCN**
14. Verify that proper certification revocation, such as Online Certificate Status Protocol (OCSP) Stapling, is enabled and configured **QuangLCN**
15. Verify that only strong algorithms, ciphers, and protocols are used, through all the certificate hierarchy, including root and intermediary certificates of your selected certifying authority **NhanNV2**
16. Verify that the TLS settings are in line with current leading practice, particularly as common configurations, ciphers, and algorithms become insecure **TungND3**

## References

For more information, please see:

* **OWASP – TLS Cheat Sheet.** <https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet>
* **Notes on “Approved modes of TLS”**. In the past, the ASVS referred to the US standard FIPS 140-2, but as a global standard, applying US standards this can be difficult, contradictory, or confusing to apply. A better method of achieving compliance with 10.8 would be to review guides such as (<https://wiki.mozilla.org/Security/Server_Side_TLS)>, generate known good configurations (<https://mozilla.github.io/server-side-tls/ssl-config-generator/>), and use known TLS evaluation tools, such as sslyze, various vulnerability scanners or trusted TLS online assessment services to obtain a desired level of security. In general, we see non-compliance for this section being the use of outdated or insecure ciphers and algorithms, the lack of perfect forward secrecy, outdated or insecure SSL protocols, weak preferred ciphers, and so on.
* **Certificate pinning**. For more information please review <https://tools.ietf.org/html/rfc7469>. The rationale behind certificate pinning for production and backup keys is business continuity - see <https://noncombatant.org/2015/05/01/about-http-public-key-pinning/>
* OWASP Certificate Pinning Cheat Sheet  
  <https://www.owasp.org/index.php/Pinning_Cheat_Sheet>
* OWASP Certificate and Public Key Pinning  
  <https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning>
* Time of first use (TOFU) Pinning   
  <https://developer.mozilla.org/en/docs/Web/Security/Public_Key_Pinning>
* **Pre-loading HTTP Strict Transport Security**<https://www.chromium.org/hsts>

# V11: HTTP security configuration verification requirements

## Control objective

Ensure that a verified application satisfies the following high level requirements:

* The application server is suitably hardened from a default configuration
* HTTP responses contain a safe character set in the content type header.

## Requirements

1. Verify that the application accepts only a defined set of required HTTP request methods, such as GET and POST are accepted, and unused methods (e.g. TRACE, PUT, and DELETE) are explicitly blocked **QuangLCN**
2. Verify that every HTTP response contains a content type header specifying a safe character set (e.g., UTF-8, ISO 8859-1) **QuangLCN**
3. Verify that HTTP headers added by a trusted proxy or SSO devices, such as a bearer token, are authenticated by the application **QuangLCN**
4. Verify that a suitable X-FRAME-OPTIONS header is in use for sites where content should not be viewed in a 3rd-party X-Frame **QuangLCN**
5. Verify that the HTTP headers or any part of the HTTP response do not expose detailed version information of system components **QuangLCN**
6. Verify that all API responses contain X-Content-Type-Options: nosniff and Content-Disposition: attachment; filename="api.json" (or other appropriate filename for the content type) **QuangLCN**
7. Verify that a content security policy (CSPv2) is in place that helps mitigate common DOM, XSS, JSON, and JavaScript injection vulnerabilities **QuangLCN**
8. Verify that the X-XSS-Protection: 1; mode=block header is in place to enable browser reflected XSS filters **QuangLCN**

## References

For more information, please see:

* OWASP Testing Guide 4.0: Testing for HTTP Verb Tampering <https://www.owasp.org/index.php/Testing_for_HTTP_Verb_Tampering_%28OTG-INPVAL-003%29>
* Adding Content-Disposition to API responses helps prevent many attacks based on misunderstanding on the MIME type between client and server, and the "filename" option specifically helps prevent Reflected File Download attacks.   
  <https://www.blackhat.com/docs/eu-14/materials/eu-14-Hafif-Reflected-File-Download-A-New-Web-Attack-Vector.pdf>
* <https://www.owasp.org/index.php?title=Content_Security_Policy_Cheat_Sheet&setlang=en>

# V12: Security configuration verification requirements

**This section was incorporated into V11 in Application Security Verification Standard 2.0.**

# V13: Malicious controls verification requirements

## Control objective

Ensure that a verified application satisfies the following high level requirements:

* Malicious activity is handled securely and properly as to not affect the rest of the application.
* Do not have time bombs or other time based attacks built into them
* do not “phone home” to malicious or unauthorized destinations
* Applications do not have back doors, Easter eggs, salami attacks, or logic flaws that can be controlled by an attacker

Malicious code is extremely rare, and is difficult to detect. Manual line by line code review can assist looking for logic bombs, but even the most experienced code reviewer will struggle to find malicious code even if they know it exists. This section is not possible to complete without access to source code, including as many third party libraries as possible.

## Requirements

1. Verify all malicious activity is adequately sandboxed, containerized or isolated to delay and deter attackers from attacking other applications **TungND3**
2. Verify that the application source code, and as many third party libraries as possible, does not contain back doors, Easter eggs, and logic flaws in authentication, access control, input validation, and the business logic of high value transactions **NhanNV2**

## References

For more information, please see:

* <http://www.dwheeler.com/essays/apple-goto-fail.html>

# V14: Internal security verification requirements

**This section was incorporated into V13 in Application Security Verification Standard 2.0.**

# V15: Business logic verification requirements

## Control objective

Ensure that a verified application satisfies the following high level requirements:

* The business logic flow is sequential and in order
* Business logic includes limits to detect and prevent automated attacks, such as continuous small funds transfers, or adding a million friends one at a time, and so on.
* High value business logic flows have considered abuse cases and malicious actors, and have protections against spoofing, tampering, repudiation, information disclosure, and elevation of privilege attacks.

## Requirements

1. Verify the application will only process business logic flows in sequential step order, with all steps being processed in realistic human time, and not process out of order, skipped steps, process steps from another user, or too quickly submitted transactions **NhanNV2**
2. Verify the application has business limits and correctly enforces on a per user basis, with configurable alerting and automated reactions to automated or unusual attack **QuangLCN**

## References

For more information, please see:

* OWASP Testing Guide 4.0: Business Logic Testing <https://www.owasp.org/index.php/Testing_for_business_logic>
* OWASP Cheat Sheet: <https://www.owasp.org/index.php/Business_Logic_Security_Cheat_Sheet>

# V16: Files and resources verification requirements

## Control objective

Ensure that a verified application satisfies the following high level requirements:

* Untrusted file data should be handled accordingly and in a secure manner
* Obtained from untrusted sources are stored outside the webroot and limited permissions.

## Requirements

1. Verify that URL redirects and forwards only allow whitelisted destinations, or show a warning when redirecting to potentially untrusted content **QuangLCN**
2. Verify that untrusted file data submitted to the application is not used directly with file I/O commands, particularly to protect against path traversal, local file include, file mime type, and OS command injection vulnerabilities **QuangLCN**
3. Verify that files obtained from untrusted sources are validated to be of expected type and scanned by antivirus scanners to prevent upload of known malicious content **QuangLCN**
4. Verify that untrusted data is not used within inclusion, class loader, or reflection capabilities to prevent remote/local file inclusion vulnerabilities **QuangLCN**
5. Verify that untrusted data is not used within cross-domain resource sharing (CORS) to protect against arbitrary remote content **QuangLCN**
6. Verify that files obtained from untrusted sources are stored outside the webroot, with limited permissions, preferably with strong validation **QuangLCN**
7. Verify that the web or application server is configured by default to deny access to remote resources or systems outside the web or application server **QuangLCN**
8. Verify the application code does not execute uploaded data obtained from untrusted sources **QuangLCN**
9. Do not use Flash, Active-X, Silverlight, NACL, client-side Java or other client side technologies not supported natively via W3C browser standards **QuangLCN**

## References

For more information, please see:

* File Extension Handling for Sensitive Information: <https://www.owasp.org/index.php/Unrestricted_File_Upload>

# V17: Mobile verification requirements

## Control objective

This section contains controls that are mobile application specific. These controls have been de-duplicated from 2.0, so must be taken in conjunction with all other sections of the relevant ASVS Verification Level.

Mobile applications should:

* Should have the same level of security controls within the mobile client as found in the server, by enforcing security controls in a trusted environment
* Sensitive information assets stored on the device should be done so in a secure manner
* All sensitive data transmitted from the device should be done so with transport layer security in mind.

## Requirements

1. Verify that ID values stored on the device and retrievable by other applications, such as the UDID or IMEI number are not used as authentication tokens **TaiNT6**
2. Verify that the mobile app does not store sensitive data onto potentially unencrypted shared resources on the device (e.g. SD card or shared folders) **N/A**
3. Verify that sensitive data is not stored unprotected on the device, even in system protected areas such as key chains **NhanNV2**
4. Verify that secret keys, API tokens, or passwords are dynamically generated in mobile applications **N/A**
5. Verify that the mobile app prevents leaking of sensitive information (for example, screenshots are saved of the current application as the application is backgrounded or writing sensitive information in console) **N/A**
6. Verify that the application is requesting minimal permissions for required functionality and resources **QuangLCN**
7. Verify that the application sensitive code is laid out unpredictably in memory (For example ASLR) **NhanNV2**
8. Verify that there are anti-debugging techniques present that are sufficient enough to deter or delay likely attackers from injecting debuggers into the mobile app (For example GDB) **N/A**
9. Verify that the app does not export sensitive activities, intents, or content providers for other mobile apps on the same device to exploit **N/A**
10. Verify that sensitive information maintained in memory is overwritten with zeros as soon as it is no longer required, to mitigate memory dumping attacks **NhanNV2**
11. Verify that the app validates input to exported activities, intents, or content providers **QuangLCN**

## References

For more information, please see:

* OWASP Mobile Security Project: <https://www.owasp.org/index.php/OWASP_Mobile_Security_Project>
* iOS Developer Cheat Sheet: <https://www.owasp.org/index.php/IOS_Developer_Cheat_Sheet>

# V18: Web services verification requirements

## Control objective

Ensure that a verified application that uses RESTful or SOAP based web services has:

* Adequate authentication, session management and authorization of all web services
* Input validation of all parameters that transit from a lower to higher trust level
* Basic interoperability of SOAP web services layer to promote API use

## Requirements

1. Verify that the same encoding style is used between the client and the server **QuangLCN**
2. Verify that access to administration and management functions within the Web Service Application is limited to web service administrators **QuangLCN**
3. Verify that all input is limited to an appropriate size limit **QuangLCN**
4. Verify that all input is limited to an appropriate size limit **QuangLCN**
5. Verify that SOAP based web services are compliant with Web Services-Interoperability (WS-I) Basic Profile at minimum. This essentially means TLS encryption **N/A**
6. Verify the use of session-based authentication and authorization. Please refer to sections 2, 3 and 4 for further guidance. Avoid the use of static "API keys" and similar **QuangLCN**
7. Verify that the REST service is protected from Cross-Site Request Forgery via the use of at least one or more of the following: ORIGIN checks, double submit cookie pattern, CSRF nonces, and referrer checks **NhanNV2**
8. Verify the REST service explicitly check the incoming Content-Type to be the expected one, such as application/xml or application/json **NhanNV2**
9. Verify that the message payload is signed to ensure reliable transport between client and service, using JSON Web Signing or WS-Security for SOAP requests **N/A**
10. Verify that alternative and less secure access paths do not exist **QuangLCN**

## References

For more information, please see:

* OWASP Testing Guide 4.0: Configuration and Deployment Management Testing <https://www.owasp.org/index.php/Testing_for_configuration_management>
* OWASP Cross-Site Request Forgery cheat sheet  
  <https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet>
* JSON Web Tokens (and Signing)  
  <https://jwt.io/>

# V19. Configuration

## Control objective

Ensure that a verified application has:

* Up to date libraries and platform(s).
* A secure by default configuration.
* Sufficient hardening that user initiated changes to default configuration do not unnecessarily expose or create security weaknesses or flaws to underlying systems.

## Requirements

1. All components should be up to date with proper security configuration(s) and version(s). This should include removal of unneeded configurations and folders such as sample applications, platform documentation, and default or example users **NhanNV2**
2. Communications between components, such as between the application server and the database server, should be encrypted, particularly when the components are in different containers or on different systems **NhanNV2**
3. Communications between components, such as between the application server and the database server should be authenticated using an account with the least necessary privileges **NhanNV2**
4. Verify application deployments are adequately sandboxed, containerized or isolated to delay and deter attackers from attacking other applications **TungND3**
5. Verify that the application build and deployment processes are performed in a secure fashion **TungND3**
6. Verify that authorised administrators have the capability to verify the integrity of all security-relevant configurations to ensure that they have not been tampered with **TungND3**
7. Verify that all application components are signed **NhanNV2**
8. Verify that third party components come from trusted repositories **NhanNV2**
9. Verify that build processes for system level languages have all security flags enabled, such as ASLR, DEP, and security checks **NhanNV2**
10. Verify that all application assets, such as JavaScript libraries, CSS stylesheets and web fonts are hosted by the application rather than rely on a CDN or external provider **QuangLCN**

## References

For more information, please see:

* OWASP Testing Guide 4.0: Configuration and Deployment Management Testing <https://www.owasp.org/index.php/Testing_for_configuration_management>

# Appendix A: What ever happened to…

| **Original #** | **Description** | **Status** | | **Removed** | **Reason** |
| --- | --- | --- | --- | --- | --- |
| **2.3** | Verify that if a maximum number of authentication attempts is exceeded, the account is locked for a period of time long enough to deter brute force attacks. | Deprecated | | 2.0 | A more complex requirement replaced it (v2.20) |
| **2.5** | Verify that all authentication controls (including libraries that call external authentication services) have a centralized implementation. | Merged | | 3.0 | Genericized to include all security controls and moved to 1.10 |
| **2.10** | Verify that re-authentication is required before any application- specific sensitive operations are permitted. | Deprecated | | 2.0 | Re-authentication is so rarely observed that we decided to remove the control |
| **2.11** | Verify that after an administratively- configurable period of time, authentication credentials expire. | Deprecated | | 2.0 | Absolute timeouts and credential expiry removed as not being an effective control. |
| **2.14** | Verify that all authentication credentials for accessing services external to the application are encrypted and stored in a protected location (not in source code). | Updated | | 2.0 | Became V2.21 |
| **2.15** | Verify that all code implementing or using authentication controls is not affected by any malicious code. | Moved | | 2.0 | Moved to V13 - Malicious Code |
| **2.30** | Verify that if an application allows users to authenticate, they use a proven secure authentication mechanism. | | Deprecated | 3.0.1 | Too ambiguous to be tested, actually a summary of all the V2 requirements |
| **3.8** | Verify that the session id is changed upon re-authentication | | Updated | 3.0 | Rolled into 3.7 |
| **3.9** | Verify that the session id is changed or cleared on logout | | Updated | 3.0 | Rolled into 3.7 |
| **3.13** | Verify that all code implementing or using session management controls is not affected by any malicious code | | Moved | 2.0 | Moved to V13 - Malicious code |
| **3.14** | Verify that authenticated session tokens using cookies are protected by the use of "HttpOnly". | | Updated | 3.0 | Moved into 3.13 |
| **3.15** | Verify that authenticated session tokens using cookies are protected with the "secure" attribute. | | Updated | 3.0 | Moved into 3.13 |
| **4.2** | Verify that users can only access secured URLs for which they possess specific authorization. | | Updated | 3.0 | Rolled into 4.1 |
| **4.3** | Verify that users can only access secured data files for which they possess specific authorization. | | Updated | 3.0 | Rolled into 4.1 |
| **4.13** | Verify that limitations on input and access imposed by the business on the application (such as daily transaction limits or sequencing of tasks) cannot be bypassed. | | Moved | 3.0 | Moved to V15 Business Logic |
| **4.15** | Verify that all code implementing or using access controls is not affected by any malicious code. | | Moved | 2.0 | Moved to V13 Malicious Controls |
| **5.2** | Verify that a positive validation pattern is defined and applied to all input | | Deprecated | 2.0 | Removed as too difficult to implement particularly for free form text inputs |
| **5.4** | Verify that a character set, such as UTF-8, is specified for all sources of input | | Deprecated | 3.0 | Removed as too difficult to implement in most languages |
| **5.7** | Verify that all input validation failures are logged. | | Deprecated | 3.0 | Removed as would create too many useless logs that would be ignored |
| **5.8** | Verify that all input data is canonicalized for all downstream decoders or interpreters prior to validation. | | Deprecated | 3.0 | Removed as Type 1 JSP technology specific and not an issue for most modern frameworks |
| **5.9** | Verify that all input validation controls are not affected by any malicious code | | Moved | 2.0 | Moved to V13 Malicious controls |
| **5.14** | Verify that the runtime environment is not susceptible to XML Injections or that security controls prevents XML Injections | | Merged | 3.0 | Merged with V5.13 |
| **5.15** | -- EMPTY REQUIREMENT -- | | Deleted | 3.0 | This requirement never existed |
| **5.19** | Verify that for each type of output encoding/escaping performed by the application, there is a single security control for that type of output for the intended destination | | Merged | 3.0 | Genericized to include all security controls and moved to 1.10 |
| **7.1** | Verify that all cryptographic functions used to protect secrets from the application user are implemented server side | | Deprecated | 3.0 | Many modern responsive and mobile apps include this by design |
| **7.3** | Verify that access to any master secret(s) is protected from unauthorized access (A master secret is an application credential stored as plaintext on disk that is used to protect access to security configuration information). | | Moved | 3.0 | Moved to V2.29 |
| **7.4** | Verify that password hashes are salted when they are created | | Moved | 2.0 | Moved to V2.13 |
| **7.5** | Verify that cryptographic module failures are logged | | Deprecated | 2.0 | Creating unnecessary logs that are never reviewed is counterproductive |
| **7.10** | Verify that all code supporting or using a cryptographic module is not affected by any malicious code | | Moved | 2.0 | Moved to V13 |
| **8.2** | Verify that all error handling is performed on trusted devices | |  | 3.0 | Deprecated |
| **8.3** | Verify that all logging controls are implemented on the server. | | Moved | 3.0 | Became a more generic architectural control V1.13 |
| **8.9** | Verify that there is a single application-level logging implementation that is used by the software. | | Moved | 3.0 | Became a more generic architectural control V1.13 |
| **8.11** | Verify that a log analysis tool is available which allows the analyst to search for log events based on combinations of search criteria across all fields in the log record format supported by this system. | | Deprecated | 3.0 | Removed as not required for secure software |
| **8.12** | Verify that all code implementing or using error handling and logging controls is not affected by any malicious code. | | Moved | 2.0 | Moved to V13 Malicious Controls |
| **8.15** | Verify that logging is performed before executing the transaction. If logging was unsuccessful (e.g. disk full, insufficient permissions) the application fails safe. This is for when integrity and non-repudiation are a must. | | Deprecated | 3.0 | Removed as too detailed a control that would only be applicable to small percentage of all apps |
| **10.2** | Verify that failed TLS connections do not fall back to an insecure HTTP connection | | Merged | 3.0 | Merged with 10.3 |
| **10.7** | Verify that all connections to external systems that involve sensitive information or functions use an account that has been set up to have the minimum privileges necessary for the application to function properly | |  |  |  |
| **10.9** | Verify that specific character encodings are defined for all connections (e.g., UTF-8). | |  |  |  |
| **11.1** | Deprecated | |  |  |  |
| **11.4** | Deprecated | |  |  |  |
| **11.5** | Deprecated | |  |  |  |
| **11.6** | Deprecated | |  |  |  |
| **11.7** | Deprecated | |  |  |  |
| **11.8** | Deprecated | |  |  |  |
| **11.4** | Deprecated | |  |  |  |
| **13.1** | Deprecated | |  |  |  |
| **13.2** | Deprecated | |  |  |  |
| **13.3** | Deprecated | |  |  |  |
| **13.4** | Deprecated | |  |  |  |
| **13.5** | Deprecated | |  |  |  |
| **13.6** | Deprecated | |  |  |  |
| **13.7** | Deprecated | |  |  |  |
| **13.8** | Deprecated | |  |  |  |
| **13.9** | Deprecated | |  |  |  |
| **15.1-15.7 15.9** | Business Logic Section. | | Merged | 3.0 | Most of section 15 has been merged into 15.8 and 15.10. |
| **15.11** | Verify that the application covers off risks associated with Spoofing, Tampering, Repudiation, Information Disclosure, and Elevation of privilege (STRIDE). | | Duplicate | 3.0 | Duplicated requirement. Captured by V1.6 |
| **16.4** | Verify that parameters obtained from untrusted sources are not used in manipulating filenames, pathnames or any file system object without first being canonicalized and input validated to prevent local file inclusion attacks. | | Moved | 3.0 | Moved to V16.2 |
| **17.1** | Verify that the client validates SSL certificates | | Deprecated | 3.0 | Duplicated requirement. General requirement already captured by V10. |
| **17.7** | Deprecated | |  |  |  |
| **17.8** | Deprecated | |  |  |  |
| **17.10** | Deprecated | |  |  |  |
| **17.11** | Deprecated | |  |  |  |
| **17.12** | Deprecated | |  |  |  |
| **17.13** | Deprecated | |  |  |  |
| **17.14** | Deprecated | |  |  |  |
| **17.15** | Deprecated | |  |  |  |
| **17.16** | Deprecated | |  |  |  |
| **17.17** | Deprecated | |  |  |  |
| **17.18** | Deprecated | |  |  |  |
| **17.19** | Deprecated | |  |  |  |
| **17.20** | Deprecated | |  |  |  |
| **17.22** | Deprecated | |  |  |  |
| **17.23** | Deprecated | |  |  |  |
| **17.24** | Deprecated | |  |  |  |

# Appendix B: Glossary

* **Access Control** – A means of restricting access to files, referenced functions, URLs, and data based on the identity of users and/or groups to which they belong.
* **Address Space Layout Randomization (ASLR)** – A technique to help protect against buffer overflow attacks.
* **Application Security** – Application-level security focuses on the analysis of components that comprise the application layer of the Open Systems Interconnection Reference Model (OSI Model), rather than focusing on for example the underlying operating system or connected networks.
* **Application Security Verification** – The technical assessment of an application against the OWASP ASVS.
* **Application Security Verification Report** – A report that documents the overall results and supporting analysis produced by the verifier for a particular application.
* **Authentication** – The verification of the claimed identity of an application user.
* **Automated Verification** – The use of automated tools (either dynamic analysis tools, static analysis tools, or both) that use vulnerability signatures to find problems.
* **Back Doors** – A type of malicious code that allows unauthorized access to an application.
* **Blacklist** – A list of data or operations that are not permitted, for example a list of characters that are not allowed as input.
* **Cascading Style Sheets** (CSS) - A style sheet language used for describing the presentation semantics of document written in a markup language, such as HTML.
* **Certificate Authority** (CA) – An entity that issues digital certificates.
* **Communication Security** – The protection of application data when it is transmitted between application components, between clients and servers, and between external systems and the application.
* **Component** – a self-contained unit of code, with associated disk and network interfaces that communicates with other components.
* **Cross-Site Scripting** (XSS) – A security vulnerability typically found in web applications allowing the injection of client-side scripts into content.
* **Cryptographic module** – Hardware, software, and/or firmware that implements cryptographic algorithms and/or generates cryptographic keys.
* **Denial of Service (DoS) Attacks** – The flooding of an application with more requests than it can handle.
* **Design Verification** – The technical assessment of the security architecture of an application.
* **Dynamic Verification** – The use of automated tools that use vulnerability signatures to find problems during the execution of an application.
* **Easter Eggs** – A type of malicious code that does not run until a specific user input event occurs.
* **External Systems** – A server-side application or service that is not part of the application.
* **FIPS 140-2** – A standard that can be used as the basis for the verification of the design and implementation of cryptographic modules
* **Globally Unique Identifier** (GUID) – a unique reference number used as an identifier in software.
* **HyperText Markup Language (HTML)** - The main markup language for the creation of web pages and other information displayed in a web browser.
* **Hyper Text Transfer Protocol** (HTTP) – An application protocol for distributed, collaborative, hypermedia information systems. It is the foundation of data communication for the World Wide Web.
* **Input Validation** – The canonicalization and validation of untrusted user input.
* **Lightweight Directory Access Protocol (LDAP)** – An application protocol for accessing and maintaining distributed directory information services over a network.
* **Malicious Code** – Code introduced into an application during its development unbeknownst to the application owner, which circumvents the application’s intended security policy. Not the same as malware such as a virus or worm!
* **Malware** – Executable code that is introduced into an application during runtime without the knowledge of the application user or administrator.
* **Open Web Application Security Project** (OWASP) – The Open Web Application Security Project (OWASP) is a worldwide free and open community focused on improving the security of application software. Our mission is to make application security "visible," so that people and organizations can make informed decisions about application security risks. See: http://www.owasp.org/
* **Output encoding** – The canonicalization and validation of application output to Web browsers and to external systems.
* **Personally Identifiable Information** (PII) - is information that can be used on its own or with other information to identify, contact, or locate a single person, or to identify an individual in context.
* **Positive** **validation** – See whitelist.
* **Security Architecture** – An abstraction of an application’s design that identifies and describes where and how security controls are used, and also identifies and describes the location and sensitivity of both user and application data.
* **Security Configuration** – The runtime configuration of an application that affects how security controls are used.
* **Security Control** – A function or component that performs a security check (e.g. an access control check) or when called results in a security effect (e.g. generating an audit record).
* **SQL Injection (SQLi)** – A code injection technique used to attack data driven applications, in which malicious SQL statements are inserted into an entry point.
* **Static Verification** – The use of automated tools that use vulnerability signatures to find problems in application source code.
* **Target of Verification (TOV)** – If you are performing application security verification according to the OWASP ASVS requirements, the verification will be of a particular application. This application is called the “Target of Verification” or simply the TOV.
* **Threat Modeling** - A technique consisting of developing increasingly refined security architectures to identify threat agents, security zones, security controls, and important technical and business assets.
* **Transport Layer Security** – Cryptographic protocols that provide communication security over the Internet
* **URI/URL/URL fragments** – A Uniform Resource Identifier is a string of characters used to identify a name or a web resource. A Uniform Resource Locator is often used as a reference to a resource.
* **User acceptance testing (UAT)**– Traditionally a test environment that behaves like the production environment where all software testing is performed before going live.
* **Verifier** - The person or team that is reviewing an application against the OWASP ASVS requirements.
* **Whitelist** – A list of permitted data or operations, for example a list of characters that are allowed to perform input validation.
* **XML** – A markup language that defines a set of rules for encoding documents.

# Appendix C: References

The following OWASP projects are most likely to be useful to users/adopters of this standard:

* OWASP Testing Guide  
  <https://www.owasp.org/index.php/OWASP_Testing_Project>
* OWASP Code Review Guide  
  <http://www.owasp.org/index.php/Category:OWASP_Code_Review_Project>
* OWASP Cheat Sheets  
  <https://www.owasp.org/index.php/OWASP_Cheat_Sheet_Series>
* OWASP Proactive Controls  
  <https://www.owasp.org/index.php/OWASP_Proactive_Controls>
* OWASP Top 10  
  <https://www.owasp.org/index.php/Top_10_2013-Top_10>
* OWASP Mobile Top 10  
  <https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks>

Similarly, the following web sites are most likely to be useful to users/adopters of this standard:

* MITRE Common Weakness Enumeration - <http://cwe.mitre.org/>
* PCI Security Standards Council - <https://www.pcisecuritystandards.org>
* PCI Data Security Standard (DSS) v3.0 Requirements and Security Assessment Procedures <https://www.pcisecuritystandards.org/documents/PCI_DSS_v3.pdf>

# Appendix D: Standards Mappings

PCI DSS 6.5 is derived from the OWASP Top 10 2004/2007, with some recent process extensions. The ASVS is a strict superset of the OWASP Top 10 2013 (154 items to 10 items), so all of the issues covered by OWASP Top 10 and PCI DSS 6.5.x are handled by more fine grained ASVS control requirements. For example, “Broken authentication and session management” maps exactly to sections V2 Authentication and V3 Session Management.

Full mapping is achieved by verification level 3, although verification level 2 will address most PCI DSS 6.5 requirements except 6.5.3 and 6.5.4. Process issues, such as PCI DSS 6.5.6, are not covered by the ASVS.

|  |  |  |
| --- | --- | --- |
| PCI-DSS 3.0 | ASVS 3.0 | Description |
| 6.5.1 Injection flaws, particularly SQL injection. Also consider OS Command Injection, LDAP and XPath injection flaws as well as other injection flaws | 5.11, 5.12, 5.13, 8.14, 16.2 | Exact mapping. |
| 6.5.2 Buffer overflows | 5.1 | Exact mapping |
| 6.5.3 Insecure cryptographic storage | v7 - all | Comprehensive mapping from Level 1 up |
| 6.5.4 Insecure communications | v10 - all | Comprehensive mapping from Level 1 up |
| 6.5.5 Improper error handling | 3.6, 7.2, 8.1, 8.2 | Exact mapping |
| 6.5.7 Cross-site scripting (XSS) | 5.16, 5.20, 5.21, 5.24, 5.25, 5.26, 5.27, 11.4,11.15 | ASVS breaks down XSS into several requirements highlighting the complexity of XSS defense especially for legacy applications |
| 6.5.8 Improper Access Control (such as insecure direct object references, failure to restrict URL access, directory traversal and failure to restrict user access to functions). | v4 - all | Comprehensive mapping from Level 1 up |
| 6.5.9 Cross-site request forgery (CSRF). | 4.13 | Exact mapping. ASVS considers CSRF defense to be an aspect of access control. |
| 6.5.10 Broken authentication and session management. | v2 and v3 - all | Comprehensive mapping from Level 1 up |