SECURITY In Five

Secure Application Development Guidelines

Table of Contents

Overview

Security is a vital component of any application and information system and must be designed into the project from the beginning. This document lays out the security principles and guidelines for designing, developing and deploying a secure applications. This document is built from industry best practices accumulated from sources such as the Open Web Application Security Project (OWASP), OWASP Application Security Verification Standard, National Institute of Standards and Technology (NIST), International Organization of Standardization (ISO), Microsoft and others.

People responsible for the design and development of an application must be familiar with all of the security control areas within this document. Regardless of the application's purpose this document covers the best practice security control considerations that should be integrated in every application developed. It is important that all applications and information systems built provides a consistent level of security, reducing the risk of litigation or regulatory compliance issues on products built for clients.

This document is a guide to a best practice for secure application development. The security controls are organized by security controls and detailed to their applicability to the level of security you are working toward. This document is to be used during the design and development lifecycles to direct security discussions and ensure security controls and requirements are not ignored, overlooked or underdeveloped.

This document is best used during the design phase of a project as a blueprint to create the security checklist specific to the application, platform, organization and/or industry. This will allow you to tailor your use cases and development stories around security that best fits your project's needs.

Note – The security controls are mapped directly from the OWASP Application Security Verification Standard.

Security Principles

In designing and building an application security should be part of the process throughout. Regardless of the functionality of the application there are security principles that should always be considered. Following security principles will help guide the design, development, configuration, deployment and support that will result in enhanced security of an application and information system.

Simplicity

Keep things simple. The simplest security is often the most effective. If the security in and around a system is too complex for the user it will either not be used, used improperly/incorrectly and users will attempt to bypass it.

Security Documentation

It is important to include adequate documentation of an application's security with test plans to verify compliance to best practices or other requirements (client or regulatory). The documentation should also include details on how to user, support and maintain the application as to not compromise or reduce the security.

Containerization

Containerization, segregation or compartmentalization of users, processes and data will assist in remediating and limiting any security problems. If a component of an application is compromised the attacker can be limited or blocked from expanding into other components through proper separation.

Defense In Layers

In addition to application containerization the design on the information system should be in multiple layers. There should be checks between each layer and if one layer is compromised that should not result in the entire system being compromised.

User Input Is Never To Be Trusted

Any input from a user should be filtered, validated, scrubbed, and scrutinized per the application's rules. The primary source of attacks on applications are in areas where user input is taken, see the OWASP Top 10. Assume all user input is malicious until verified.

Least Privilege / Need-To-Know

All users, processes, operators, and privileged users should operate with the minimum level of privilege needed.

Authenticate At The Door

Users should be authenticated and authorization verified prior to accessing the information system. These checks should be done as early as possible to prevent unwanted exposure of data to unauthorized individuals.

Fail Closed

Applications and information systems should be designed so that if a component fails it fails securely or closed. Ensure that in the event of a failure no data is left exposed or accessible. A failure should not display any information to the user to the reason of the failure or any details of the systems that failed. (i.e. web server and version). Those details could provide details to identify vulnerabilities for an attacker.

Security Defaults

Applications and information systems should be designed and deployed in such a way that by default they are secure. As an example, all administrator accounts are renamed, disabled if not needed and other security restrictions are closed by default and require manual enablement.

Limit the Attack Surface

If a resource, account, process, system, etc. is not needed, remove or disable it. Only have the needed ports open, shut off protocols not used, and be cognizant of your functionality. By reducing the attack area you make it harder for the attackers.

Security Through Obscurity Does Not Exist

Security through obscurity is a dangerous mentality. The obscurity only exists in the mind of a person. The scripts, bots, analyzers the attackers use will piece together your obscure puzzle eventually. Obscuring applications and data is not the same as protecting it. Obfuscation relies on the premise that no one will stumble upon it. Once the obfuscation is discovered the system is no longer secure.

Security Control Levels

The security controls through this document are categorized into three levels. These levels determine the security strength of your application. For example, If an application is determined to require Level 2 protections, then all controls marked as L2 will be part of the design.

Level 1 – Bare minimum

A Level 1 application will have security controls in place to protect against easy to detect vulnerabilities and including the OWASP Top 10 and other baseline checklists. Level 1 controls are used for applications that do not process any sensitive data, generally public and static pages and would not cause any damage to the organization if it were compromised.

Every application should be Level 1 ready. Level 1 controls can be checked through automated tools or manual verification without access to source code. Applications that are Level 1 will be attacked through low level methods, automated scanning and using easy to find vulnerabilities.

If you application processes any data deemed sensitive, classified or otherwise protected Level 1 is not sufficient.

Level 2 – Standard, Most Applications

Applications that are at a Level 2 will be able to adequately defend against most of the risks associated with software today. Level 2 ensures security controls are in place, used properly and working appropriately.

Threats to Level 2 applications will be skilled, targeted and focused attackers using tactical tools that are specialized in discovering and exploiting vulnerabilities and weaknesses in applications.

Level 3 – High value, Critical, Sensitive Applications.

Level 3 applications have the highest level of security controls. Applications that require Level 3 are ones that if they fail or are compromised would have a significant and potentially irreversible damage to an organization. These applications are processing highly sensitive data or are in highly regulated or sensitive industries like healthcare, payment processing, government/military, insurance, etc…

Level 3 applications require far deeper analysis, architecture, coding and testing than the other two levels. A Level 3 applications is modularized for resiliency, scalability, performance and most important the layers of security. Level 3 applications have defense in depth where each module takes care of its own security controls.

Application Development

Secure application development is required for every development project regardless of size or scope. The controls listed below are industry standard controls that should be reviewed with every project and client at design and requirement phase. This initial review should be able to be completed in less than an hour with a detailed follow up with the proper client resources at a later time.

Application Development Security controls areas:

1. Architecture, Design and Threat Modeling
2. Authentication Verification
3. Session Management Verification
4. Access Control Verification
5. Validation, Sanitization and Encoding Verification
6. Stored Cryptography Verification
7. Error Handling and Logging Verification
8. Data Protection Verification
9. Communications Verification
10. Malicious Code Verification
11. Business Logic Verification
12. File and Resources Verification
13. API and Web Service Verification
14. Configuration Verification

Security Controls

The security controls and the domains listed are based on best practices for application development. These are not requirements until the clients reviews and/or delivers their similar requirements during project design phase. Not all controls are going to be implemented or applicable in every project. These are to ensure that when security reviews are conducted on projects we can deliver and provide the path to the most secure application we can create.

The earlier security can be designed, coded, configured and implemented into a project the faster those can be placed into production and the time saved on remediations can be placed into more numerous and advanced features.

1. Architecture, Design and Threat Modeling

Security architecture needs to evolve into a more flexible practice in the DevSecOps world. The core components of security architecture remains unchanged. Availability, Confidentiality, Integrity and Privacy are always going to be goals regardless of how applications are built and delivered. In order to meet the speed and flexibility architecture needs to 'shift-left', be as early and often as possible.

The 'shift-left' momentum begins with the developer enablement. Equipping the teams with the proper training, security checklists (like this document), mentoring, secure coding, security testing, deployment, configuration and operations. Security professionals are required now to keep up with the agile processes which means the security teams need to learn to code, work closer with developers at the creative phases and be involved before coding begins rather than at the end.

Sample User Stories

Secure Software Development Lifecycle

Go To References

Control	*Description*	*L1*	*L2*	*L3*
1.1.1	Verify the use of a secure software development lifecycle that addresses security in all stages of development.		X	X
1.1.2	Verify the use of threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing.		X	X
1.1.3	Verify that all user stories and features contain functional security constraints, such as "As a user, I should be able to view and edit my profile. I should not be able to view or edit anyone else's profile".		X	X
1.1.4	Verify documentation and justification of all the application's trust boundaries, components, and significant data flows.		X	X
1.1.5	Verify definition and security analysis of the application's high-level architecture and all connected remote services.		X	X
1.1.6	Verify implementation of centralized, simple (economy of design), vetted, secure, and reusable security controls to avoid duplicate, missing, ineffective, or insecure controls.		X	X
1.1.7	Verify availability of a secure coding checklist, security requirements, guideline, or policy to all developers and testers.		X	X

Go To References

Authentication Architecture

Authentication design can be rendered useless if an attacker can reset accounts through social engineering, guess weak passwords and communication channels and other bypassing attempts. Authentication must have the same strength through all the pathways of identity proofing.

Go To References

Control	Description	L1	L2	L3
1.2.1	Verify the use of unique or special low-privilege operating system accounts for all application components, services, and servers.		X	X
1.2.2	Verify that communications between application components, including APIs, middleware and data layers, are authenticated. Components should have the least necessary privileges needed.		X	X
1.2.3	Verify that the application uses a single vetted authentication mechanism that is known to be secure, can be extended to include strong authentication, and has sufficient logging and monitoring to detect account abuse or breaches.		X	X
1.2.4	Verify that all authentication pathways and identity management APIs implement consistent authentication security control strength, such that there are no weaker alternatives per the risk of the application.		X	X

Go To References

Access Control Architecture

Go To References

Control	Description	L1	L2	L3
1.4.1	Verify that trusted enforcement points such as at access control gateways, servers, and serverless functions enforce access controls. Never enforce access controls on the client.		X	X
1.4.2	Verify that the chosen access control solution is flexible enough to meet the application's needs.		X	X
1.4.3	Verify enforcement of the principle of least privilege in functions, data files, URLs, controllers, services, and other resources. This implies protection against spoofing and elevation of privilege.		X	X
1.4.4	Verify the application uses a single and well-vetted access control mechanism for accessing protected data and resources. All requests must pass through this single mechanism to avoid copy and paste or insecure alternative paths.		X	X
1.4.5	Verify that attribute or feature-based access control is used whereby the code checks the user's authorization for a feature/data item rather than just their role. Permissions should still be allocated using roles.		X	X

Go To References

Input and Output Architecture

Input validation is one of the top security functions and routines that should be put into every single application that has user input capabilities. Input validation is one of the top proactive measurers to protect against the top OWASP vulnerabilities. This gap will be called out in any static code analysis every time as a High or Critical finding.

Input validation is performed to ensure only proper formed data is entering the workflow in an application/system to prevent data persisting in a database and triggering to interrupt downstream components. Input validation needs to happen as early as possible in the data flow, preferable as soon as the data is received by an external party.

Input validation is NOT the primary method of preventing the attacks listed below but is a significant contribution to reducing impacts if implemented properly. Review the references for this section to see the detailed documents and cheat sheets.

It is important to note that the concept of input validation to be 'server side' is obsolete in today's cloud computing world. The term 'trusted service layer' is used by OWASP to mean any trusted enforcement point, regardless of location, like an API, microservice, Single page application, server side, etc…

The following are the common vulnerabilities that are handled through input validations:

- **Cross-site Scripting (XSS)** – This allows code to be injected into web pages, typically HTML or JavaScript, by malicious users and results in the execution of that code when it is loaded by other users to steal credentials, exfiltrate data or damage the application.
- **Command/SQL Injection** – This allows user-supplied input to be interpreted as part of the system command or query.
- **Header Injection** – This allows the manipulation of HTTP response headers that are dynamically generated based on user inputs. A common attack is the Carriage Return Line Feed (CRLF) which allows an attacker to inject HTTP response headers that control the behavior of the returned webpage. This type of injection is common with XSS or Phishing attacks.
- **URL Redirection** - This allows input submitted to a redirect function to be manipulated in order to redirect the victim to an alternate page or site of an attacker's choosing. This is common in Phishing attacks.
- **Directory Traversal** – This allows an attacker to submit input file names containing characters representing a directory traverse to a parent (e.g. \ ). This attack attempts to access and/or execute files that are not normally accessible.

Input validation should be applied on both syntactical and semantic levels.

- **Syntactical** – Enforce correct syntax of structured fields. (Ex. SSN, CCD, Dates, Email…)
- **Semantic** – Enforce the correctness of the values in the specific business context.

- Start data before end date, numbers within expected ranges, email checks…

Whitelisting validation is the appropriate method for all input fields. This approach will force developers to clearly define what IS approved and exclude anything else. On structured input (SSN, CCD, Dates, etc) the format rules can be very strong to the exact input patterns.

When validating free-form input the primary methods of validating are:

- **Normalization** – Ensure canonical encoding is used across all the text and no invalid characters are present.
- **Whitelisting** – Unicode allows for whitelisting categories such as 'decimal digits' and 'letters'.
- **Individual character whitelisting -** If you need to allow an apostrophe for names but not the whole character category.

Go To References

Control	Description	L1	L2	L3
1.5.1	Verify that input and output requirements clearly define how to handle and process data based on type, content, and applicable laws, regulations, and other policy compliance.		X	X
1.5.2	Verify that serialization is not used when communicating with untrusted clients. If this is not possible, ensure that adequate integrity controls (and possibly encryption if sensitive data is sent) are enforced to prevent deserialization attacks including object injection.		X	X
1.5.3	Verify that input validation is enforced on a trusted service layer.		X	X
1.5.4	Verify that output encoding occurs close to or by the interpreter for which it is intended.		X	X

Go To References

Cryptographic Architecture

Applications need to be designed with strong cryptographic architecture to protect data assets according to their data classifications. Encrypting everything is wasteful but not encrypting properly can be legally negligent.

Go To References

Control	Description	L1	L2	L3
1.6.1	Verify that there is an explicit policy for management of cryptographic keys and that a cryptographic key lifecycle follows a key management standard such as NIST SP 800-57.		X	X
1.6.2	Verify that consumers of cryptographic services protect key material and other secrets by using key vaults or API based alternatives.		X	X
1.6.3	Verify that all keys and passwords are replaceable and are part of a well-defined process to re-encrypt sensitive data.		X	X

1.6.4	Verify that symmetric keys, passwords, or API secrets generated by or shared with clients are used only in protecting low risk secrets, such as encrypting local storage, or temporary ephemeral uses such as parameter obfuscation. Sharing secrets with clients is clear-text equivalent and architecturally should be treated as such.		X	X

Go To References

Errors, Logging and Auditing Architecture

Go To References

Control	Description	L1	L2	L3
1.7.1	Verify that a common logging format and approach is used across the system.		X	X
1.7.2	Verify that logs are securely transmitted to a preferably remote system for analysis, detection, alerting, and escalation.		X	X

Go To References

Data Protection and Privacy Architecture

Go To References

Control	Description	L1	L2	L3
1.8.1	Verify that all sensitive data is identified and classified into protection levels.		X	X
1.8.2	Verify that all protection levels have an associated set of protection requirements, such as encryption requirements, integrity requirements, retention, privacy and other confidentiality requirements, and that these are applied in the architecture.		X	X

Go To References

Communications Architecture

Go To References

Control	Description	L1	L2	L3
1.9.1	Verify the application encrypts communications between components, particularly when these components are in different containers, systems, sites, or cloud providers.		X	X
1.9.2	Verify that application components verify the authenticity of each side in a communication link to prevent person-in-the-middle attacks. For example, application components should validate TLS certificates and chains.		X	X

Go To References

Malicious Software Architecture

Go To References

Control	Description	L1	L2	L3
1.10.1	Verify that a source code control system is in use, with procedures to ensure that check-ins are accompanied by issues or change tickets. The source code		X	X

	control system should have access control and identifiable users to allow traceability of any changes.			

Go To References

Business Logic Architecture

Go To References

Control	Description	L1	L2	L3
1.11.1	Verify the definition and documentation of all application components in terms of the business or security functions they provide.		X	X
1.11.2	Verify that all high-value business logic flows, including authentication, session management and access control, do not share unsynchronized state.		X	X
1.11.3	Verify that all high-value business logic flows, including authentication, session management and access control are thread safe and resistant to time-of-check and time-of-use race conditions.			X

Go To References

Secure File Upload Architecture

Go To References

Control	Description	L1	L2	L3
1.12.1	Verify that user-uploaded files are stored outside of the web root.		X	X
1.12.2	Verify that user-uploaded files - if required to be displayed or downloaded from the application - are served by either octet stream downloads, or from an unrelated domain, such as a cloud file storage bucket. Implement a suitable content security policy to reduce the risk from XSS vectors or other attacks from the uploaded file.		X	X

Go To References

Configuration Architecture

Go To References

Control	Description	L1	L2	L3
1.14.1	Verify the segregation of components of differing trust levels through well-defined security controls, firewall rules, API gateways, reverse proxies, cloud-based security groups, or similar mechanisms.		X	X
1.14.2	Verify that if deploying binaries to untrusted devices makes use of binary signatures, trusted connections, and verified endpoints.		X	X
1.14.3	Verify that the build pipeline warns of out-of-date or insecure components and takes appropriate actions.		X	X
1.14.4	Verify that the build pipeline contains a build step to automatically build and verify the secure deployment of the application, particularly if the application infrastructure is software defined, such as cloud environment build scripts.		X	X

1.14.5	Verify that application deployments adequately sandbox, containerize and/or isolate at the network level to delay and deter attackers from attacking other applications, especially when they are performing sensitive or dangerous actions such as deserialization.		X	X
1.14.6	Verify the application does not use unsupported, insecure, or deprecated client-side technologies such as NSAPI plugins, Flash, Shockwave, ActiveX, Silverlight, NACL, or client-side Java applets.		X	X
1.14.1	Verify the segregation of components of differing trust levels through well-defined security controls, firewall rules, API gateways, reverse proxies, cloud-based security groups, or similar mechanisms.		X	X

Go To References

2. Authentication Verification

Authentication is the act of establishing/confirming someone or something as authentic and the claims made are correct, resistant to impersonation, altercation and interception.

In NIST 800-63 passwords are now called ‘memorized secrets’ and include passwords, PINs, patterns, image selections and passphrases. They are still considered ‘something you know’ and still a single factor authentication. Applications should strongly consider multi-factor authentication implementations and allow users to re-use already owned tokens or credential services.

Sample User Stories

Password Security

Go To References

Control	Description	L1	L2	L3
2.1.1	Verify that user set passwords are at least 12 characters in length.	X	X	X
2.1.2	Verify that passwords 64 characters or longer are permitted.	X	X	X
2.1.3	Verify that passwords can contain spaces and truncation is not performed. Consecutive multiple spaces MAY optionally be coalesced.	X	X	X
2.1.4	Verify that Unicode characters are permitted in passwords. A single Unicode code point is considered a character, so 12 emoji or 64 kanji characters should be valid and permitted.	X	X	X
2.1.5	Verify users can change their password.	X	X	X
2.1.6	Verify that password change functionality requires the user's current and new password.	X	X	X
2.1.7	Verify that passwords submitted during account registration, login, and password change are checked against a set of breached passwords either locally (such as the top 1,000 or 10,000 most common passwords which match the system's password policy) or using an external API. If using an API a zero knowledge proof or other mechanism should be used to ensure that the plain text password is not sent or used in verifying the breach status of the password. If the password is breached, the application must require the user to set a new non-breached password.	X	X	X
2.1.8	Verify that a password strength meter is provided to help users set a stronger password.	X	X	X
2.1.9	Verify that there are no password composition rules limiting the type of characters permitted. There should be no requirement for upper or lower case or numbers or special characters.	X	X	X
2.1.10	Verify that there are no periodic credential rotation or password history requirements.	X	X	X
2.1.11	Verify that "paste" functionality, browser password helpers, and external password managers are permitted.	X	X	X

2.1.12	Verify that the user can choose to either temporarily view the entire masked password, or temporarily view the last typed character of the password on platforms that do not have this as native functionality.			

Go To References

General Authenticators

Go To References

Control	Description	L1	L2	L3
2.2.1	Verify that anti-automation controls are effective at mitigating breached credential testing, brute force, and account lockout attacks. Such controls include blocking the most common breached passwords, soft lockouts, rate limiting, CAPTCHA, ever increasing delays between attempts, IP address restrictions, or risk-based restrictions such as location, first login on a device, recent attempts to unlock the account, or similar. Verify that no more than 100 failed attempts per hour is possible on a single account.	X	X	X
2.2.2	Verify that the use of weak authenticators (such as SMS and email) is limited to secondary verification and transaction approval and not as a replacement for more secure authentication methods. Verify that stronger methods are offered before weak methods, users are aware of the risks, or that proper measures are in place to limit the risks of account compromise.	X	X	X
2.2.3	Verify that secure notifications are sent to users after updates to authentication details, such as credential resets, email or address changes, logging in from unknown or risky locations. The use of push notifications - rather than SMS or email - is preferred, but in the absence of push notifications, SMS or email is acceptable as long as no sensitive information is disclosed in the notification.	X	X	X
2.2.4	Verify impersonation resistance against phishing, such as the use of multi-factor authentication, cryptographic devices with intent (such as connected keys with a push to authenticate), or at higher AAL levels, client-side certificates.		X	X
2.2.5	Verify that where a credential service provider (CSP) and the application verifying authentication are separated, mutually authenticated TLS is in place between the two endpoints.		X	X
2.2.6	Verify replay resistance through the mandated use of OTP devices, cryptographic authenticators, or lookup codes.		X	X
2.2.7	Verify intent to authenticate by requiring the entry of an OTP token or user-initiated action such as a button press on a FIDO hardware key.		X	X

Go To References

Authenticator Lifecycle

Authenticators are passwords, soft tokens, hard tokens, and biometric methods. The life cycle of the authenticators is essential and critical to the application. If a user can self-register without providing evidence to the identity there can be little to no trust in the identity assertion.

The industry direction is that passwords are to NOT have a maximum life or be subject to forced rotation on a scheduled basis. Rather passwords should be checked for being breached.

Go To References

Control	Description	L1	L2	L3
2.3.1	Verify system generated initial passwords or activation codes SHOULD be securely randomly generated, SHOULD be at least 6 characters long, and MAY contain letters and numbers, and expire after a short period of time. These initial secrets must not be permitted to become the long-term password.	X	X	X
2.3.2	Verify that enrollment and use of subscriber-provided authentication devices are supported, such as a U2F or FIDO tokens.		X	X
2.3.3	Verify that renewal instructions are sent with sufficient time to renew time bound authenticators.			X

Go To References

Credential Storage

Architects and developers should closely adhere to this section when developing or changing code. This section can only be verified through source code reviews, secure unit or integration testing. Penetration tests cannot identify any issues in the controls below and therefore none are marked as Level 1 per the definition but these should be considered for all applications.

Go To References

Control	Description	L1	L2	L3
2.4.1	Verify that passwords are stored in a form that is resistant to offline attacks. Passwords SHALL be salted and hashed using an approved one-way key derivation or password hashing function. Key derivation and password hashing functions take a password, a salt, and a cost factor as inputs when generating a password hash.		X	X
2.4.2	Verify that the salt is at least 32 bits in length and be chosen arbitrarily to minimize salt value collisions among stored hashes. For each credential, a unique salt value and the resulting hash SHALL be stored.		X	X
2.4.3	Verify that if PBKDF2 is used, the iteration count SHOULD be as large as verification server performance will allow, typically at least 100,000 iterations.		X	X
2.4.4	Verify that if bcrypt is used, the work factor SHOULD be as large as verification server performance will allow, typically at least 13.		X	X

2.4.5	Verify that an additional iteration of a key derivation function is performed, using a salt value that is secret and known only to the verifier. Generate the salt value using an approved random bit generator [SP 800-90Ar1] and provide at least the minimum security strength specified in the latest revision of SP 800-131A. The secret salt value SHALL be stored separately from the hashed passwords (e.g., in a specialized device like a hardware security module).		X	X

Go To References

Credential Recovery

Go To References

Control	Description	L1	L2	L3
2.5.1	Verify that a system generated initial activation or recovery secret is not sent in clear text to the user.	X	X	X
2.5.2	Verify password hints or knowledge-based authentication (so-called "secret questions") are not present.	X	X	X
2.5.3	Verify password credential recovery does not reveal the current password in any way.	X	X	X
2.5.4	Verify shared or default accounts are not present (e.g. "root", "admin", or "sa").	X	X	X
2.5.5	Verify that if an authentication factor is changed or replaced, that the user is notified of this event.	X	X	X
2.5.6	Verify forgotten password, and other recovery paths use a secure recovery mechanism, such as TOTP or other soft token, mobile push, or another offline recovery mechanism.	X	X	X
2.5.7	Verify that if OTP or multi-factor authentication factors are lost, that evidence of identity proofing is performed at the same level as during enrollment.		X	X

Go To References

Look-Up Secret Verifiers

Look-up codes are sent to users securely. They are used once and if all the look-up codes are used the secret list is discarded. These are considered ‘something you have’ authentication factor.

Go To References

Control	Description	L1	L2	L3
2.6.1	Verify that lookup secrets can be used only once.		X	X
2.6.2	Verify that lookup secrets have sufficient randomness (112 bits of entropy), or if less than 112 bits of entropy, salted with a unique and random 32-bit salt and hashed with an approved one-way hash.		X	X
2.6.3	Verify that lookup secrets are resistant to offline attacks, such as predictable values.		X	X

Go To References

Out of Band Verifiers

Out of band verifiers are SMS text messages, which should not be considered for use, secure email or a secure channel to deliver verification codes like a push to a mobile device.

Note the use the SMS authentication method is considered 'restricted' by NIST and should be depreciated in favor of push notification or similar methods.

Go To References

Control	Description	L1	L2	L3
2.7.1	Verify that clear text out of band (NIST "restricted") authenticators, such as SMS or PSTN, are not offered by default, and stronger alternatives such as push notifications are offered first.	X	X	X
2.7.2	Verify that the out of band verifier expires out of band authentication requests, codes, or tokens after 10 minutes.	X	X	X
2.7.3	Verify that the out of band verifier authentication requests, codes, or tokens are only usable once, and only for the original authentication request.	X	X	X
2.7.4	Verify that the out of band authenticator and verifier communicates over a secure independent channel.	X	X	X
2.7.5	Verify that the out of band verifier retains only a hashed version of the authentication code.			X
2.7.6	Verify that the initial authentication code is generated by a secure random number generator, containing at least 20 bits of entropy (typically a six digital random number is sufficient).			X

Go To References

Single or Multi-Factor One Time Verifiers

Single factor one time passwords (OTPs) are physical or soft tokens that display a continually changing one time challenge. Multi-factor authentication tokens are similar to a single factor OTP but require a valid PIN, biometric, USB pairing or some other additional value.

Go To References

Control	Description	L1	L2	L3
2.8.1	Verify that time-based OTPs have a defined lifetime before expiring.	X	X	X
2.8.2	Verify that symmetric keys used to verify submitted OTPs are highly protected, such as by using a hardware security module or secure operating system based key storage.		X	X
2.8.3	Verify that approved cryptographic algorithms are used in the generation, seeding, and verification.		X	X
2.8.4	Verify that time-based OTP can be used only once within the validity period.		X	X
2.8.5	Verify that if a time-based multi factor OTP token is re-used during the validity period, it is logged and rejected with secure notifications being sent to the holder of the device.		X	X

2.8.6	Verify physical single factor OTP generator can be revoked in case of theft or other loss. Ensure that revocation is immediately effective across logged in sessions, regardless of location.		X	X
2.8.7	Verify that biometric authenticators are limited to use only as secondary factors in conjunction with either something you have and something you know.		X	X

Go To References

Cryptographic Software and Devices Verifiers

Cryptographic security keys are smart cards or other device where the user has to plug in or pair the device to the computer to complete authentication. These are becoming more popular through the lowered costs of the new, strong USB keys coming on the market.

Go To References

Control	*Description*	*L1*	*L2*	*L3*
2.9.1	Verify that cryptographic keys used in verification are stored securely and protected against disclosure, such as using a TPM or HSM, or an OS service that can use this secure storage.		X	X
2.9.2	Verify that the challenge nonce is at least 64 bits in length, and statistically unique or unique over the lifetime of the cryptographic device.		X	X
2.9.3	Verify that approved cryptographic algorithms are used in the generation, seeding, and verification.		X	X

Go To References

Service Authentication

This section cannot be verified through penetration testing so does not have any L1 controls. However, remember that clear text storage of secrets is unacceptable under any circumstances.

Go To References

Control	Description	L1	L2	L3
2.10.1	Verify that integration secrets do not rely on unchanging passwords, such as API keys or shared privileged accounts.		X	X
2.10.2	Verify that if passwords are required, the credentials are not a default account.		X	X
2.10.3	Verify that passwords are stored with sufficient protection to prevent offline recovery attacks, including local system access.		X	X
2.10.4	Verify passwords, integrations with databases and third-party systems, seeds and internal secrets, and API keys are managed securely and not included in the source code or stored within source code repositories. Such storage SHOULD resist offline attacks. The use of a secure software key store, hardware trusted platform module (TPM), or a hardware security module (L3) is recommended for password storage.		X	X

Go To References

3. Session Management Verification

Session management is what makes the stateless protocols stateful which is critical to differentiate and maintain different users and devices.

It's vital to ensure that a verified application complies to the 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 time out during a period of inactivity.

Sample User Stories

Fundamental Session Management

Go To References

Control	Description	L1	L2	L3
3.1.1	Verify the application never reveals session tokens in URL parameters or error messages.	X	X	X

Go To References

Session Binding

Go To References

Control	Description	L1	L2	L3
3.2.1	Verify the application generates a new session token on user authentication.	X	X	X
3.2.2	Verify that session tokens possess at least 64 bits of entropy.	X	X	X
3.2.3	Verify the application only stores session tokens in the browser using secure methods such as appropriately secured cookies (see section 3.4) or HTML 5 session storage.	X	X	X
3.2.4	Verify that session token are generated using approved cryptographic algorithms.		X	X

Go To References

Session Logout and Timeout

Review with the project for time out requirements. Use NIST 800-63 as the ceiling limits on timeouts.

Go To References

Control	Description	L1	L2	L3
3.3.1	Verify that logout and expiration invalidate the session token, such that the back button or a downstream relying party does not resume an authenticated session, including across relying parties.	X	X	X

3.3.2	If authenticators permit users to remain logged in, verify that re-authentication occurs periodically both when actively used or after an idle period.	30 Days	12 hours or 30 min. of inactivity, 2FA optional	12 hours or 15 min. of inactivity with 2FA required
3.3.3	Verify that the application terminates all other active sessions after a successful password change, and that this is effective across the application, federated login (if present), and any relying parties.		X	X
3.3.4	Verify that users are able to view and log out of any or all currently active sessions and devices.		X	X

Go To References

Cookie Based Session Management

Go To References

Control	Description	L1	L2	L3
3.4.1	Verify that cookie-based session tokens have the 'Secure' attribute set. (C6)	X	X	X
3.4.2	Verify that cookie-based session tokens have the 'HttpOnly' attribute set. (C6)	X	X	X
3.4.3	Verify that cookie-based session tokens utilize the 'SameSite' attribute to limit exposure to cross-site request forgery attacks.	X	X	X
3.4.4	Verify that cookie-based session tokens use "__Host-" prefix (see references) to provide session cookie confidentiality.	X	X	X
3.4.5	Verify that if the application is published under a domain name with other applications that set or use session cookies that might override or disclose the session cookies, set the path attribute in cookie-based session tokens using the most precise path possible.	X	X	X

Go To References

Token Based Session Management

Go To References

Control	Description	L1	L2	L3
3.5.1	Verify the application does not treat OAuth and refresh tokens — on their own — as the presence of the subscriber and allows users to terminate trust relationships with linked applications.		X	X
3.5.2	Verify the application uses session tokens rather than static API secrets and keys, except with legacy implementations.		X	X

3.5.3	Verify that stateless session tokens use digital signatures, encryption, and other countermeasures to protect against tampering, enveloping, replay, null cipher, and key substitution attacks.		X	X

Go To References

Re-Authentication From A Federation Or Assertion

This section relates to a relying party (RP) or credential service provider (CSP).

Go To References

Control	Description	L1	L2	L3
3.6.1	Verify that relying parties specify the maximum authentication time to CSPs and that CSPs re-authenticate the subscriber if they haven't used a session within that period.			X
3.6.2	Verify that CSPs inform relying parties of the last authentication event, to allow RPs to determine if they need to re-authenticate the user.			X

Go To References

Defenses Against Session Management Exploits

Go To References

Control	Description	L1	L2	L3
3.7.1	Verify the application ensures a valid login session or requires re-authentication or secondary verification before allowing any sensitive transactions or account modifications.	X	X	X

Go To References

4. Access Control Verification

Authorization is the concept of allowing access to resources to only those permitted to use them. A verified application needs to ensure the high-level requirements are met:

- Individuals accessing resources hold 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.

Sample User Stories

General Access Control Design

Go To References

Control	Description	L1	L2	L3
4.1.1	Verify that the application enforces access control rules on a trusted service layer, especially if client-side access control is present and could be bypassed.	X	X	X
4.1.2	Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.	X	X	X
4.1.3	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.	X	X	X
4.1.4	Verify that the principle of deny by default exists whereby new users/roles start with minimal or no permissions and users/roles do not receive access to new features until access is explicitly assigned.	X	X	X
4.1.5	Verify that access controls fail securely including when an exception occurs.	X	X	X

Go To References

Operation Level Access Control

Go To References

Control	Description	L1	L2	L3
4.2.1	Verify that sensitive data and APIs are protected against direct object attacks targeting creation, reading, updating and deletion of records, such as creating or updating someone else's record, viewing everyone's records, or deleting all records.	X	X	X
4.2.2	Verify that the application or framework enforces a strong anti-CSRF mechanism to protect authenticated functionality, and effective anti-automation or anti-CSRF protects unauthenticated functionality.	X	X	X

Go To References

Other Access Control Considerations

Go To References

Control	Description	L1	L2	L3
4.3.1	Verify administrative interfaces use appropriate multi-factor authentication to prevent unauthorized use.	X	X	X
4.3.2	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.	X	X	X
4.3.3	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.		X	X

Go To References

5. Validation, Sanitation and Encoding Verification

The most common application weakness is the failure to properly validate input coming from the user or environment before using it without proper encoding. This weakness leads to almost all of the significant vulnerabilities in application like Cross-Site Scripting, Injection, file system, attacks and buffer overflows.

The application must ensure the high-level requirements are met:

- Input validation and output encoding architecture have an agreed pipeline to prevent injection attacks.
- Input data is strongly typed, validated, range or length checked, sanitized or filtered.
- Output data is encoded or escaped as per the context of the data as close to the interpreter as possible.

Sample User Stories

Input Validation

Properly implemented input validation controls, using whitelisting and strong data typing, can eliminate more than 90% of injection attacks. Developers and secure code reviewers should treat this section as if L1 is required for all items to prevent injection.

Go To References

Control	Description	L1	L2	L3
5.1.1	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, or environment variables).	X	X	X
5.1.2	Verify that frameworks protect against mass parameter assignment attacks, or that the application has countermeasures to protect against unsafe parameter assignment, such as marking fields private or similar.	X	X	X
5.1.3	Verify that all input (HTML form fields, REST requests, URL parameters, HTTP headers, cookies, batch files, RSS feeds, etc) is validated using positive validation (whitelisting).	X	X	X
5.1.4	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 checking that suburb and zip/postcode match).	X	X	X
5.1.5	Verify that URL redirects and forwards only allow whitelisted destinations, or show a warning when redirecting to potentially untrusted content.	X	X	X

Go To References

Sanitization and Sandboxing

Go To References

Control	Description	L1	L2	L3
5.2.1	Verify that all untrusted HTML input from WYSIWYG editors or similar is properly sanitized with an HTML sanitizer library or framework feature.	X	X	X
5.2.2	Verify that unstructured data is sanitized to enforce safety measures such as allowed characters and length.	X	X	X
5.2.3	Verify that the application sanitizes user input before passing to mail systems to protect against SMTP or IMAP injection.	X	X	X
5.2.4	Verify that the application avoids the use of eval() or other dynamic code execution features. Where there is no alternative, any user input being included must be sanitized or sandboxed before being executed.	X	X	X
5.2.5	Verify that the application protects against template injection attacks by ensuring that any user input being included is sanitized or sandboxed.	X	X	X
5.2.6	Verify that the application protects against SSRF attacks, by validating or sanitizing untrusted data or HTTP file metadata, such as filenames and URL input fields, use whitelisting of protocols, domains, paths and ports.	X	X	X
5.2.7	Verify that the application sanitizes, disables, or sandboxes user-supplied SVG scriptable content, especially as they relate to XSS resulting from inline scripts, and foreignObject.	X	X	X
5.2.8	Verify that the application sanitizes, disables, or sandboxes user-supplied scriptable or expression template language content, such as Markdown, CSS or XSL stylesheets, BBCode, or similar.	X	X	X

Go To References

Output Encoding and Injection Prevention

Go To References

Control	Description	L1	L2	L3
5.3.1	Verify that output encoding is relevant for the interpreter and context required. For example, use encoders specifically for HTML values, HTML attributes, JavaScript, URL Parameters, HTTP headers, SMTP, and others as the context requires, especially from untrusted inputs.	X	X	X
5.3.2	Verify that output encoding preserves the user's chosen character set and locale, such that any Unicode character point is valid and safely handled.	X	X	X
5.3.3	Verify that context-aware, preferably automated - or at worst, manual - output escaping protects against reflected, stored, and DOM based XSS.	X	X	X
5.3.4	Verify that data selection or database queries (e.g. SQL, HQL, ORM, NoSQL) use parameterized queries, ORMs, entity frameworks, or are otherwise protected from database injection attacks.	X	X	X

5.3.5	Verify that where parameterized or safer mechanisms are not present, context-specific output encoding is used to protect against injection attacks, such as the use of SQL escaping to protect against SQL injection.	X	X	X
5.3.6	Verify that the application projects against JavaScript or JSON injection attacks, including for eval attacks, remote JavaScript includes, CSP bypasses, DOM XSS, and JavaScript expression evaluation.	X	X	X
5.3.7	Verify that the application protects against LDAP Injection vulnerabilities, or that specific security controls to prevent LDAP Injection have been implemented.	X	X	X
5.3.8	Verify that the application protects against OS command injection and that operating system calls use parameterized OS queries or use contextual command line output encoding.	X	X	X
5.3.9	Verify that the application protects against Local File Inclusion (LFI) or Remote File Inclusion (RFI) attacks.	X	X	X
5.3.10	Verify that the application protects against XPath injection or XML injection attacks.	X	X	X

Go To References

Memory, String and Unmanaged Code

This section is only applicable when an application uses system language or unmanaged code.

Go To References

Control	*Description*	*L1*	*L2*	*L3*
5.4.1	Verify that the application uses memory-safe string, safer memory copy and pointer arithmetic to detect or prevent stack, buffer, or heap overflows.	X	X	X
5.4.2	Verify that format strings do not take potentially hostile input, and are constant.	X	X	X
5.4.3	Verify that sign, range, and input validation techniques are used to prevent integer overflows.	X	X	X

Go To References

Deserialization Prevention

Go To References

Control	*Description*	*L1*	*L2*	*L3*
5.5.1	Verify that serialized objects use integrity checks or are encrypted to prevent hostile object creation or data tampering.	X	X	X
5.5.2	Verify that the application correctly restricts XML parsers to only use the most restrictive configuration possible and to ensure that unsafe features such as resolving external entities are disabled to prevent XXE.	X	X	X

5.5.3	Verify that deserialization of untrusted data is avoided or is protected in both custom code and third-party libraries (such as JSON, XML and YAML parsers).	X	X	X
5.5.4	Verify that when parsing JSON in browsers or JavaScript-based backends, JSON.parse is used to parse the JSON document. Do not use eval() to parse JSON.	X	X	X

Go To References

6. Stored Cryptography Verification

The application must ensure the high-level requirements are met:

- All cryptographic modules fail in a secure manner and that errors are handles properly.
- A suitable random number generator is used.
- Access to keys is securely managed.

Sample User Stories

Data Classification

Every application should have a privacy impact assessment to classify the data protection needs of any stored data correctly.

Go To References

Control	Description	L1	L2	L3
6.1.1	Verify that regulated private data is stored encrypted while at rest, such as personally identifiable information (PII), sensitive personal information, or data assessed likely to be subject to EU's GDPR.		X	X
6.1.2	Verify that regulated health data is stored encrypted while at rest, such as medical records, medical device details, or de-anonymized research records.		X	X
6.1.3	Verify that regulated financial data is stored encrypted while at rest, such as financial accounts, defaults or credit history, tax records, pay history, beneficiaries, or de-anonymized market or research records.		X	X

Go To References

Algorithms

This section is not easily verified through penetration tests, developers should consider every item L1.

Go To References

Control	Description	L1	L2	L3
6.2.1	Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable Padding Oracle attacks.	X	X	X
6.2.2	Verify that industry proven or government approved cryptographic algorithms, modes, and libraries are used, instead of custom coded cryptography.		X	X
6.2.3	Verify that encryption initialization vector, cipher configuration, and block modes are configured securely using the latest advice.		X	X
6.2.4	Verify that random number, encryption or hashing algorithms, key lengths, rounds, ciphers or modes, can be reconfigured, upgraded, or swapped at any time, to protect against cryptographic breaks.		X	X

6.2.5	Verify that known insecure block modes (i.e. ECB, etc.), padding modes (i.e. PKCS#1 v1.5, etc.), ciphers with small block sizes (i.e. Triple-DES, Blowfish, etc.), and weak hashing algorithms (i.e. MD5, SHA1, etc.) are not used unless required for backwards compatibility.		X	X
6.2.6	Verify that nonces, initialization vectors, and other single use numbers must not be used more than once with a given encryption key. The method of generation must be appropriate for the algorithm being used.			X
6.2.7	Verify that encrypted data is authenticated via signatures, authenticated cipher modes, or HMAC to ensure that ciphertext is not altered by an unauthorized party.			X
6.2.8	Verify that all cryptographic operations are constant-time, with no 'short-circuit' operations in comparisons, calculations, or returns, to avoid leaking information.			

Go To References

Random Values

Go To References

Control	*Description*	*L1*	*L2*	*L3*
6.3.1	Verify that all random numbers, random file names, random GUIDs, and random strings are generated using the cryptographic module's approved cryptographically secure random number generator when these random values are intended to be not guessable by an attacker.		X	X
6.3.2	Verify that random GUIDs are created using the GUID v4 algorithm, and a cryptographically-secure pseudo-random number generator (CSPRNG). GUIDs created using other pseudo-random number generators may be predictable.		X	X
6.3.3	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.			X

Go To References

Secret Management

Go To References

Control	*Description*	*L1*	*L2*	*L3*
6.4.1	Verify that a secrets management solution such as a key vault is used to securely create, store, control access to and destroy secrets.		X	X
6.4.2	Verify that key material is not exposed to the application but instead uses an isolated security module like a vault for cryptographic operations.		X	X

Go To References

7. Error Handling and Logging Verification

Logging and error handling is intended to provide useful information for the users, administrators and security operation teams. The goal is to create high quality logs not high volume.

High quality logs will contain sensitive data and must be protected per policies, regulations and privacy laws. This should include:

- Not collecting or logging sensitive information unless specifically required.
- Ensuring all logged information is handled securely and protected as per its data classification.
- Ensuring that logs are not stored forever but have an absolute lifetime that is as short as possible.
- Ensure logs and errors do not disclose unnecessary information.

Sample User Stories

Log Content

This section covers the OWASP Top 10 2017 list, item A10. This section is not validated through penetration tests, developers should consider the section as L1 and penetration testers should validate through interviews, screenshots or assertions.

Go To References

Control	*Description*	*L1*	*L2*	*L3*
7.1.1	Verify that the application does not log credentials or payment details. Session tokens should only be stored in logs in an irreversible, hashed form.	X	X	X
7.1.2	Verify that the application does not log other sensitive data as defined under local privacy laws or relevant security policy.	X	X	X
7.1.3	Verify that the application logs security relevant events including successful and failed authentication events, access control failures, deserialization failures and input validation failures.		X	X
7.1.4	Verify that each log event includes necessary information that would allow for a detailed investigation of the timeline when an event happens.		X	X

Go To References

Log Processing

Logs should be clear, easily monitored and analyzed either locally or sent to a remote monitoring system like a SIEM.

Go To References

Control	Description	L1	L2	L3
7.2.1	Verify that all authentication decisions are logged, without storing sensitive session identifiers or passwords. This should include requests with relevant metadata needed for security investigations.		X	X
7.2.2	Verify that all access control decisions can be logged and all failed decisions are logged. This should include requests with relevant metadata needed for security investigations.		X	X

Go To References

Log Protection

Logs that can be modified or deleted are useless for investigations and prosecutions. Logs must be protected from unauthorized disclosure, modification or deletion.

Go To References

Control	Description	L1	L2	L3
7.3.1	Verify that the application appropriately encodes user-supplied data to prevent log injection.		X	X
7.3.2	Verify that all events are protected from injection when viewed in log viewing software.		X	X
7.3.3	Verify that security logs are protected from unauthorized access and modification.		X	X
7.3.4	Verify that time sources are synchronized to the correct time and time zone. Strongly consider logging only in UTC if systems are global to assist with post-incident forensic analysis.		X	X

Go To References

Error Handling

Security related events that are logged should be able to be distinguished by a SIEM or analysis software and the log has a purpose.

Go To References

Control	Description	L1	L2	L3
7.4.1	Verify that a generic message is shown when an unexpected or security sensitive error occurs, potentially with a unique ID which support personnel can use to investigate.	X	X	X
7.4.2	Verify that exception handling (or a functional equivalent) is used across the codebase to account for expected and unexpected error conditions.		X	X
7.4.3	Verify that a "last resort" error handler is defined which will catch all unhandled exceptions.		X	X

Go To References

8. Data Protection Verification

The application must ensure the high-level requirements are met:

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

Sample User Stories

General Data Protection

Go To References

Control	Description	L1	L2	L3
8.1.1	Verify the application protects sensitive data from being cached in server components such as load balancers and application caches.		X	X
8.1.2	Verify that all cached or temporary copies of sensitive data stored on the server are protected from unauthorized access or purged/invalidated after the authorized user accesses the sensitive data.		X	X
8.1.3	Verify the application minimizes the number of parameters in a request, such as hidden fields, Ajax variables, cookies and header values.		X	X
8.1.4	Verify the application can detect and alert on abnormal numbers of requests, such as by IP, user, total per hour or day, or whatever makes sense for the application.		X	X
8.1.5	Verify that regular backups of important data are performed and that test restoration of data is performed.		X	X
8.1.6	Verify that backups are stored securely to prevent data from being stolen or corrupted.			X

Go To References

Client-Side Data Protection

Go To References

Control	Description	L1	L2	L3
8.2.1	Verify the application sets sufficient anti-caching headers so that sensitive data is not cached in modern browsers.	X	X	X
8.2.2	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.	X	X	X
8.2.3	Verify that authenticated data is cleared from client storage, such as the browser DOM, after the client or session is terminated.	X	X	X

Go To References

Sensitive Private Data

Privacy regulations and laws directly affect how applications must approach the implementation of storage, use, and transmission of sensitive personal information. This ranges from severe penalties to simple advice. Please consult your local laws and regulations and consult a qualified privacy specialist or lawyer as required.

Go To References

Control	Description	L1	L2	L3
8.3.1	Verify that sensitive data is sent to the server in the HTTP message body or headers, and that query string parameters from any HTTP verb do not contain sensitive data.	X	X	X
8.3.2	Verify that users have a method to remove or export their data on demand.	X	X	X
8.3.3	Verify that users are provided clear language regarding collection and use of supplied personal information and that users have provided opt-in consent for the use of that data before it is used in any way.	X	X	X
8.3.4	Verify that all sensitive data created and processed by the application has been identified, and ensure that a policy is in place on how to deal with sensitive data.	X	X	X
8.3.5	Verify accessing sensitive data is audited (without logging the sensitive data itself), if the data is collected under relevant data protection directives or where logging of access is required.		X	X
8.3.6	Verify that sensitive information contained in memory is overwritten as soon as it is no longer required to mitigate memory dumping attacks, using zeroes or random data.		X	X
8.3.7	Verify that sensitive or private information that is required to be encrypted, is encrypted using approved algorithms that provide both confidentiality and integrity.		X	X
8.3.8	Verify that sensitive personal information is subject to data retention classification, such that old or out of date data is deleted automatically, on a schedule, or as the situation requires.		X	X

Go To References

9. Communications Verification

The application must ensure the high-level requirements are met:

- TLS or strong encryption is always used, regardless of the sensitivity of the data being transmitted.
- Enable and order preferred algorithms and ciphers.
- Weak or soon to be deprecated algorithms and ciphers are ordered as a last resort.
- Deprecated or known insecure algorithms and ciphers are disabled.

Always use the most recent versions of TLS review tools like SSLyze or TLS scanners to configure the preferred order and algorithm selection. These should be checked periodically to ensure the secure communications is current and effective.

Sample User Stories

Communication Security

All communications should use encrypted channels. TLS 1.2 or later is the minimum strength allowed.

Go To References

Control	Description	L1	L2	L3
9.1.1	Verify that secured TLS is used for all client connectivity, and does not fall back to insecure or unencrypted protocols.	X	X	X
9.1.2	Verify using online or up to date TLS testing tools that only strong algorithms, ciphers, and protocols are enabled, with the strongest algorithms and ciphers set as preferred.	X	X	X
9.1.3	Verify that old versions of SSL and TLS protocols, algorithms, ciphers, and configuration are disabled, such as SSLv2, SSLv3, or TLS 1.0 and TLS 1.1. The latest version of TLS should be the preferred cipher suite.	X	X	X

Go To References

Server Communications Security

Server communications are more than HTTP(S). Secure communications to and from other systems such as monitoring systems, management tools, remote access, middleware, databases, directories, or external systems must be in place.

Go To References

Control	Description	L1	L2	L3
9.2.1	Verify that connections to and from the server use trusted TLS certificates. Where internally generated or self-signed certificates are used, the server		X	X

	must be configured to only trust specific internal CAs and specific self-signed certificates. All others should be rejected.			
9.2.2	Verify that encrypted communications such as TLS is used for all inbound and outbound connections, including for management ports, monitoring, authentication, API, or web service calls, database, cloud, serverless, mainframe, external, and partner connections. The server must not fall back to insecure or unencrypted protocols.		X	X
9.2.3	Verify that all encrypted connections to external systems that involve sensitive information or functions are authenticated.		X	X
9.2.4	Verify that proper certification revocation, such as Online Certificate Status Protocol (OCSP) Stapling, is enabled and configured.		X	X
9.2.5	Verify that backend TLS connection failures are logged.		X	X

Go To References

10. Malicious Code Verification

The application must ensure the high-level requirements are met:

- Malicious activity is handled securely and properly to not affect the rest of the application.
- Code does not have time-bombs or other time based attacks.
- Code does not have 'phone home' to malicious, unauthorized or command and control destinations.
- Code does not have any back doors, Easter eggs, rootkits or unauthorized code that can be controlled by an attacker.

Sample User Stories

Code Integrity Controls

Trust but verify. Lead developers should regularly review code check-ins.

Go To References

Control	Description	L1	L2	L3
10.1.1	Verify that a code analysis tool is in use that can detect potentially malicious code, such as time functions, unsafe file operations and network connections.			X

Go To References

Malicious Code Search

This section cannot be fully possible without full access to the source code including 3rd party libraries/components.

Go To References

Control	Description	L1	L2	L3
10.2.1	Verify that the application source code and third party libraries do not contain unauthorized phone home or data collection capabilities. Where such functionality exists, obtain the user's permission for it to operate before collecting any data.		X	X
10.2.2	Verify that the application does not ask for unnecessary or excessive permissions to privacy related features or sensors, such as contacts, cameras, microphones, or location.		X	X
10.2.3	Verify that the application source code and third party libraries do not contain back doors, such as hard-coded or additional undocumented accounts or keys, code obfuscation, undocumented binary blobs, rootkits, or anti-debugging, insecure debugging features, or otherwise out of date, insecure, or hidden functionality that could be used maliciously if discovered.			X
10.2.4	Verify that the application source code and third party libraries does not contain time bombs by searching for date and time related functions.			X

10.2.5	Verify that the application source code and third party libraries does not contain malicious code, such as salami attacks, logic bypasses, or logic bombs.			X
10.2.6	Verify that the application source code and third party libraries do not contain Easter eggs or any other potentially unwanted functionality.			X

Go To References

Deployed Application Security Controls

This section is a continuous operation.

Go To References

Control	*Description*	*L1*	*L2*	*L3*
10.3.1	Verify that if the application has a client or server auto-update feature, updates should be obtained over secure channels and digitally signed. The update code must validate the digital signature of the update before installing or executing the update.	X	X	X
10.3.2	Verify that the application employs integrity protections, such as code signing or sub-resource integrity. The application must not load or execute code from untrusted sources, such as loading includes, modules, plugins, code, or libraries from untrusted sources or the Internet.	X	X	X
10.3.3	Verify that the application has protection from sub-domain takeovers if the application relies upon DNS entries or DNS sub-domains, such as expired domain names, out of date DNS pointers or CNAMEs, expired projects at public source code repos, or transient cloud APIs, serverless functions, or storage buckets (*autogen-bucket-id*.cloud.example.com) or similar. Protections can include ensuring that DNS names used by applications are regularly checked for expiry or change.	X	X	X

Go To References

11. Business Logic Verification

The application must ensure the high-level requirements are met:

- The business logic flow is sequential, processed in order and cannot be bypassed.
- Business logic includes limits to detect and prevent automated attacks.
- 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.

Sample User Stories

Business Logic Security

Business logic security must be designed into the application and cannot be added using web application firewalls or secure communications. Threat modeling during the design sprints will assist in this process.

Go To References

Control	Description	L1	L2	L3
11.1.1	Verify the application will only process business logic flows for the same user in sequential step order and without skipping steps.	X	X	X
11.1.2	Verify the application will only process business logic flows with all steps being processed in realistic human time, i.e. transactions are not submitted too quickly.	X	X	X
11.1.3	Verify the application has appropriate limits for specific business actions or transactions which are correctly enforced on a per user basis.	X	X	X
11.1.4	Verify the application has sufficient anti-automation controls to detect and protect against data exfiltration, excessive business logic requests, excessive file uploads or denial of service attacks.	X	X	X
11.1.5	Verify the application has business logic limits or validation to protect against likely business risks or threats, identified using threat modelling or similar methodologies.	X	X	X
11.1.6	Verify the application does not suffer from "time of check to time of use" (TOCTOU) issues or other race conditions for sensitive operations.		X	X
11.1.7	Verify the application monitors for unusual events or activity from a business logic perspective. For example, attempts to perform actions out of order or actions which a normal user would never attempt.		X	X
11.1.8	Verify the application has configurable alerting when automated attacks or unusual activity is detected.		X	X

Go To References

12. File and Resources Verifications

The application must ensure the high-level requirements are met:

- Untrusted file data should be handled accordingly and in a secure manner.
- Untrusted file data obtained from untrusted sources are stored outside the web root and with limited permissions.

Sample User Stories

File Upload Requirements

Go To References

Control	Description	L1	L2	L3
12.1.1	Verify that the application will not accept large files that could fill up storage or cause a denial of service attack.	X	X	X
12.1.2	Verify that compressed files are checked for "zip bombs" - small input files that will decompress into huge files thus exhausting file storage limits.		X	X
12.1.3	Verify that a file size quota and maximum number of files per user is enforced to ensure that a single user cannot fill up the storage with too many files, or excessively large files.		X	X

Go To References

File Integrity

Go To References

Control	Description	L1	L2	L3
12.2.1	Verify that files obtained from untrusted sources are validated to be of expected type based on the file's content.		X	X

Go To References

File Execution

Go To References

Control	Description	L1	L2	L3
12.3.1	Verify that user-submitted filename metadata is not used directly with system or framework file and URL API to protect against path traversal.	X	X	X
12.3.2	Verify that user-submitted filename metadata is validated or ignored to prevent the disclosure, creation, updating or removal of local files (LFI).	X	X	X
12.3.3	Verify that user-submitted filename metadata is validated or ignored to prevent the disclosure or execution of remote files (RFI), which may also lead to SSRF.	X	X	X

12.3.4	Verify that the application protects against reflective file download (RFD) by validating or ignoring user-submitted filenames in a JSON, JSONP, or URL parameter, the response Content-Type header should be set to text/plain, and the Content-Disposition header should have a fixed filename.	X	X	X
12.3.5	Verify that untrusted file metadata is not used directly with system API or libraries, to protect against OS command injection.	X	X	X
12.3.6	Verify that the application does not include and execute functionality from untrusted sources, such as unverified content distribution networks, JavaScript libraries, node npm libraries, or server-side DLLs.	X	X	X

Go To References

File Storage

Go To References

Control	Description	L1	L2	L3
12.4.1	Verify that files obtained from untrusted sources are stored outside the web root, with limited permissions, preferably with strong validation.	X	X	X
12.4.2	Verify that files obtained from untrusted sources are scanned by antivirus scanners to prevent upload of known malicious content.	X	X	X

Go To References

File Downloads

Go To References

Control	Description	L1	L2	L3
12.5.1	Verify that the web tier is configured to serve only files with specific file extensions to prevent unintentional information and source code leakage. For example, backup files (e.g. .bak), temporary working files (e.g. .swp), compressed files (.zip, .tar.gz, etc) and other extensions commonly used by editors should be blocked unless required.	X	X	X
12.5.2	Verify that direct requests to uploaded files will never be executed as HTML/JavaScript content.	X	X	X

Go To References

SSRF Protection

Go To References

Control	Description	L1	L2	L3
12.6.1	Verify that the web or application server is configured with a whitelist of resources or systems to which the server can send requests or load data/files from.	X	X	X

Go To References

13. API and Web Service Verifications

The application must ensure the high-level requirements are met:

- Adequate authentication, session management and authorization of all web services are in place.
- Input validation of all parameters that transit from a lower to higher trust level.
- Effective security controls for all API types, including cloud and serverless APIs.

Sample User Stories

Generic Web Service Security Verification

Go To References

Control	Description	L1	L2	L3
13.1.1	Verify that all application components use the same encodings and parsers to avoid parsing attacks that exploit different URI or file parsing behavior that could be used in SSRF and RFI attacks.	X	X	X
13.1.2	Verify that access to administration and management functions is limited to authorized administrators.	X	X	X
13.1.3	Verify API URLs do not expose sensitive information, such as the API key, session tokens etc.	X	X	X
13.1.4	Verify that authorization decisions are made at both the URI, enforced by programmatic or declarative security at the controller or router, and at the resource level, enforced by model-based permissions.	X	X	X
13.1.5	Verify that requests containing unexpected or missing content types are rejected with appropriate headers (HTTP response status 406 Unacceptable or 415 Unsupported Media Type).		X	X

Go To References

RESTful Web Service Verification

Go To References

Control	Description	L1	L2	L3
13.2.1	Verify that enabled RESTful HTTP methods are a valid choice for the user or action, such as preventing normal users using DELETE or PUT on protected API or resources.	X	X	X
13.2.2	Verify that JSON schema validation is in place and verified before accepting input.	X	X	X
13.2.3	Verify that RESTful web services that utilize cookies are protected from Cross-Site Request Forgery via the use of at least one or more of the following: triple or double submit cookie pattern (see references), CSRF nonces, or ORIGIN request header checks.	X	X	X

13.2.4	Verify that REST services have anti-automation controls to protect against excessive calls, especially if the API is unauthenticated.	X	X	X
13.2.5	Verify that REST services explicitly check the incoming Content-Type to be the expected one, such as application/xml or application/JSON.		X	X
13.2.6	Verify that the message headers and payload are trustworthy and not modified in transit. Requiring strong encryption for transport (TLS only) may be sufficient in many cases as it provides both confidentiality and integrity protection. Per-message digital signatures can provide additional assurance on top of the transport protections for high-security applications but bring with them additional complexity and risks to weigh against the benefits.		X	X

Go To References

SOAP Web Service Verification

Go To References

Control	*Description*	*L1*	*L2*	*L3*
13.3.1	Verify that XSD schema validation takes place to ensure a properly formed XML document, followed by validation of each input field before any processing of that data takes place.	X	X	X
13.3.2	Verify that the message payload is signed using WS-Security to ensure reliable transport between client and service.		X	X

Go To References

GraphQL and other Web Service Data Layer Security

Go To References

Control	*Description*	*L1*	*L2*	*L3*
13.4.1	Verify that query whitelisting or a combination of depth limiting and amount limiting should be used to prevent GraphQL or data layer expression denial of service (DoS) as a result of expensive, nested queries. For more advanced scenarios, query cost analysis should be used.		X	X
13.4.2	Verify that GraphQL or other data layer authorization logic should be implemented at the business logic layer instead of the GraphQL layer.		X	X

Go To References

14. Configuration Verification

The application must ensure the high-level requirements are met:

- A secure, repeatable, automatable build environment.
- Hardened 3rd party library, dependency and configuration management such that out of date or insecure components are not included in the application.
- A secure-by-default configuration, such that administrators and users have to weaken the default security posture.

Sample User Stories

Build

Build pipelines are the basis for repeatable security. Every time something insecure is discovered it should be resolved in the source code, build or deployment scripts and tested automatically.

This section requires an automated build system and access to build and deployment scripts.

Go To References

Control	Description	L1	L2	L3
14.1.1	Verify that the application build and deployment processes are performed in a secure and repeatable way, such as CI / CD automation, automated configuration management, and automated deployment scripts.	X	X	X
14.1.2	Verify that compiler flags are configured to enable all available buffer overflow protections and warnings, including stack randomization, data execution prevention, and to break the build if an unsafe pointer, memory, format string, integer, or string operations are found.	X	X	X
14.1.3	Verify that server configuration is hardened as per the recommendations of the application server and frameworks in use.	X	X	X
14.1.4	Verify that the application, configuration, and all dependencies can be re-deployed using automated deployment scripts, built from a documented and tested runbook in a reasonable time, or restored from backups in a timely fashion.	X	X	X
14.1.5	Verify that authorized administrators can verify the integrity of all security-relevant configurations to detect tampering.		X	X

Go To References

Dependency

Dependency management is critical to the safe operation of an application. Failure to keep up top date with outdated or insecure dependencies have been the root cause of the largest and most expensive attacks to date.

Go To References

Control	Description	L1	L2	L3
14.2.1	Verify that all components are up to date, preferably using a dependency checker during build or compile time.	X	X	X
14.2.2	Verify that all unneeded features, documentation, samples, configurations are removed, such as sample applications, platform documentation, and default or example users.	X	X	X
14.2.3	Verify that if application assets, such as JavaScript libraries, CSS stylesheets or web fonts, are hosted externally on a content delivery network (CDN) or external provider, Subresource Integrity (SRI) is used to validate the integrity of the asset.	X	X	X
14.2.4	Verify that third party components come from pre-defined, trusted and continually maintained repositories.		X	X
14.2.5	Verify that an inventory catalog is maintained of all third party libraries in use.		X	X
14.2.6	Verify that the attack surface is reduced by sandboxing or encapsulating third party libraries to expose only the required behavior into the application.		X	X

Go To References

Unintended Security Disclosure

Go To References

Control	Description	L1	L2	L3
14.3.1	Verify that web or application server and framework error messages are configured to deliver user actionable, customized responses to eliminate any unintended security disclosures.		X	X
14.3.2	Verify that web or application server and application framework debug modes are disabled in production to eliminate debug features, developer consoles, and unintended security disclosures.		X	X
14.3.3	Verify that the HTTP headers or any part of the HTTP response do not expose detailed version information of system components.		X	X

Go To References

HTTP Security Headers

Go To References

Control	Description	L1	L2	L3
14.4.1	Verify that every HTTP response contains a content type header specifying a safe character set (e.g., UTF-8, ISO 8859-1).	X	X	X
14.4.2	Verify that all API responses contain Content-Disposition: attachment; filename="api.json" (or other appropriate filename for the content type).	X	X	X
14.4.3	Verify that a content security policy (CSPv2) is in place that helps mitigate impact for XSS attacks like HTML, DOM, JSON, and JavaScript injection vulnerabilities.	X	X	X
14.4.4	Verify that all responses contain X-Content-Type-Options: nosniff.	X	X	X
14.4.5	Verify that HTTP Strict Transport Security headers are included on all responses and for all subdomains, such as Strict-Transport-Security: max-age=15724800; includeSubdomains.	X	X	X
14.4.6	Verify that a suitable "Referrer-Policy" header is included, such as "no-referrer" or "same-origin".	X	X	X

Go To References

Validate HTTP Request Header

Go To References

Control	Description	L1	L2	L3
14.5.1	Verify that the application server only accepts the HTTP methods in use by the application or API, including pre-flight OPTIONS.	X	X	X
14.5.2	Verify that the supplied Origin header is not used for authentication or access control decisions, as the Origin header can easily be changed by an attacker.	X	X	X
14.5.3	Verify that the cross-domain resource sharing (CORS) Access-Control-Allow-Origin header uses a strict white-list of trusted domains to match against and does not support the "null" origin.	X	X	X
14.5.4	Verify that HTTP headers added by a trusted proxy or SSO devices, such as a bearer token, are authenticated by the application.	X	X	X
14.5.1	Verify that the application server only accepts the HTTP methods in use by the application or API, including pre-flight OPTIONS.	X	X	X
14.5.2	Verify that the supplied Origin header is not used for authentication or access control decisions, as the Origin header can easily be changed by an attacker.		X	X

Go To References

Sample User Stories

Architecture, Design and Threat Modeling

Return To Section

- As a user, I want to the application to be built using a secure development lifecycle process.
- As a user, I want the application built using threat models.
- As a user, I want the application's security to be verified before I use it.
- As a user, I want the application to only use secure and authenticated communications.
- As a user, I want the application to follow least privilege principals.
- As a user, I want all user input to be validated to prevent injection attacks.
- As a user, I want the application to use current cryptographic processes and secured properly.
- As a user, I want to the application to log appropriate data for records and analysis.
- As a user, I want my sensitive data identified, classified and protected to the appropriate levels.
- As a user, I want the application's source code to be controlled.
- As a user, I want the application to isolate and protect uploaded files.
- As a user, I want the application's configuration to be controlled, consistent and protected.

Authentication Verification

Return To Section

- As a user, I want the application to have strong password policies in place for my account.
- As a user, I want to change my password and be forced to enter my old one first.
- As a user, I want the application to allow passwords longer than 64 characters so I can use phrases.
- As a user, I want to use multi-factor authentication.
- As a user, I do not want the application to perform mutli-factor over text messages (SMS).
- As a user, I want to use my own token generator for multi-factor authentication.
- As a user, I want the application to store my password hashed and salted to the current security standards and practices.
- As a user, I want the application to follow password management, resets, storage and utilization best practices.

Session Management Verification

Return To Section

- As a user, I want the application to use sessions to ensure my use is unique and protected.
- As a user, I want the application to follow security best practices to session use, generation and management.
- As a user, I want the session to time my account out after a period of time of inactivity.
- As a user, I want cookie-based sessions to have all the appropriate security settings set.

- As a user, I want token based sessions to be using digital signatures, encryption and other measures to ensure my session cannot be tampered with.

Access Control Verification

Return To Section

- As a user, I want the application to have access controls in place to ensure I can only access what I need to through least privilege principals,
- As a user, I want any APIs to be protected against direct attacks.
- As a user, I want the application's administrative features to have multi-factor authentication.

Validation, Sanitation and Encoding Verification

Return To Section

- As a user, I want the application to validate, using best practices, all input and output to ensure my data is protected from injection attacks.

Stored Cryptography Verification

Return To Section

- As a user, I want the application to encrypt my sensitive and regulated data while at rest.
- As a user, I want the application to use industry proven or government approves cryptographic algorithms only.
- As a user, I want the application to store encryption keys securely with access tightly controlled.

Error Handling and Logging Verification

Return To Section

- As a user, I want the application to not log sensitive data that would result in my account information or data being breached.
- As a user, I want the application to log security events that would help investigating a potential issue with my account and data.
- As a user, I want the application to log all authentication attempts, successful and unsuccessful, for my account.
- As a user, I want the application to properly secure access to the security logs to prevent my account information and data from being breached.
- As a user, I want the application to show errors with minimal information as to not expose unnecessary information about my account or data.

Data Protection Verification

Return To Section

- As a user, I want the application to protect my sensitive data from being cached or keep temporary copies that may lead to an unintentional exposure.

- As a user, I want the application to have regular backups of my sensitive data and the recovery is tested regularly.
- As a user, I want the application to ensure sensitive data transmitted done securely.

Communications Verification

Return To Section

- As a user, I want the application to use TLS 1.2 or above for all client communications and does not fall back to an unencrypted or insecure, lower state.
- As a user, I want the application to encrypt all communication channels for inbound and outbound connections.
- As a user, I want the application to authenticate all external communication connections where my sensitive data is transmitted.

Malicious Code Verification

Return To Section

- As a user, I want the application to ensure all its source code and 3rd party libraries do not contain malicious capabilities.
- As a user, I want the application to have its source code and components analyzed for malicious capabilities before I use it.

Business Logic Verification

Return To Section

- As a user, I want the application to ensure the business logic is processed, monitored and controlled within the expected utilization.

File and Resources Verifications

Return To Section

- As a user, I want the application to validate any files or data uploaded do not cause any adverse impact to my access.

API and Web Service Verifications

Return To Section

- As a user, I want the application to have adequate authentication, session management and authorization on all web services that have access to my data.

Configuration Verification

Return To Section

- As a user, I want the application to be built in a secure, repeatable and automated way.
- As a user, I want the application to maintain a 3rd party library dependency management process so security issues are in introduced near my data.
- As a user, I want the application to secure the configurations to prevent unauthorized access, modification or other activities that could expose my data.

References

Collection of references applicable to each area of security throughout this document. This is not a definitive list nor is it static. These are the broadest collection that cover a majority of situations you will work on and meant to guide you in the direction not provide all the answers.

All resources listed are free, paid memberships are not included (i.e. ISO 27001).

Authentication, Design and Threat Modeling

Return To Section

Source	Description	Location
NIST	Systems Security Engineering 800-160	PDF Download
NIST	Security Considerations in the System Development Lifecycle 800-64r2	PDF Download
NIST	Recommendation For Key Management 800-57	PDF Download
Microsoft	Azure Architecture Reference Center	Main Site
Microsoft	Threat Modeling Tool	Download Page
Microsoft	Microsoft SDL	Main Site
GIT	Open Architecture for Security and Privacy	OASP Main Site
SAFECode	Practices for Secure Development of Cloud Applications	PDF Download
OWASP	Threat Modeling Cheat Sheet	Main Site
OWASP	Threat Dragon Modeling Tool	Main Site
OWASP	Application Security Architecture Cheat Sheet	Main Site
OWASP	Vulnerable Web Applications Directory	Main Site
NIST	Guidelines for Application Container Security	PDF Download
NIST	Application Container Security Guide 800-190	PDF Download
CVE	Common Vulnerabilities and Exposures	Main Site
CWE	SANS Top 25 Most Dangerous Software Errors	Main Site
SEI	Security Coding Standards for C	C Coding Standards
SEI	Security Coding Standards for C++	C++ Coding Standards
SEI	Security Coding Standards for Java	Java Coding Standards
SEI	Security Coding Standards for Android	Android Coding Standards
SEI	Security Coding Standards for Perl	Perl Coding Standards
SAFECode	Security Stories and Tasks for Agile Development	PDF Download
SAFECode	Managing Risks Inherent In 3rd Party Components	PDF Download
SANS	Checklist For Web Application Design	PDF Download

OWASP	Secure Software Development Lifecycle Cheat Sheet	Main Site
OWASP	OWASP Top 10 (You should know this by heart)	Main Site
OWASP	Secure Coding Cheat Sheet	Main Site
OWASP	Application Security Verification Project	Main Site
OWASP	Testing Guide 4.0	Main Site
OWASP	Mobile Security Testing Guide	Main Site
OWASP/GIT	Security Knowledge Framework (Highly Recommend)	Download Page
OWASP	.NET Security Cheat Sheet	Main Site
OWASP	XML Security Cheat Sheet	Main Site
OWASP	AJAX Security Cheat Sheet	Main Site
OWASP	HTML5 Security Cheat Sheet	Main Site
OWASP	Clickjacking Cheat Sheet	Main Site
OWASP	Unvalidated Redirects and Forwards Cheat Sheet	Main Site
OWASP	Insecure Direct Object Reference Prevention Cheat Sheet	Main Site
Sonarqube	Sonarcube Open Source Static Code Analyzer	Main Site

Return To Section

Authentication Verification

Return To Section

Source	Description	Location
NIST	Digital Identity Guidelines 800-63	PDF Download
NIST	Enrollment and Identity Proofing 800-63A	PDF Download
NIST	Authentication and Lifecycle Management 800-63B	PDF Download
NIST	Federation and Assertion 800-63C	PDF Download
NIST	800-63 FAQ	Main Site
OpenID	OpenID Libraries and Tools (Connected to OAuth 2)	OpenID Libraries
OWASP	Testing Guide 4.0 – Testing for Authentication	Main Site
OWASP	Authentication Cheat Sheet	Main Site
OWASP	SAML Security Cheat Sheet	Main Site
OWASP	Credential Stuffing Prevention Cheat Sheet	Main Site
OWASP	Password Storage Cheat Sheet	Main Site
OWASP	Forgot Password Cheat Sheet	Main Site
OWASP	Choosing and using security questions Cheat Sheet	Main Site
OWASP	Forgot Password Cheat Sheet	Main Site
OWASP	Password Storage Cheat Sheet	Main Site

Return To Section

Session Management Verification

Return To Section

Source	Description	Location
OWASP	Session Management Cheat Sheet	Main Site
OWASP	Testing Guide 4.0 – Testing For Session Management	Main Site
Mozilla	Set-Cookie Host-Prefix Details	Main Site

Return To Section

Access Control Verification

Return To Section

Source	Description	Location
OAuth	OAuth 2 Framework (Also see OpenID Authentication)	Main Site
OWASP	Authorization/Access Control Cheat Sheet	Main Site
OWASP	Transaction Authorization Cheat Sheet	Main Site
OWASP	Testing Guide 4.0 – Testing For Authorization	Main Site
OWASP	CSRF Cheat Sheet	Main Site
OWASP	REST Cheat Sheet	Main Site

Return To Section

Validation, Sanitization and Encoding Verification

Return To Section

Source	Description	Location
OWASP	Input Validation Cheat Sheet	Main Site
OWASP	Deserialization Cheat Sheet	Main Site
OWASP	Code Injection Overview	Main Site
OWASP	Injection Cheat Sheet	Main Site
OWASP	Injection Cheat Sheet With Java	Main Site
OWASP	Cross Site Scripting Prevention Cheat Sheet	Main Site
OWASP	Cross Site Request Forgery (CSRF) Cheat Sheet	Main Site
OWASP	DOM Based Cross Site Scripting Cheat Sheet	Main Site
OWASP	LDAP Injection Cheat Sheet	Main Site
OWASP	Query Parameterization Cheat Sheet	Main Site
OWASP	Mass Assignment Prevention Cheat Sheet	Main Site
OWASP	XML External Entity (XEE) Prevention Cheat Sheet	Main Site
OWASP	Deserialization Cheat Sheet	Main Site
OWASP	Deserialization of Untrusted Data Guide	Main Site
OWASP	SQL Injection Cheat Sheet	Main Site
OWASP	How To Review Code For SQL Injection	Main Site
OWASP	How To Test For SQL Injection	Main Site
OWASP	SQL Injection Bypassing A Web Application Firewall	Main Site

OWASP	Java Encoding Project	Main Site
OWASP	Testing Guide 4.0 – Input Validation Testing	Main Site
OWASP	Testing Guide 4.0 – Testing for HTTP Parameter Pollution	Main Site
OWASP	Testing Guide 4.0 – Client Side Testing	Main Site
AngularJS	Strict Contextual Escaping	Main Site
AngularJS	ngBind	Main Site
AngularJS	Sanitization	Main Site
AngularJS	Template Security	Main Site
Wikipedia	SQL Injection	SQL Injection
Wikipedia	Cross Site Scripting	XSS
Web Site	SQL Injection Knowledge Base (MySQL, MSSQL, Oracle)	Main Site

Return To Section

Stored Cryptography Verification

Return To Section

Source	Description	Location
OWASP	Guide To Cryptography	Main Site
OWASP	Cryptographic Cheat Sheet	Main Site
OWASP	Cryptographic Storage Cheat Sheet	Main Site
OWASP	Key Management Cheat Sheet	Main Site
OWASP	Testing Guide 4.0 – Testing for Weak Cryptography	Main Site
NIST	FIPS 140-2	Main Site
Microsoft	Azure Data At Rest for IaaS VMs	Main Site
Microsoft	Azure Data Security and Encryption Best Practices	Main Site
Microsoft	Bitlocker Documentation	Main Site
Amazon	AWS – Protecting Data Using Encryption	Main Site
Amazon	How To Protect Data At Rest With EC2 Instance	Main Site

Return To Section

Error Handling and Logging Verification

Return To Section

Source	Description	Location
OWASP	Security Logging Project	Main Site
OWASP	Logging Cheat Sheet	Main Site
OWASP	Testing Guide 4.0 – Testing For Error Handling	Main Site

Return To Section

Data Protection Verification

Return To Section

Source	Description	Location
NIST	Introduction to Information Security 800-12	PDF Download
NIST	800-53r4 – The go-to security and privacy reference.	PDF Download
NIST	Managing Information Security Risk 800-39	PDF Download
NIST	For Federal Clients – Protecting Unclassified Data on Non-Federal Systems	PDF Download
PCI	Payment Card Industry Standards Overview	PCI Overview
HIPAA	Health Insurance Portability and Accountability Act – Summarized Report: • Transaction Standards • Identifier Standards • Privacy Rule • Security Rule • Enforcement Rule • Breach Notification Rule	Summary Report
GDPR	General Data Protection Regulation – If you work with EU citizen data, the GDPR applies.	GDPR Main Site
GDPR	GDPR Privacy Impact Assessment	GDPR Privacy Assessment
State of CA	California Consumer Privacy Act of 2018	Link to the law
State of CO	Colorado Privacy Act	Link to the law
CIS	Center For Internet Security Top 20 Controls	Main Site
OWASP	User Privacy Protection Cheat Sheet	Main Site
OWASP	Secure Headers Project	Main Site
OWASP	Privacy Risks Project	Main Site
OWASP	User Privacy Protection Cheat Sheet	Main Site

Return To Section

Communication Verification

Return To Section

Source	Description	Location
OWASP	Transport Layer Protection (TLS) Cheat Sheet	Main Site
OWASP	Web Services Cheat Sheet	Main Site
OWASP	JSON Web Token (JWT) Cheat Sheet	Main Site

Return To Section

Malicious Code Verification

Return To Section

Source	Description	Location
Detectify	Hostile Sub-Domain Takeover Part 1	Main Site
Detectify	Hostile Sub-Domain Takeover Part 2	Main Site

Return To Section

Business Logic Verification

Return To Section

Source	Description	Location
OWASP	Testing Guide 4.0 – Business Logic Testing	Main Site
OWASP	Business Logic Cheat Sheet	Main Site
OWASP	AppSensor Project	Main Site
OWASP	Automated Threats To Web Applications	Main Site
OWASP	Cornucopia	Main Site

Return To Section

File and Resources Verification

Return To Section

Source	Description	Location
OWASP	File Extension Handling For Sensitive Information	Main Site
OWASP	Third party JavaScript Management Cheat Sheet	Main Site
OWASP	Dependency Checker For Open Source Components	Main Site

Return To Section

API and Web Service Verification

Return To Section

Source	Description	Location
OWASP	Serverless Top 10	Main Site
OWASP	Serverless Project	Main Site
OWASP	Testing Guide 4.0 – Configuration and Deployment Management	Main Site

Return To Section

Configuration Verification

Return To Section

Source	Description	Location
NIST	National Security Checklist For IT Products 800-70	PDF Download
NIST	National Security Checklist For IT Products Search	Security Checklist Repository
CIS	100+ Security Benchmarks For Main IT Products	CIS Benchmarks
NIST	Recommendation For Hypervisor Deployments	PDF Download
GIT	DevSecOps Security Hardening Framework	https://github.com/dev-sec
OWASP	Testing Guide 4.0 – Testing for HTTP Verb Tampering	Main Site
OWASP	Content Security Policy Cheat Sheet	Main Site

Return To Section

Security Operations References

Source	Description	Location
NIST	Information Security Handbook: A Guide For Mangers 800-100	PDF Download
NIST	Building A Security Awareness Program	PDF Download
NIST	Guide For Conducting Risk Assessments 800-30	PDF Download
US-CERT	US-Cert Alerts	Main Site
OWASP	OWASP Benchmark for Security Automation	Main Site
OWASP	REST Assessment Cheat Sheet	Main Site
OWASP/GIT	Mobile Security Testing Guide	Main Site

Quick Best Practice Reference Checklist

Based off the SANS SWAT list, The Common Weakness Enumeration numbers can be found here.

Authentication	
Don't hardcode credentials	CWE-798
Develop a strong password reset system	CWE-640
Implement a strong password policy	CWE-521
Implement account lockout against brute force attacks	CWE-307
Don't disclose too much information in error messages	
Store sensitive credentials securely	CWE-257
Applications should run with minimal privileges	CWE-250
Session Management	
Ensure that session identifiers are sufficiently random	CWE-6
Regenerate session tokens	CWE-384
Implement and idle session timeout	CWE-613
Implement and absolute session timeout	CWE-613
Destroy sessions and any sign of tampering	
Invalidate the session after logout	CWE-613
Place a logout button on every page	
User secure cookie attributes	CWE-79 CWE-614
Set the cookie domain and path correctly	
Use non-persistent cookies	
Access Control	
Apple access control checks consistently	CWE-284
Apple the principle of least privilege	CWE-272 CWE-250
Don't use direct object references for access control	CWE-284
Don't use unvalidated forwards or redirects	CWE-601
Input and Output Control	
Conduct contextual output encoding	CWE-79
Use whitelists over blacklists	CWE-159 CWE-144
Use parameterized SQL queries	CWE-89 CWE-564
Prevent Insecure Deserialization	CWE-502
Use tokens to prevent forged requests	CWE-352
Set the encoding for your application	CWE-172
Validate uploaded files	CWE-434 CWE-616

	CWE-22
Use the nosniff header for uploaded content	CWE430
Prevent tabnabbing	CWE-1022
Validate the source of input	CWE-20 CWE-346
X-Frame-Options or CSP headers	CAPEC-103 CWE-693
Use secure HTTP response headers	CWE-79 CWE-692
Data Protection	
Use HTTPS Everywhere	CWE-311 CWE-319 CWE-523
Disable HTTP access for all protected resources	CWE-319
Use strong TLS configurations (v1.3 preferred, v1.2 minimum)	
Use the Strict-Transport-Security header	
Store user passwords using a strong, iterative salted hash	CWE-257
Securely exchange encryption keys	
Set up secure key management processes	CWE-320
Use valid HTTPS certificates from a reputable cert. authority	
Disable data caching using cache control headers	CWE-524
Encrypt sensitive data at rest	CWE-311 CWE-312
Limit the use and storage of sensitive data	
Error Handling and Logging	
Display generic error messages	CWE-209
No unhandled exceptions	CWE-391
Suppress framework-generated errors	CWE-209
Log all authentication and validate activities	CWE-778
Log all privilege changes	CWE-778
Log administrative activities	CWE-778
Log access to sensitive data	CWE-778
Do not log inappropriate data	CWE-532
Store logs securely	CWE-533
Configuration and Operations	
Automate application deployment	
Establish a rigorous change management process	CWE-439
Define security requirements	
Conduct a design review	CWE-701 CWE-656

Perform code reviews	CWE-702
Perform security testing	
Harden the infrastructure	CWE-15 CWE-656
Define and incident response plan	
Educate the team on security	

GIT Cheat Sheet

Repos
Create new local repo
$ git init
Clone existing repo
$ git clone <url>
Commits
Get (pull) changes
$ git pull
Upload (push) changes
$ git push
List files to be committed
$ git status
Show unstaged changes
$ git diff
Show staged changes
$ git diff –staged
Show changes using file diff tool
$ git difftool
Commit (permanently record) staged changes
$ git commit -m “message”
Create a ‘fixup’ commit for older commit
$ git commit –fixup <sha1>
Create a ‘squash’ commit for older commit
$ git commit –squash <sha1>
Show history, one line per commit
$ git log –oneline
Show history for specific file
$ git log –follow <file_path>
Apply an existing commit to current branch
$ git cherry-pick <sha1>
Delete a file and stage deletion
$ git rm <file_name>
Stage the deletion file from the repo, keep it locally
$ git rm –cached <file_name>
Stage file moving or renaming
$ git mv <old_filepath> <new_filepath>
Stash (temporarily) uncommitted changeset
$ git stash
Restore most recent stashed changeset
$ git stash pop
List all stashed changesets
$ git stash list
Edit last commit with new staged changes
$ git commit –amend
Undo (discard) last commit
$ git reset HEAD-

Branches
Create new branch and switch to it
$ git checkout -b <new_branch_name>
Create new branch, stay in the current one
$ git branch <new_branch_name>
Rename a branch
$ git branch -m <old_name> <new_name>
Delete a branch
$ git branch -d <old_branch>
List all local branches
$ git branch
Print name of current branch
$ git rev-parse --abbrev-ref HEAD
Merge other branch into current one
$ git merge <other_branch>
Show history for current branch
$ git log
Show history, with merged branch graph
$ git log --graph
Perform interactive rebase
$ git rebase -i <other_branch>
Perform interactive rebase with autosquash
$ git rebase -i –autosquash <other_branch>
Upload new branch to origin remote
$ git push -u origin <new_branch>
Create new tag
$ git tag -a <tag_name>
List tags
$ git tag
Misc
Delete all untracked files and directories
$ git clean -fd
Launch GitWeb (requires supported webserver)
$ git instaweb
Stop GitWeb
$ git instaweb stop
Launch official git GUI
$ gitk
Undo (discard) last 3 commits
$ git reset HEAD- 3
Revert commit by creating opposite commit
$ git revert <sha1>
Show specific commit (contents and metadata)
$ git show <sha1>
Discard uncommitted changes
$ git reset –hard

Security In Five is a daily podcast produces by a security professional with over 20 years of experience. The show discusses cybersecurity and IT related topics through daily episodes that are roughly five minutes long. Quick to listen to, easy to understand.

This document is a collection of freely available information. In some cases wording from the original source has been omitted or modified to the author's editorial decisions. All links to the original materials are noted in throughout this document. The author does not attest to the accuracy of referenced materials and encourages readers to always trust but verify and confirm using multiple sources and always check for updates.

Any feedback, questions or corrections can be sent to bblogger@protonmail.com.

Be aware, be safe.

Security In Five podcast can be found in all the major podcast locations or directly at https://securityinfive.libsyn.com and on the website https://securityinfive.com.

Twitter: @SecurityInFive

Patreon: https://www.patreon.com/SecurityInFive