# 



# Application Security Verification Standard 4.0

## DRAFT

November 2018

# Frontispiece

## About the Standard

The Application Security Verification Standard is a list of application security requirements or tests that can be used by architects, developers, testers, security professionals, and even consumers to define what a secure application is.

## Copyright and License

  
Copyright © 2018 The OWASP Foundation. This document is released under the Creative Commons Attribution ShareAlike 3.0 license. For any reuse or distribution, you must make clear to others the license terms of this work.

## Draft Version

This is a draft version of the standard, and as such should not be used until released. If you find any issues within the standard that should be addressed:

* Design of the standard.
* Missing controls.
* Ineffective or outdated controls.
* Unclear wording, spelling, grammar issues.
* Formatting issues.
* Translation issues – if a control’s wording is such that trying express it in your language will be difficult or impossible, please let us know. If it doesn’t work in Spanish or Thai, it probably isn’t working in English either.
* Offers of translation – please let us know so that we can direct you to folks already working on your language.

Please log an issue here: <https://github.com/OWASP/ASVS/issues>

Version 4.0, 2018

## Project Leads

Andrew van der Stock  
Daniel Cuthbert  
Jim Manico

## Contributors and Reviewers

* Find from closed bugs on Github

# Preface

Welcome to the Application Security Verification Standard (ASVS) version 4.0. The ASVS is a community-effort to establish a framework of security requirements and controls that focus on normalizing the functional and non-functional security controls required when designing, developing and testing modern web applications.

ASVS v4.0 is a culmination of community effort and industry feedback. In this release, we felt it was important to qualify the experiences of real world use cases relating to ASVS adoption. This will help newcomers to the standard plan their adoption of the ASVS, whilst assisting existing companies in learning from the experience of others.

We expect that there will most likely never be 100% agreement on this standard. Risk analysis is always subjective to some extent, which creates a challenge when attempting to generalize in a one size fits all standard. However, we hope that the latest updates made in this version are a step in the right direction, and respectfully enhance the concepts introduced in this important industry standard.

## What’s new in 4.0

The single largest change in this version is the adoption of NIST 800-63, concentrating on introducing controls at authentication assurance levels 1 through 3. Although we do expect there to be some pushback on this adoption, it's important for standards to be aligned, particularly when a newer standard is evidence based.

We have worked to comprehensively meet and exceed OWASP Top 10 2017 requirements. As the OWASP Top 10 2017 is the bare minimum to avoid negligence, we have deliberately set nearly all of OWASP Top 10 controls to be L1. This makes it easier for OWASP Top 10 adopters to continuously improve, especially when they want to continue the security journey by building in security into every application and API.

We have completed the shift of the ASVS from monolithic server side only controls, to providing solid controls for all modern applications and APIs. In the days of functional programming, server less API, mobile, cloud, IoT, CI/CD and DevOps, and federation, we cannot keep on ignoring modern application architecture. Modern applications are designed differently to those in 2008, when the original ASVS was written. The ASVS must always look far into the future so that we provide sound advice for our primary audience - developers. We have clarified or dropped any requirement that assumes that security must be solely performed on a server owned by a single organization.

We have retired the mobile section, in favor of the Mobile Application Security Verification Standard (MASVS) and the Internet of Things section in favour of the OWASP Internet of Things Projects. Our aim is to support them as much as possible, by being a reference where needed. As that standard does not particularly strongly elucidate API security issues, we have decided to concentrate on being the best standard for web apps and all forms of API - RESTful and SOAP web services, monolithic or micro services. Mobile application developers should see the MASVS as being highly useful for mobile client issues, and the ASVS as being highly useful for mobile APIs and micro services.

Lastly, we have de-duped and retired controls that cannot get you hacked. Over time, the ASVS started being a comprehensive set of controls, but not all controls are equal at producing secure software. We have re-validated L1 to be a "Core" or "Essential" control set, by setting a CVSS score bar of 7.5 or above to be in L1. This helps compliant organizations adhere to PCI DSS's requirements to resolve all High and Critical findings rapidly. If an application or API has a failed L1 control, it is a serious risk to any organization.

# Using the ASVS

ASVS has two main goals:

* to help organizations develop and maintain secure applications.
* to allow security service, security tools vendors, and consumers to align their requirements and offerings.

Application Security Verification Levels  
The Application Security Verification Standard defines three security verification levels, with each level increasing in depth.

* ASVS Level 1 is meant for all software.
* ASVS Level 2 is for applications that contain sensitive data, which requires protection.
* ASVS Level 3 is for the most critical applications - applications that perform high value transactions, contain sensitive medical data, or any application that requires the highest level of trust.

Each ASVS level contains a list of security requirements. Each of these requirements can also be mapped to security-specific features and capabilities that must be built into software by developers.

Figure 1 - OWASP Application Security Verification Standard 4.0 Levels

## How to use this standard

One of the best ways to use the Application Security Verification Standard is to use it as blueprint create a Secure Coding Checklist specific to your application, platform or organization. Tailoring the ASVS to your use cases will increase the focus on the security requirements that are most important to your projects and environments.

### Level 1: Opportunistic

An application achieves ASVS Level 1 (or Opportunistic) if it adequately defends against application security vulnerabilities that are easy to discover, and included in the OWASP Top 10 and other similar checklists.

Level 1 is typically appropriate for applications where low confidence in the correct use of security controls is required, or to provide a quick analysis of a fleet of enterprise applications, or assisting in developing a prioritized list of security requirements as part of a multi-phase effort. Level 1 controls can be ensured either automatically by tools or simply manually without access to source code. We consider Level 1 the minimum required for all applications.

Threats to the application will most likely be from attackers who are using simple and low effort techniques to identify easy-to-find and easy-to-exploit vulnerabilities. This is in contrast to a determined attacker who will spend focused energy to specifically target the application. If data processed by your application has high value, you would rarely want to stop at a Level 1 review.

### Level 2: Standard

An application achieves ASVS Level 2 (or Standard) if it adequately defends against most of the risks associated with software today.

Level 2 ensures that security controls are in place, effective, and used within the application. Level 2 is typically appropriate for applications that handle significant business-to-business transactions, including those that process healthcare information, implement business-critical or sensitive functions, or process other sensitive assets.

Threats to Level 2 applications will typically be skilled and motivated attackers focusing on specific targets using tools and techniques that are highly practiced and effective at discovering and exploiting weaknesses within applications.

### Level 3: Advanced

ASVS Level 3 is the highest level of verification within the ASVS. This level is typically reserved for applications that require significant levels of security verification, such as those that may be found within areas of military, health and safety, critical infrastructure, etc.

Organizations may require ASVS Level 3 for applications that perform critical functions, where failure could significantly impact the organization's operations, and even its survivability. Example guidance on the application of ASVS Level 3 is provided below. An application achieves ASVS Level 3 (or Advanced) if it adequately defends against advanced application security vulnerabilities and also demonstrates principles of good security design.

An application at ASVS Level 3 requires more in depth analysis, architecture, coding, and testing than all the other levels. A secure application is modularized in a meaningful way (to facilitate e.g. resiliency, scalability, and most of all, layers of security), and each module (separated by network connection and/or physical instance) takes care of its own security responsibilities (defense in depth), that need to be properly documented. Responsibilities include controls for ensuring confidentiality (e.g. encryption), integrity (e.g. transactions, input validation), availability (e.g. handling load gracefully), authentication (including between systems), non-repudiation, authorization, and auditing (logging).

## Applying ASVS in Practice

Different threats have different motivations. Some industries have unique information and technology assets and domain specific regulatory compliance requirements.

Below we provide industry-specific guidance regarding recommended ASVS levels. Although some unique criteria and some differences in threats exist for each industry, a common theme throughout all industry segments is that opportunistic attackers will look for any easily exploitable vulnerable applications, which is why ASVS Level 1 is recommended for all applications regardless of industry. This is a suggested starting point to manage the easiest to find risks.

Organizations are strongly encouraged to look more deeply at their unique risk characteristics based on the nature of their business. At the other end of the spectrum is ASVS Level 3, which is reserved for those cases that might endanger human safety or when a full application breach could severely impact the organization.

TBD - Need to re-do case studies / applicability.

# Assessment and Certification

## OWASP's Stance on ASVS Certifications and Trust Marks

OWASP, as a vendor-neutral not-for-profit organization, does not currently certify any vendors, verifiers or software.

All such assurance assertions, trust marks, or certifications are not officially vetted, registered, or certified by OWASP, so an organization relying upon such a view needs to be cautious of the trust placed in any third party or trust mark claiming ASVS certification.

This should not inhibit organizations from offering such assurance services, as long as they do not claim official OWASP certification.

## Guidance for Certifying Organizations

The Application Security Verification Standard can be used as an open book verification of the application, including open and unfettered access to key resources such as architects and developers, project documentation, source code, authenticated access to test systems (including access to at least one account in each role), particularly for L2 and L3 verifications.

Historically, penetration testing and secure code reviews have included issues “by exception” - that is only failed issues appear in the final report. A certifying organization must include in any report the scope of the verification (particularly if a key component is out of scope, such as SSO authentication), a summary of verification findings, including passed and failed tests, with clear indications of how to resolve the failed tests.

Keeping detailed work papers, screenshots or movies, scripts to reliably and repeatedly exploit an issue, and electronic records of testing, such as intercepting proxy logs and associated notes such as a cleanup list, is considered standard industry practice and can be really useful as proofs of the findings for the most doubts developers. It is not sufficient to simply run a tool and report on the failures; this does not (at all) provide sufficient evidence that all issues at a certifying level have been tested and tested thoroughly. In case of dispute, there should be sufficient assurance evidence to demonstrate each and every verified requirement has indeed been tested.

### The Role of Automated Security Testing Tools

Automated penetration tools are encouraged to provide as much as possible coverage and to exercise as many parameters as possible with many different forms of malicious inputs as possible.

It is not possible to fully complete ASVS verification using automated penetration testing tools alone. Whilst a large majority of requirements in L1 can be performed using automated tests, the overall majority of requirements are not amenable to automated penetration testing.

Please note that the lines between automated and manual testing have blurred as the application security industry matures. Automated tools are often manually tuned by experts and manual testers often leverage a wide variety of automated tools.

## The Role of Penetration Testing

It is possible to perform a manual penetration test and verify all L1 issues without requiring access to source code, but this is not a leading practice. L2 requires at least some access to developers, documentation, code, and authenticated access to the system. Complete penetration testing coverage at Level 3 is not possible, as most of the additional issues involve review of system configuration, malicious code review, threat modeling, and other non-penetration testing artifacts.

### As Detailed Security Architecture Guidance

One of the more common uses for the Application Security Verification Standard is as a resource for security architects. The two major security architecture frameworks, SABSA or TOGAF, are missing a great deal of information that is necessary to complete application security architecture review. ASVS can be used to fill in those gaps by allowing security architects to choose better controls for common problems, such as data protection patterns and input validation strategies.

### As a Replacement for Off-the-shelf Secure Coding Checklists

Many organizations can benefit from adopting the ASVS, by choosing one of the three levels, or by forking ASVS and changing what is required for each application risk level in a domain specific way. We encourage this type of forking as long as traceability is maintained, so that if an app has passed requirement 4.1, this means the same thing for forked copies as the standard as it evolves.

### As a Guide for Automated Unit and Integration Tests

The ASVS is designed to highly testable, with the sole exception of architectural and malicious code requirements. By building unit and integration tests that test for specific and relevant fuzz and abuse cases, the application becomes nearly self-verifying with each and every build. For example, additional tests can be crafted for the test suite for a login controller, testing the username parameter for common usernames, account enumeration, brute forcing, LDAP and SQL injection, and XSS. Similarly, a test on the password parameter should include common passwords, password length, null byte injection, removing the parameter, XSS, account enumeration, and more.

### For Secure Development Training

ASVS can also be used to define characteristics of secure software. Many “secure coding” courses are simply ethical hacking courses with a light smear of coding tips. This does not help developers. Instead, secure development courses can use the ASVS with a strong focus on the proactive controls found in the ASVS, rather than the Top 10 negative things not to do.

# V1: Architecture, Design and Threat Modeling Requirements

## Control Objective

To build secure software, we need to consider security throughout all development phases. However, in reality, security is often only an afterthought in the SDLC.

The ASVS remediates this problem by not only focusing on technical controls. It also requires the explicit adoption of security processes. These processes ensure that security is taken into account as soon as the first architecture is drawn. Additionally, the ASVS requires specifying both functional and security properties for application components.

Note that distributed applications cannot be tested in isolation. They need to be tested in their distributed setup. Prime examples of such applications are mobile clients or Single Page Applications (SPAs) that interact with an API. Examining such a client without taking the API into account will yield incomplete results.

Category “V1” lists requirements related to the architecture and design of the application. Unlike other requirements in the ASVS, these requirements do not map to technical test cases in the OWASP Testing Guide. The resources at the end of this category include related OWASP projects that do cover these requirements. Example requirements are threat modeling, a secure SDLC, cryptographic key management.

## Security Verification Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | Since |
| **1.1** | All app components are identified and known to be needed. | ✓ | ✓ | ✓ | 1.0 |
| **1.2** | Security controls are never enforced only on the client side, but on the respective remote endpoints. |  | ✓ | ✓ | 1.0 |
| **1.3** | A high-level architecture for the application and all connected remote services has been defined and security has been addressed in that architecture. |  | ✓ | ✓ | 1.0 |
| **1.4** | Data considered sensitive in the context of the application is clearly identified. |  |  | ✓ | 1.0 |
| **1.5** | All app components are defined in terms of the business functions and/or security functions they provide. |  |  | ✓ | 1.0 |
| **1.6** | A rudimentary threat analysis has been made to determine which attackers are in scope, and which are currently not in scope (e.g. the internal network is considered safe). |  | ✓ | ✓ | 4.0 |
| **1.7** | A detailed threat model for the application and the associated remote services has been produced that identifies potential threats and countermeasures. |  |  | ✓ | 1.0 |
| **1.8** | All security controls have a centralized implementation as to avoid duplication of critical code. |  | ✓ | ✓ | 3.0 |
| **1.9** | Components are segregated from each other via a defined security control, such as network segmentation, firewall rules, or cloud based security groups. |  | ✓ | ✓ | 3.0 |
| **1.10** | A mechanism for enforcing updates of the application exists. |  | ✓ | ✓ | 3.0 |
| **1.11** | Security is addressed within all parts of the software development lifecycle. |  | ✓ | ✓ | 3.0 |
| **1.12** | All application components, libraries, modules, frameworks, platform, and operating systems are free from known vulnerabilities. |  | ✓ | ✓ | 3.0.1 |
| **1.13** | There is an explicit policy for how cryptographic keys (if any) are managed, and the lifecycle of cryptographic keys is enforced. Ideally, follow a key management standard such as NIST SP 800-57. |  | ✓ | ✓ | 4.0 |
| **1.14** | Validate that there are security requirements that can be used as an input to test the undesired behaviour of the application. |  |  | ✓ | 4.0 |
|  |  |  |  |  |  |

## References

For more information, see also:

For more information, please see:

* [OWASP Threat Modeling Cheat Sheet](https://www.owasp.org/index.php/Application_Security_Architecture_Cheat_Sheet)
* [OWASP Attack Surface Analysis Cheat Sheet](https://www.owasp.org/index.php/Attack_Surface_Analysis_Cheat_Sheet)
* [OWASP Security Architecture Cheat Sheet](https://www.owasp.org/index.php/Application_Security_Architecture_Cheat_Sheet)
* [OWASP Threat modelling](https://www.owasp.org/index.php/Application_Threat_Modeling)
* [OWASP Secure SDLC Cheat Sheet](https://www.owasp.org/index.php/Secure_SDLC_Cheat_Sheet)
* [Microsoft SDL](https://www.microsoft.com/en-us/sdl/)
* [NIST SP 800-57](http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf)

# V2: Authentication Verification Requirements

## Control Objective

Authentication is the act of establishing, or confirming, something (or someone) as authentic, that is, that claims made by a person or about a device are correct, resistant to impersonation (phishing), and prevents recovery or interception of memorized secrets (passwords).

ASVS V2 Authentication, V3 Session Management, and V4 Access Controls have been adapted to be a compliant subset of selected NIST 800-63 controls, focused around common threats and commonly exploited authentication weaknesses. Previous verification requirements have been retired, de-duped, or in most cases adapted to either be identical or strongly aligned with the intent of NIST 800-63 normative (mandatory) requirements. Logging requirements are now located the V8 Error and Logging chapter, and TLS requirements are now found in the V10 Communications Chapter.

NIST 800-63 is a modern, evidence-based standard, and represents the best advice available, regardless of applicability. The standard is helpful for all organizations all over the world but is particularly relevant to US agencies and those dealing with US agencies.

Implementers requiring the full set of controls should review the entire standard, especially regarding evidence of identity, identity binding, identity assertion, the deployment and management of multi-factor, biometric and crypto devices, security usability, and a great deal more advanced topics. Full compliance with the ASVS 4.0 is not the same as full compliance with NIST 800-63.

NB: We use the term "password" when NIST uses "memorized secret" throughout this standard.

### Selecting an appropriate NIST AAL Level

The Application Security Verification Standard has mapped ASVS L1 to AAL1 requirements, L2 to AAL2, and L3 to AAL3. However, the approach of ASVS Level 1 as "essential" controls may not necessarily be the correct AAL level to verify an application or API. For example, if the application is a Level 3 application or has regulatory requirements to be AAL3, Level 3 should be chosen in Sections V2 and V3 Session Management.

The choice of NIST compliant authentication assertion level (AAL) should be performed as per NIST 800-63 guidelines as set out in *Selecting AAL* in [NIST 800-63 Section 6.2](https://pages.nist.gov/800-63-3/sp800-63-3.html#AAL_CYOA).

We strongly urge everyone to adopt NIST 800-63, and align any policies, guidelines and standards with it, such as we've done here.

## Authentication Verification Requirements

Legend

Applications can always exceed the current level's requirements. The following keys are used throughout this standard:

No check tick or "o" - not required at this level

"o" - recommended, but not required

"✓" - required

### V2.1 General Authenticator Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | NIST § |
| 2.1.1 | Verify revocability of physical authenticators in case of theft or other loss. Ensure that revocation is immediately effective across all Identity Providers and Relying Parties. | o | ✓ | ✓ | 5.2.1 |
| 2.1.2 | Verify that one or more anti-automation controls--including rate limiting, CAPTCHA, increasing delays, IP address restrictions, risk-based restrictions--are in place and effective to mitigate breached credential testing, brute force, and account lockout attacks. Verify that no more than 100 failed attempts is possible on a single account. | ✓ | ✓ | ✓ | 5.2.2 / 5.1.1.2 |
| 2.1.3 | Verify that biometric authenticators are limited to use only as secondary factors in conjunction with either something you have and something you know. |  | o | ✓ | 5.2.3 |
| 2.1.5 | 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. |  | o | ✓ | 5.2.5 |
| 2.1.6 | Verify that in cases where a verifier and CSP are separate, mutually authenticated TLS is in place between the two endpoints. |  | o | ✓ | 5.2.6 |
| 2.1.8 | Verify replay resistance through the mandated use of OTP devices, cryptographic authenticators, or lookup codes. |  | o | ✓ | 5.2.8 |
| 2.1.9 | 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. |  | o | ✓ | 5.2.9 |
| 2.1.10 | Verify that restricted authenticators--such as email and SMS--are not a preferred recovery mechanism or second factor, and at least one alternative is offered to the user first. If the user selects a restricted authenticator, a meaningful warning covering the potential risks of that restricted authenticator SHOULD be presented to the user, including that the future use of the restricted authenticator may be removed in the future. | ✓ | ✓ | ✓ | 5.2.10 |

NIST considers email and SMS as "restricted" plain text authentication channels. As restricted authenticators, they will be removed from NIST 800-63 and thus the ASVS at some point the future. Authenticator agility is essential to future-proof applications. Refactor application verifiers to allow additional authenticators as per user preferences, as well as allowing retiring deprecated or unsafe authenticators in an orderly fashion.

### V2.2 Authenticator Lifecycle Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | NIST § |
| 2.2.1 | 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. | ✓ | ✓ | ✓ | 5.1.1.2 / A.3 |
| 2.2.2 | Verify that enrollment and use of subscriber-provided authentication devices are supported, such as a U2F or FIDO tokens. | ✓ | ✓ |  | 6.1.3 |
| 2.2.3 | Verify that renewal instructions are sent with sufficient time to renew time bound authenticators. | ✓ | ✓ | ✓ | 6.1.4 |
| 2.2.2 | Verify that enrollment and use of subscriber-provided authentication devices are supported, such as a U2F or FIDO tokens. | ✓ | ✓ |  | 6.1.3 |

There are additional requirements in section 6.2 for US agencies that are out of scope for the ASVS.

### V2.3 Authenticator Recovery Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | NIST § |
| 2.3.1 | Verify that a system generated activation or recovery password is not sent in clear text to the user. | ✓ | ✓ | ✓ | 5.1.1.2 |
| 2.3.2 | Verify password hints or knowledge-based answers (so-called "secret questions") are not present. | ✓ | ✓ | ✓ | 5.1.1.2 |
| 2.3.3 | Verify password credential recovery does not reveal the current password in any way. | ✓ | ✓ | ✓ | 5.1.1.2 |
| 2.3.4 | Verify forgotten password, and other recovery paths use a TOTP or other soft token, mobile push, or another offline recovery mechanism. | ✓ | ✓ | ✓ | 5.1.1.2 |
| 2.3.5 | Verify identities cannot be re-bound to a different identity (spoofing), and shared accounts are not present ("root", "admin", or "sa"). | ✓ | ✓ | ✓ | 5.1.1.2 / A.3 |
| 2.3.6 | Verify that if one or more multi-factor authentication factors are lost, that identity proofing and binding is performed at the same level as during enrollment. Once replaced, the verifier MAY use a single factor to re-bind the account to the new factor. | ✓ | ✓ | ✓ | 6.1.2.3 |
| 2.3.7 | Verify that if an authentication factor is changed or replaced, that the user is notified of this event. | ✓ | ✓ | ✓ | 6.1.2.3 |

The use of email or PSTN verifiers (such as SMS tokens) has been deprecated by NIST and SHOULD NOT be used.

### V2.4 Memorized Secrets Authenticator Requirements

Memorized secrets are passwords, PINs, unlock patterns, pick the correct kitten or another image element, and passphrases. They are generally considered "something you know", and often used as single factor authenticators. There are significant challenges to the continued use of single-factor authentication, including billions of valid usernames and passwords disclosed on the Internet, default or weak passwords, rainbow tables and ordered dictionaries of the most common passwords.

Memorized secrets are not sufficient to protect against today's threats. The ASVS has long required multi-factor authentication. We have not relaxed this point of view. NIST 800-63 recommends but does not require multi-factor authentication at IAL1/AAL1. The ASVS Leadership Team strongly feels that it is essential to start the process of universal multi-factor authentication, as it defeats the most common attacks against passwords. Therefore, we have set multi-factor authentication as a baseline L1 requirement.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | NIST § |
| 2.4.1 | Verify that passwords are at least 8 characters in length. | ✓ | ✓ | ✓ | 5.1.1.2 |
| 2.4.2 | Verify that passwords 64 characters or longer are permitted. | ✓ | ✓ | ✓ | 5.1.1.2 |
| 2.4.3 | Verify that passwords can contain spaces and truncation is not performed. Consecutive multiple spaces MAY optionally be coalesced. | ✓ | ✓ | ✓ | 5.1.1.2 |
| 2.4.4 | Verify that Unicode characters are permitted in passwords. A single Unicode code point is considered a character, so 8 emoji or 64 kanji characters should be valid and permitted. | ✓ | ✓ | ✓ | 5.1.1.2 |
| 2.4.5 | Verify users can change their password, and the change validates the current secret. | ✓ | ✓ | ✓ | 5.1.1.2 |
| 2.4.6 | Verify that new or changed passwords are validated against a list of compromised secrets, and if found to be compromised, the user is prompted to choose another secret. | ✓ | ✓ | ✓ | 5.1.1.2 |
| 2.4.7 | Verify that a password strength meter is provided to help users set a stronger secret. | ✓ | ✓ | ✓ | 5.1.1.2 |
| 2.4.8 | 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. | ✓ | ✓ | ✓ | 5.1.1.2 |
| 2.4.9 | Verify that there are no arbitrary or periodic credential rotation requirements. | ✓ | ✓ | ✓ | 5.1.1.2 |
| 2.4.10 | Verify that the user is required to change their password if the credential has found to be compromised. | ✓ | ✓ | ✓ | 5.1.1.2 |
| 2.4.11 | Verify that "paste" functionality, browser password helpers, and external password managers are permitted. | ✓ | ✓ | ✓ | 5.1.1.2 |
| 2.4.12 | Verify that the user can choose to either temporarily view the masked entered password, or temporarily view the last typed character of the password. | ✓ | ✓ | ✓ | 5.1.1.2 |

### V2.5 Credential Storage Requirements

Architects and developers should adhere to this section when building or refactoring code. This section can only be fully verified using source code review or through secure unit or integration tests. Penetration testing cannot identify any of these issues.

The list of approved one-way key derivation functions is detailed in NIST 800-63 B section 5.1.1.2, and in [BSI Kryptographische Verfahren: Empfehlungen und Schlussellängen (2018)](https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile). The latest national or regional algorithm and key length standards can be chosen in place of these choices.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | NIST § |
| 2.5.1 | 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, and a cost factor as inputs then generate a password hash. | ✓ | ✓ | ✓ | 5.1.1.2 |
| 2.5.2 | 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. | ✓ | ✓ | ✓ | 5.1.1.2 |
| 2.5.3 | Verify that if PBKDF2 is used, the iteration count SHOULD be as large as verification server performance will allow, typically at least 10,000 iterations. | ✓ | ✓ | ✓ | 5.1.1.2 |
| 2.5.4 | Verify that if bcrypt is used, the work factor SHOULD be as large as verification server performance will allow, typically at least 13. | ✓ | ✓ | ✓ |  |
| 2.5.5 | Verify that an additional iteration of a key derivation function using a salt value that is secret and known only to the verifier. This salt value, if used, SHALL be generated by 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). | o | ✓ | ✓ | 5.1.1.2 |
| 2.5.6 | Allow users to set a passphrase as their password | o |  | ✓ |  |
| 2.5.7 | Recommend customers to use a password manager to store credentials securely | o |  | ✓ |  |

### V2.6 Look-up Secret Verifier Requirements

Look up secrets are pre-generated lists of secret codes, similar to Transaction Authorization Numbers (TAN), Google Recovery Codes, or a grid containing a set of random values. These are distributed securely to users. These lookup codes are used once, and once all used, the lookup secret list is discarded. This type of authenticator is considered "something you have".

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | NIST § |
| 2.6.1 | Verify that lookup secrets can be used only once. | o | ✓ | ✓ | 5.1.2.2 |
| 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. | o | ✓ | ✓ | 5.1.2.2 |
| 2.6.3 | Verify that lookup secrets are resistant to offline attacks, such as predictable values. | o | ✓ | ✓ | 5.1.2.2 |
| 2.6.4 | Verify that lookup secrets are rate limited, particularly when using less than 64 bits of entropy. | o | ✓ | ✓ | 5.1.2.2 |

### V2.7 Out of Band Verifier Requirements

Out of band authenticators are physical devices that can communicate with the verifier over a secure secondary channel. Examples include push notifications to mobile devices. This type of authenticator is considered "something you have". When a user wishes to authenticate, the verifying application sends a message to the out of band authenticator via a connection to the authenticator directly or indirectly through a third party service. The message contains an authentication key (typically a random six digit number or a modal approval dialog). The verifying application waits to receive the authentication key through the primary channel and compares the hash of the received value to the hash of the original authentication key. If they match, the out of band verifier can assume that the user has authenticated.

The ASVS assumes that only a few developers will be developing new out of band authenticators, such as push notifications, and thus the following ASVS controls apply to verifiers, such as authentication API, applications, and single sign-on implementations. If developing a new out of band authenticator, please refer to NIST 800-63B § 5.1.3.1.

Unsafe out of band authenticators such as e-mail and VOIP is not permitted. PSTN and SMS authentication are currently restricted and should be deprecated in favor of push notifications or similar. If you need to use telephone or SMS out of band authentication, please see § 5.1.3.3.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | NIST § |
| 2.7.1 | Verify that prohibited unencrypted out of band authenticators, such as e-mail or VOIP, is not in use. | ✓ | ✓ | ✓ | 5.1.3.1 |
| 2.7.2 | Verify that restricted out of band authenticators, such as SMS or PSTN, are not offered by default, and stronger alternatives such as push notifications are offered first. | ✓ | ✓ | ✓ | 5.1.3.2 |
| 2.7.3 | Verify that the out of band authenticator and verifier communicates over a secure independent channel. | ✓ | ✓ | ✓ | 5.1.3.2 |
| 2.7.4 | Verify that the out of band verifier retains only a hashed version of the authentication key. | ✓ | ✓ | ✓ | 5.1.3.2 |
| 2.7.5 | Verify that the out of band verifier rejects out of band authentication attempts after 10 minutes. | ✓ | ✓ | ✓ | 5.1.3.2 |
| 2.7.6 | Verify that the out of band verifier authentication keys is only usable once, and only for the original authentication request. | ✓ | ✓ | ✓ | 5.1.3.2 |
| 2.7.7 | Verify that the authentication key is generated by a secure random number generator, containing at least 20 bits of entropy (typically a six digital random number is sufficient). | ✓ | ✓ | ✓ | 5.1.3.2 |
| 2.7.8 | Verify that rate limiting is in place if the authentication secret uses less than 64 bits of entropy (typically all numeric authenticators). | ✓ | ✓ | ✓ | 5.1.3.2 |

### V2.8 Single Factor OTP Requirements

Single factor one time passwords are physical or soft tokens that display a continually changing pseudo-random one time challenge. These devices make phishing (impersonation) difficult, but not impossible. This type of authenticator is considered "something you have".

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | NIST § |
| 2.8.1 | Verify that symmetric keys used to verify submitted codes is highly protected, such as using an HSM or OS based key storage. | o | ✓ | ✓ | 5.1.4.2 |
| 2.8.2 | Verify that approved cryptographic algorithms are used in the generation, seeding, and verification. | o | ✓ | ✓ | 5.1.4.2 |
| 2.8.3 | Verify that time-based tokens have a defined lifetime before needing re-seeding or replacement. | o | o | ✓ | 5.1.4.2 |
| 2.8.4 | Verify that time-based OTP can be used only once within the validity period. | ✓ | ✓ | ✓ | 5.1.4.2 |
| 2.8.5 | Verify that rate limiting is in place if the authentication secret uses less than 64 bits of entropy (typically all numeric OTP authenticators). | o | ✓ | ✓ | 5.1.4.2 |

### V2.9 Multi-factor OTP Verifier Requirements

Multi-factor tokens are similar to single factor OTP tokens, but require a valid PIN code, biometric unlocking, USB insertion or NFC pairing or some additional value (such as transaction signing calculators) to be entered to create the final OTP.

In the context of this section, the claimant should be read as the person or organization who enrolled the OTP device. Claimants may not necessarily be the person in possession of the OTP device.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | NIST § |
| 2.9.1 | Verify that symmetric keys used to verify submitted codes is highly protected, such as using an HSM or OS based key storage. | o | ✓ | ✓ | 5.1.5.2 |
| 2.9.2 | Verify that approved cryptographic algorithms are used in the generation, seeding, and verification. | o | ✓ | ✓ | 5.1.5.2 |
| 2.9.3 | Verify that MFA OTP device is a multi-factor device. Otherwise, the device should be treated as a single factor OTP device. | o | o | ✓ | 5.1.5.2 |
| 2.9.4 | Verify that time-based OTPs have a defined lifetime before needing re-seeding or replacement. | o | o | ✓ | 5.1.5.2 |
| 2.9.5 | Verify that time-based OTP can be used only once within the validity period. | o | ✓ | ✓ | 5.1.5.2 |
| 2.9.6 | Verify that if a time-based OTP is re-used during the validity period, a warning is presented to the claimant, and optionally the user or the existing session. | o | ✓ | ✓ | 5.1.5.2 |
| 2.9.7 | Verify that rate limiting is in place if the authentication secret uses less than 64 bits of entropy (typically all numeric OTP authenticators). | o | ✓ | ✓ | 5.1.5.2 |

### V2.10 Cryptographic Software and Devices Verifier Requirements

Cryptographic security keys are smart cards or FIDO keys, where the user has to plug in or pair the cryptographic device to the computer to complete authentication. Verifiers send a challenge nonce to the cryptographic devices or software, and the device or software calculates a response based upon a securely stored cryptographic key.

The requirements for single factor cryptographic devices and software, and multi-factor cryptographic devices and software are the same, as verification of the cryptographic authenticator proves possession of the authentication factor.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | NIST § |
| 2.10.1 | 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. | ✓ | ✓ | ✓ | 5.1.7.2 |
| 2.10.2 | Verify that the challenge nonce is at least 64 bits in length, and statistically unique or unique over the lifetime of the cryptographic device. | ✓ | ✓ | ✓ | 5.1.7.2 |
| 2.10.3 | Verify that approved cryptographic algorithms are used in the generation, seeding, and verification. | o | ✓ | ✓ | 5.1.7.2 |

### V2.11 Service Authentication Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | NIST § |
| 2.11.1 | Integration secrets SHOULD NOT rely on unchanging passwords, such as API keys or shared privileged accounts. If passwords are required, the credential should not be a default account and stored with sufficient protection to prevent offline recovery attacks, including local system access. | Software | OS assisted | HSM | 5.1.1.1 |
| 2.11.2 | 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 (L1), hardware trusted platform module (TPM), or a hardware security module (L3) is recommended for password storage. | Software | OS assisted | HSM | TBA |

### Additional US Agency Requirements

US Agencies have mandatory requirements concerning NIST 800-63. The Application Security Verification Standard has always been about the 80% of controls that apply to nearly 100% of apps, and not the last 20% of advanced controls or those that have limited applicability. As such, the ASVS is a strict subset of NIST 800-63, especially for IAL1/2 and AAL1/2 classifications, but is not sufficiently comprehensive, particularly concerning IAL3/AAL3 classifications.

We strongly urge US government agencies to review and implement NIST 800-63 in its entirety.

## Glossary of terms

|  |  |
| --- | --- |
| Term | Meaning |
| CSP | Credential Service Provider also called an Identity Provider |
| Authenticator | TBA |
| Verifier | "An entity that verifies the claimant's identity by verifying the claimant's possession and control of one or two authenticators using an authentication protocol. To do this, the verifier may also need to validate credentials that link the authenticator(s) to the subscriber's identifier and check their status" |
| OTP | One-time password |
| SFA | Single factor authenticators, such as something you know (memorized secrets, passwords, passphrases, PINs), something you are (biometrics, fingerprint, face scans), or something you have (OTP tokens, a cryptographic device such as a smart card), |
| MFA | Multi factor authenticator, which includes two or more single factors |

## References

For more information, see also:

[NIST 800-63 - Digital Identity Guidelines](https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63-3.pdf)

[NIST 800-63 A - Enrollment and Identity Proofing](https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63a.pdf)

[NIST 800-63 B - Authentication and Lifecycle Management](https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63b.pdf)

[NIST 800-63 C - Federation and Assertions](https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63c.pdf)

[NIST 800-63 FAQ](https://pages.nist.gov/800-63-FAQ/)

[OWASP Testing Guide 4.0: Testing for Authentication](https://www.owasp.org/index.php/Testing_for_authentication)

[OWASP Cheat Sheet - Password storage](https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet)

[OWASP Cheat Sheet - Forgot password](https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet)

[OWASP Cheat Sheet - Choosing and using security questions](https://www.owasp.org/index.php/Choosing_and_Using_Security_Questions_Cheat_Sheet)

# V3: Session Management Verification Requirements

## Control Objective

One of the core components of any web-based application or API is the mechanism by which it controls and maintains the state for a user or device interacting with it. Session management changes a stateless protocol to stateful, which is critical at differentiating different users or devices.

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

Sessions are unique to each individual and cannot be guessed or shared.

Sessions are invalidated when no longer required and timed out during periods of inactivity.

## Security Verification Requirements

### V3.1 Fundamental Session Management Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | NIST § |
| 3.1.1 | Verify that the session token is never disclosed in URLs, error messages, or logs. This includes verifying that the application does not support URL rewriting of session cookies. |  | ✓ | ✓ | - |

### V3.2 Session Binding Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | NIST § |
| 3.2.1 | Verify that session tokens are created or generated after authentication, not before. | ✓ | ✓ | ✓ | 7.1 |
| 3.2.2 | Verify that session tokens are created using approved cryptographic algorithms with at least 64 bits of entropy. | ✓ | ✓ | ✓ | 7.1 |
| 3.2.3 | Verify that session tokens are not stored in insecure local storage, such as HTML 5 local storage. | ✓ | ✓ | ✓ | 7.1 |
| 3.2.4 | Verify that active session functionality exists, which allows users to either selectively or completely log out all active devices / sessions. |  | ✓ | ✓ | 7.1 |

TLS or another secure transport channel is mandatory for session management. This is covered off in the Communications Security chapter.

### V3.3 Session logout and timeout Requirements

Session timeouts have been aligned with NIST 800-63, which permits much longer session timeouts than traditionally permitted. This reflects modern common industry practice but is backed by increasing timeouts when multi-factor or stronger authenticators are used.

L1 in this context is IAL1/AAL1, L2 is IAL2/AAL3, L3 is IAL3/AAL3. For IAL2/AAL2 and IAL3/AAL3, the shorter idle timeout is, the lower bound of idle times for being logged out or re-authenticated to resume the session.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | NIST § |
| 3.3.1 | Verify that logout timeout invalidates or erases any client- or server-side session storage, such that the back button or a downstream relying party does not resume an authenticated session, including across relying parties. | ✓ | ✓ | ✓ | 7.1 |
| 3.3.2 | Verify that absolute or idle timeouts invalidates or erases any client- or server-side session storage. |  | ✓ | ✓ | 7.1 |
| 3.3.3 | Verify if authenticators permit users to remain logged in, that re-authentication occurs periodically both when actively used or after an idle period. | 30 days | 12 hours or 30 minutes of inactivity, 2FA optional | 12 hours or 15 minutes of inactivity, with 2FA | 7.2 |
| 3.3.4 | Verify that the user can terminate all other active sessions after a successful change password process, and this is effective across the application, federated login (if present) and any relying parties. |  | ✓ | ✓ | - |

### V3.4 Cookie-based Session Management

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | NIST § |
| 3.4.1 | Verify that cookie-based session IDs have the 'Secure' attribute. | ✓ | ✓ | ✓ | 7.1.1 |
| 3.4.2 | Verify that cookie-based session IDs have the 'HttpOnly' attributes. | ✓ | ✓ | ✓ | 7.1.1 |
| 3.4.3 | Verify that cookie-based session IDs have minimum practical hostnames, domain and path attributes, along with the 'SameSite' attribute. | ✓ | ✓ | ✓ | 7.1.1 |
| 3.4.4 | Verify that cookie-based session IDs are set to expire soon after the default session timeout period. | ✓ | ✓ | ✓ | 7.1.1 |

### V3.5 Token based Session Management

Token-based session management includes JWT, oAuth, SAML, and API keys. Of these, API keys are known weak and should not be used in new code.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | NIST § |
| 3.5.1 | Verify that OAuth and refresh tokens are not considered as the presence of the subscriber; this pertains to high-value transactions where re-authentication may be required to protect the user. |  | ✓ | ✓ | 7.1.2 |
| 3.5.2 | Verify single factor unchanging API secrets, and keys are not used, except with legacy implementations. |  | ✓ | ✓ | - |
| 3.5.3 | Verify that stateless session tokens enveloping sensitive session data are digitally signed or encrypted and regularly verified in a timely fashion to protect against tampering, enveloping, replay, null cipher and key substitution attacks. |  | ✓ | ✓ | - |

### V3.6 Re-authentication from a federation or assertion

This section relates to those writing relying party (RP) or credential service provider (CSP) code. If relying on code implementing these features, ensure that these issues are handled correctly.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | NIST § |
| 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. |  |  | ✓ | 7.2.1 |
| 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 subscriber. |  |  | ✓ | 7.2.1 |

### V3.7 Defences against session management exploits

There are a small number of session management attacks, some related to the user experience (UX) of sessions. Previously, based on ISO 27002 requirements, the ASVS has required blocking multiple simultaneous sessions. Blocking simultaneous sessions is no longer appropriate, not only as modern users have many devices or the app is an API without a browser session, but in most of these implementations, the last authenticator wins, which is is often the attacker. This section provides leading guidance on deterring, delaying and detecting session management attacks using code.

#### Description of the half-open attack

In early 2018, several financial institutions were compromised using what the attackers called "half-open attacks". This term has stuck in the industry. The attackers struck multiple institutions with different proprietary code bases, and indeed it seems different code bases within the same institutions. The half-open attack is exploiting a design pattern flaw commonly found in many existing authentication, session management and access control systems.

Attackers start a half-open attack by attempting to lock, reset, or recover a credential. A popular session management design pattern re-uses user profile session objects/models between unauthenticated, half-authenticated (password resets, forgot username), and fully authenticated code. This design pattern populates a valid session object or token containing the victim's profile, including password hashes and roles. If access control checks in controllers or routers does not correctly verify that the user is fully logged in, the attacker will be able to act as the user. Attacks could include changing the user's password to a known value, update the email address to perform a valid password reset, disable multi-factor authentication or enrol a new MFA device, reveal or change API keys, and so on.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | NIST § |
| 3.7.1 | Verify that all post-reset, post-registration, and post-authentication high value or adminstrative functionality verifies the user session is both fully logged in and has a valid post-authentication role before permitting any changes or transactions, especially in relation to user profile updates, password changes, MFA enrollment, and administrative functionality. | ✓ | ✓ | ✓ | - |

## References

For more information, see also:

[OWASP Testing Guide 4.0: Session Management Testing](https://www.owasp.org/index.php/Testing_for_Session_Management)

[OWASP Session Management Cheat Sheet](https://www.owasp.org/index.php/Session_Management_Cheat_Sheet)

# V4: Access Control Verification Requirements

## Control Objective

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

Persons accessing resources holds valid credentials to do so.

Users are associated with a well-defined set of roles and privileges.

Role and permission metadata is protected from replay or tampering.

## Security Verification Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | Since |
| **4.1** | Verify that the principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege. | ✓ | ✓ | ✓ | 1.0 |
| **4.4** | Verify that sensitive data and APIs are protected against direct object attacks targeting creation, reading, updating and deletion of records. | ✓ | ✓ | ✓ | 4.0 |
| **4.5** | Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS\_Store, .git or .svn folders. | ✓ | ✓ | ✓ | 1.0 |
| **4.8** | Verify that access controls fail securely. | ✓ | ✓ | ✓ | 1.0 |
| **4.9** | Verify that the same access control rules implied by the presentation layer are enforced on the server side. | ✓ | ✓ | ✓ | 1.0 |
| **4.10** | Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized. |  | ✓ | ✓ | 1.0 |
| **4.11** | Verify that there is preferably only one vetted access control mechanism for protecting access to protected data and resources, such that hard coded access control checks are not required throughout the application. |  | ✓ | ✓ | 4.0 |
| **4.12** | Verify that all access control decisions can be logged and all failed decisions are logged. |  | ✓ | ✓ | 2.0 |
| **4.13** | Verify that the application or framework enforces a strong anti-CSRF mechanism any sensitive functionality. | ✓ | ✓ | ✓ | 4.0 |
| **4.14** | Verify the application has sufficient anti-automation to detect and protect against data exfiltration, excessive business logic requests, or denial of service attacks. |  | ✓ | ✓ | 4.0 |
| **4.15** | Verify the application has additional authorization (such as step up or adaptive authentication) for lower value systems, and / or segregation of duties for high value applications to enforce anti-fraud controls as per the risk of application and past fraud. |  | ✓ | ✓ | 3.0 |
| **4.16** | Verify that access control policy is enforced by trusted server-side components. | ✓ | ✓ | ✓ | 4.0 |
| **4.17** | Verify that data-level access control is implemented such that access to individual records can be managed in a centralized and standard way. | ✓ | ✓ | ✓ | 4.0 |
| **4.18** | Verify administrative interfaces use appropriate multi-factor authentication to prevent unauthorized use. | ✓ | ✓ | ✓ | 4.0 |

## References

For more information, see also:

[OWASP Testing Guide 4.0: Authorization](https://www.owasp.org/index.php/Testing_for_Authorization)

[OWASP Cheat Sheet: Access Control](https://www.owasp.org/index.php/Access_Control_Cheat_Sheet)

[OWASP CSRF Cheat Sheet](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet)

[OWASP REST Cheat Sheet](https://www.owasp.org/index.php/REST_Security_Cheat_Sheet)

Anti-automation can be achieved in many ways, including the user of [OWASP AppSensor](https://www.owasp.org/index.php/OWASP_AppSensor_Project) and [OWASP Automated Threats to Web Applications](https://www.owasp.org/index.php/OWASP_Automated_Threats_to_Web_Applications)

# V5: Validation, Sanitization and Encoding Verification Requirements

## Control Objective

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

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

All input is validated to be correct and fit for the intended purpose.

Data from an external entity or client should never be trusted and should be handled accordingly.

## Security Verification Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | Since |
| **5.1** | Verify that server-side input validation failures result in request rejection and are logged. | ✓ | ✓ | ✓ | 1.0 |
| **5.2** | Verify that input validation is enforced on the server-side. | ✓ | ✓ | ✓ | 1.0 |
| **5.3** | Verify that a centralized input validation control mechanism is used by the application. | ✓ | ✓ | ✓ | 1.0 |
| **5.4** | Verify that data selection or database queries (e.g. SQL, HQL, ORM, NoSQL, XPath, LDAP) are protected from respective injection attacks. In cases where query parameterization or escaping is not possible for certain variables, be sure to strictly validate or whitelist the data, such as ORDER BY values, or column names. | ✓ | ✓ | ✓ | 4.0 |
| **5.5** | Verify that the application is not susceptible to LDAP Injection, or that security controls prevent LDAP Injection. | ✓ | ✓ | ✓ | 2.0 |
| **5.6** | Verify that the application is not susceptible to OS Command Injection, or that security controls prevent OS Command Injection. | ✓ | ✓ | ✓ | 2.0 |
| **5.7** | Verify that the application is not susceptible to Remote File Inclusion (RFI) or Local File Inclusion (LFI) when content is used that is a path to a file. | ✓ | ✓ | ✓ | 3.0 |
| **5.8** | Verify that the application is not susceptible XPath injection or XML injection attacks. | ✓ | ✓ | ✓ | 2.0 |
| **5.9** | Verify that all string variables placed into HTML or other web client code are either properly contextually encoded manually, or utilize templates that automatically contextually encode to ensure the application is not susceptible to reflected, stored or DOM Cross-Site Scripting (XSS) attacks. | ✓ | ✓ | ✓ | 4.0 |
| **5.10** | Verify that the application does not contain mass parameter assignment (AKA automatic variable binding) vulnerabilities. |  | ✓ | ✓ | 4.0 |
| **5.11** | Verify that the application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (GET, POST, cookies, headers, environment, etc). |  | ✓ | ✓ | 2.0 |
| **5.12** | Verify that all input data is validated, not only HTML form fields but all sources of input such as REST calls, query parameters, HTTP headers, cookies, batch files, RSS feeds, etc; using positive validation (whitelisting), then lesser forms of validation such as grey listing (eliminating known bad strings), or rejecting bad inputs (blacklisting). |  | ✓ | ✓ | 3.0 |
| **5.13** | Verify that structured data is strongly typed and validated against a defined schema including allowed characters, length and pattern (e.g. credit card numbers or telephone, or validating that two related fields are reasonable, such as validating suburbs and zip or post codes match). |  | ✓ | ✓ | 3.0 |
| **5.14** | Verify that unstructured data is sanitized to enforce generic safety measures such as allowed characters and length, and characters potentially harmful in given context should be escaped (e.g. natural names with Unicode or apostrophes, such as ねこ or O'Hara). |  | ✓ | ✓ | 3.0 |
| **5.15** | Verify that all untrusted HTML input from WYSIWYG editors or similar is properly sanitized with an HTML sanitizer library or framework feature. | ✓ | ✓ | ✓ | 3.0 |
| **5.16** | Verify that where data is transferred from one DOM context to another, the transfer uses safe JavaScript methods, such as using innerText or JQuery .val to ensure the application is not susceptible to DOM Cross-Site Scripting (XSS) attacks. |  | ✓ | ✓ | 4.0 |
| **5.17** | Verify when parsing JSON in browsers or JavaScript based backends, that JSON.parse is used to parse the JSON document. Do not use eval() to parse JSON. |  | ✓ | ✓ | 3.0 |
| **5.18** | Verify the application for Server-Side Request Forgery vulnerabilities. | ✓ | ✓ | ✓ | 4.0 |
| **5.19** | Verify that the application correctly restricts XML parsers to only use the most restrictive configuration possible and to ensure that dangerous features such as resolving external entities are disabled. | ✓ | ✓ | ✓ | 4.0 |
| **5.20** | Verify that deserialization of untrusted data is avoided or is extensively protected in both custom code and third party libraries (such as JSON, XML and YAML parsers). | ✓ | ✓ | ✓ | 4.0 |

## References

For more information, see also:

[OWASP Testing Guide 4.0: Input Validation Testing](https://www.owasp.org/index.php/Testing_for_Input_Validation)

[OWASP Cheat Sheet: Input Validation](https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet)

[OWASP Testing Guide 4.0: Testing for HTTP Parameter Pollution](https://www.owasp.org/index.php/Testing_for_HTTP_Parameter_pollution_%28OTG-INPVAL-004%29)

[OWASP LDAP Injection Cheat Sheet](https://www.owasp.org/index.php/LDAP_Injection_Prevention_Cheat_Sheet)

[OWASP Testing Guide 4.0: Client Side Testing](https://www.owasp.org/index.php/Client_Side_Testing)

[OWASP Cross Site Scripting Prevention Cheat Sheet](https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet)

[OWASP DOM Based Cross Site Scripting Prevention Cheat Sheet](https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet)

[OWASP Java Encoding Project](https://www.owasp.org/index.php/OWASP_Java_Encoder_Project)

For more information on auto-escaping, please see:

[Reducing XSS by way of Automatic Context-Aware Escaping in Template Systems](http://googleonlinesecurity.blogspot.com/2009/03/reducing-xss-by-way-of-automatic.html)

[AngularJS Strict Contextual Escaping](https://docs.angularjs.org/api/ng/service/$sce)

[AngularJS ngBind](https://docs.angularjs.org/api/ng/directive/ngBind)

[Angular Sanitzation](https://angular.io/guide/security#sanitization-and-security-contexts)

[Angular Template Security](https://angular.io/guide/template-syntax#content-security)

[ReactJS Escaping](https://reactjs.org/docs/introducing-jsx.html#jsx-prevents-injection-attacks)

[Improperly Controlled Modification of Dynamically-Determined Object Attributes](https://cwe.mitre.org/data/definitions/915.html)

# V7: Cryptography Verification Requirements

## Control Objective

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

That all cryptographic modules fail in a secure manner and that errors are handled correctly.

That a suitable random number generator is used when randomness is required.

That access to keys is managed in a secure way.

## Security Verification Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | Since |
| **7.2** | Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable Padding Oracle attacks. | ✓ | ✓ | ✓ | 1.0 |
| **7.6** | Verify that all random numbers, random file names, random GUIDs, and random strings are generated using the cryptographic module’s approved random number generator when these random values are intended to be not guessable by an attacker. |  | ✓ | ✓ | 1.0 |
| **7.7** | Verify that cryptographic algorithms used by the application have been validated against FIPS 140-2 or an equivalent standard. | ✓ | ✓ | ✓ | 1.0 |
| **7.11** | Verify that consumers of cryptographic services do not have direct access to key material, such as by using key vaults or API based alternatives. |  | ✓ | ✓ | 4.0 |
| **7.12** | Verify that Personally Identifiable Information (PII) and other sensitive data is stored encrypted while at rest. |  | ✓ | ✓ | 4.0 |
| **7.13** | Verify that sensitive passwords or key material maintained in memory is overwritten with zeros as soon as it is no longer required, to mitigate memory dumping attacks. |  | ✓ | ✓ | 4.0 |
| **7.14** | Verify that all keys and passwords are replaceable, and are generated or replaced at installation time. |  | ✓ | ✓ | 3.0 |
| **7.15** | Verify that random numbers are created with proper entropy even when the application is under heavy load, or that the application degrades gracefully in such circumstances. |  |  | ✓ | 3.0 |
| **7.16** | Verify that industry proven cryptographic modules are used instead of custom coded cryptography. | ✓ | ✓ | ✓ | 4.0 |

## References

For more information, see also:

[OWASP Testing Guide 4.0: Testing for weak Cryptography](https://www.owasp.org/index.php/Testing_for_weak_Cryptography)

[OWASP Cheat Sheet: Cryptographic Storage](https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet)

# V8: Error Handling and Logging Verification Requirements

## Control Objective

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

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

Not collecting or logging sensitive information if not specifically required.

Ensuring all logged information is handled securely and protected as per its data classification.

Ensuring that logs are not forever, but have an absolute lifetime that is as short as possible.

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

## Security Verification Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | Since |
| **8.1** | Verify that the application does not output error messages or stack traces containing sensitive data that could assist an attacker, including session id, software/framework versions and personal information. | ✓ | ✓ | ✓ | 1.0 |
| **8.3** | Verify that the application logs security relevant events including successful and failed authentication events, access control failures and input validation failures. | ✓ | ✓ | ✓ | 4.0 |
| **8.4** | Verify that each log event includes necessary information that would allow for a detailed investigation of the timeline when an event happens. |  | ✓ | ✓ | 1.0 |
| **8.5** | Verify that all events that include untrusted data will not execute as code in the intended log viewing software. |  | ✓ | ✓ | 1.0 |
| **8.6** | Verify that security logs are protected from unauthorized access and modification. |  | ✓ | ✓ | 1.0 |
| **8.7** | Verify that the application does not log credentials, session tokens, payment instruments, or sensitive data, as defined under local privacy laws or relevant security policy. |  | ✓ | ✓ | 3.0 |
| **8.8** | Verify the application appropriately encodes user supplied data to prevent log injection. | ✓ | ✓ | ✓ | 4.0 |
| **8.12** | Verify that logs are transmitted to a remote system for analysis, detection, alerting, and escalation. |  |  | ✓ | 4.0 |
| **8.13** | Verify that time sources are synchronized to the correct time and time zone. | ✓ | ✓ | ✓ | 4.0 |
| **8.14** | Verify all authentication decisions are logged, without storing sensitive session identifiers or memorized secrets. This should include requests with relevant metadata needed for security investigations. | ✓ | ✓ | ✓ | 4.0 |

## References

For more information, see also:

[OWASP Testing Guide 4.0 content: Testing for Error Handling](https://www.owasp.org/index.php/Testing_for_Error_Handling)

# V9: Data Protection Verification Requirements

## Control Objective

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

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

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

Confidentiality: Data should be protected from unauthorized observation or disclosure both in transit and when stored.

Integrity: Data should be protected being maliciously created, altered or deleted by unauthorized attackers.

Availability: Data should be available to authorized users as required.

## Security Verification Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | Since |
| **9.2** | Verify that sensitive data is classified and a policy is created for how this data must be protected. | ✓ | ✓ | ✓ | 4.0 |
| **9.3** | Verify that sensitive data is sent to the server in the HTTP/S message body or headers and that all query string parameters from any HTTP verb does not contain sensitive data. | ✓ | ✓ | ✓ | 1.0 |
| **9.4** | Verify that the application sets sufficient anti-caching headers such that any sensitive and personal information displayed by the application or entered by the user should not be cached on disk by mainstream modern browsers (e.g. visit <about:cache> to review disk cache). | ✓ | ✓ | ✓ | 4.0 |
| **9.5** | Verify that on the server, all cached or temporary copies of sensitive data stored are protected from unauthorized access or purged/invalidated after the authorized user accesses the sensitive data. |  | ✓ | ✓ | 1.0 |
| **9.6** | Verify that users have a method to remove or export their data on demand. | ✓ | ✓ | ✓ | 4.0 |
| **9.7** | Verify the application minimizes the number of parameters in a request, such as hidden fields, Ajax variables, cookies and header values. |  | ✓ | ✓ | 2.0 |
| **9.8** | Verify the application has the ability to detect and alert on abnormal numbers of requests. |  | ✓ | ✓ | 4.0 |
| **9.9** | 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. | ✓ | ✓ | ✓ | 3.0.1 |
| **9.10** | Verify accessing sensitive data is logged, if the data is collected under relevant data protection directives or where logging of accesses is required. |  | ✓ | ✓ | 3.0 |
| **9.11** | Verify that sensitive information maintained in memory is overwritten with zeros as soon as it is no longer required, to mitigate memory dumping attacks. |  | ✓ | ✓ | 3.0.1 |
| **9.12** | Verify that sensitive or private information that is required to be encrypted, is encrypted using approved algorithms that provides both confidentiality and integrity. | ✓ | ✓ | ✓ | 3.0.1 |
| **9.13** | 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. | ✓ | ✓ | ✓ | 4.0 |
| **9.14** | Verify that authenticated data is cleared from client storage, such as the browser DOM, after the client or session is terminated. |  | ✓ | ✓ | 3.0 |

## References

For more information, see also:

[Consider using Security Headers website to check security and anti-caching headers](https://securityheaders.io)

[OWASP Secure Headers project](https://www.owasp.org/index.php/OWASP_Secure_Headers_Project)

[User Privacy Protection Cheat Sheet](https://www.owasp.org/index.php/User_Privacy_Protection_Cheat_Sheet)

# V10: Communications Verification Requirements

## Control Objective

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

TLS or strong encryption is always used, regardless of the sensitivity of the data being transmitted

The most recent leading configuration advice is used to 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.

Leading industry advice on secure TLS configuration changes frequently, often due to catastrophic breaks in existing algorithms and ciphers. Always use the most recent versions of TLS configuration review tools (such as SSLyze or other TLS scanners) to configure the preferred order and algorithm selection. Configuration should be periodically checked to ensure that secure communications configuration is always present and effective.

## Security Verification Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | Since |
| **10.1** | Verify that a path can be built from a trusted CA to each Transport Layer Security (TLS) server certificate, and that each server certificate is valid. | ✓ | ✓ | ✓ | 1.0 |
| **10.3** | Verify that TLS is used for all connections (including for authentication, API or web service calls, backend, external, and partner connections), and does not fall back to insecure or unencrypted protocols. | ✓ | ✓ | ✓ | 3.0 |
| **10.4** | Verify that backend TLS connection failures are logged. |  |  | ✓ | 1.0 |
| **10.5** | Verify that certificate paths are built and verified for all client certificates using configured trust anchors and revocation information. |  |  | ✓ | 1.0 |
| **10.6** | Verify that all connections to external systems that involve sensitive information or functions are authenticated. |  | ✓ | ✓ | 1.0 |
| **10.11** | Verify that HTTP Strict Transport Security headers are included on all requests and for all subdomains, such as Strict-Transport-Security: max-age=15724800; includeSubdomains. | ✓ | ✓ | ✓ | 3.0 |
| **10.12** | Verify that the application's URL has been submitted to a preloaded list of Strict Transport Security domains maintained by web browser vendors. Please see the references below. |  |  | ✓ | 3.0 |
| **10.14** | Verify that proper certification revocation, such as Online Certificate Status Protocol (OCSP) Stapling, is enabled and configured. | ✓ | ✓ | ✓ | 3.0 |
| **10.15** | Verify that only strong algorithms, ciphers, and protocols are enabled, with the strongest algorithms and ciphers set as preferred, to provide both confidentiality and integrity. | ✓ | ✓ | ✓ | 3.0 |
| **10.17** | Verify that old versions of SSL and TLS protocols, algorithms, ciphers, and configuration are not used, such as SSLv2/3 or TLS 1.0. The latest version of TLS should be the preferred cipher suite. | ✓ | ✓ | ✓ | 3.0 |

## References

For more information, see also:

[OWASP – TLS Cheat Sheet](https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet)

Notes on “Approved modes of TLS”. In the past, the ASVS referred to the US standard FIPS 140-2, but as a global standard, applying US standards can be difficult, contradictory, or confusing to apply. A better method of achieving compliance with 10.8 would be to review guides such as [Mozilla's Server Side TLS](https://wiki.mozilla.org/Security/Server_Side_TLS) or [generate known good configurations](https://mozilla.github.io/server-side-tls/ssl-config-generator/), and use known TLS evaluation tools, such as sslyze, various vulnerability scanners or trusted TLS online assessment services to obtain a desired level of security. In general, we see non-compliance for this section being the use of outdated or insecure ciphers and algorithms, the lack of perfect forward secrecy, outdated or insecure SSL protocols, weak preferred ciphers, and so on.

[Certificate pinning](https://tools.ietf.org/html/rfc7469). The rationale behind certificate pinning for production and backup keys is [business continuity](https://noncombatant.org/2015/05/01/about-http-public-key-pinning/)

[OWASP Certificate Pinning Cheat Sheet](https://www.owasp.org/index.php/Pinning_Cheat_Sheet)

[OWASP Certificate and Public Key Pinning](https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning)

[Time of first use (TOFU) Pinning](https://developer.mozilla.org/en/docs/Web/Security/Public_Key_Pinning)

[Pre-loading HTTP Strict Transport Security](https://www.chromium.org/hsts)

# V11: HTTP Security Configuration Verification Requirements

## Control Objective

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

The application server is suitably hardened from a default configuration.

HTTP responses contain a safe character set in the content type header.

## Security Verification Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | Since |
| **11.1** | Verify that the application server accepts the HTTP methods in use by the application or API, including pre-flight OPTIONS. | ✓ | ✓ | ✓ | 1.0 |
| **11.2** | Verify that every HTTP response contains a content type header specifying a safe character set (e.g., UTF-8, ISO 8859-1). | ✓ | ✓ | ✓ | 1.0 |
| **11.3** | Verify that HTTP headers added by a trusted proxy or SSO devices, such as a bearer token, are authenticated by the application. |  | ✓ | ✓ | 2.0 |
| **11.4** | Verify that a suitable X-Frame-Options or Content-Security-Policy: frame-ancestors header is in use for sites where content should not be embedded in a 3rd party site. |  | ✓ | ✓ | 3.0.1 |
| **11.5** | Verify that the HTTP headers or any part of the HTTP response do not expose detailed version information of system components. | ✓ | ✓ | ✓ | 2.0 |
| **11.6** | Verify that all API responses contain X-Content-Type-Options: nosniff and Content-Disposition: attachment; filename="api.json" (or other appropriate filename for the content type). | ✓ | ✓ | ✓ | 3.0 |
| **11.7** | Verify that a content security policy (CSPv2) is in place that helps mitigate common DOM, XSS, JSON, and JavaScript injection vulnerabilities. | ✓ | ✓ | ✓ | 3.0.1 |
| **11.8** | Verify that the X-XSS-Protection: 1; mode=block header is in place to enable browser reflected XSS filters. | ✓ | ✓ | ✓ | 3.0 |
| **11.9** | 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. | ✓ | ✓ | ✓ | 4.0 |

## References

For more information, please see:

[OWASP Testing Guide 4.0: Testing for HTTP Verb Tampering](https://www.owasp.org/index.php/Testing_for_HTTP_Verb_Tampering_%28OTG-INPVAL-003%29)

Adding Content-Disposition to API responses helps prevent many attacks based on misunderstanding on the MIME type between client and server, and the "filename" option specifically helps prevent [Reflected File Download attacks.](https://www.blackhat.com/docs/eu-14/materials/eu-14-Hafif-Reflected-File-Download-A-New-Web-Attack-Vector.pdf)

[Content Security Policy Cheat Sheet](https://www.owasp.org/index.php?title=Content_Security_Policy_Cheat_Sheet)

[Exploiting CORS misconfiguration for BitCoins and Bounties](https://portswigger.net/blog/exploiting-cors-misconfigurations-for-bitcoins-and-bounties)

# V13: Malicious Code Verification Requirements

## Control Objective

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

Malicious activity is handled securely and properly as to not affect the rest of the application.

Do not have time bombs or other time based attacks built into them.

Do not “phone home” to malicious or unauthorized destinations.

Applications do not have back doors, Easter eggs, salami attacks, or logic flaws that can be controlled by an attacker.

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

## Security Verification Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | Since |
| **13.1** | Verify all malicious activity is adequately sandboxed, containerized or isolated to delay and deter attackers from attacking other applications. | ✓ | ✓ | ✓ | 2.0 |
| **13.2** | Verify that the application source code and third party libraries do not contain back doors or easter eggs. | ✓ | ✓ | ✓ | 3.0.1 |

## References

# V15: Business Logic Verification Requirements

## Control Objective

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

The business logic flow is sequential, cannot be obviated, and processed in order.

Business logic includes limits to detect and prevent automated attacks, such as continuous small funds transfers, or adding a million friends one at a time, and so on.

High value business logic flows have considered abuse cases and malicious actors, and have protections against spoofing, tampering, repudiation, information disclosure, and elevation of privilege attacks.

## Security Verification Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | Since |
| **15.1** | Verify the application will only process business logic flows in sequential step order, with all steps being processed in realistic human time, and not process out of order, skipped steps, process steps from another user, or too quickly submitted transactions. |  | ✓ | ✓ | 2.0 |
| **15.2** | Verify the application has business limits and correctly enforces on a per user basis, with configurable alerting and automated reactions to automated or unusual attack. |  | ✓ | ✓ | 2.0 |

## References

For more information, see also:

[OWASP Testing Guide 4.0: Business Logic Testing](https://www.owasp.org/index.php/Testing_for_business_logic)

[OWASP Cheat Sheet](https://www.owasp.org/index.php/Business_Logic_Security_Cheat_Sheet)

# V16: File and Resources Verification Requirements

## Control Objective

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

Untrusted file data should be handled accordingly and in a secure manner.

Obtained from untrusted sources are stored outside the web root and limited permissions.

## Security Verification Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | Since |
| **16.1** | Verify that URL redirects and forwards only allow whitelisted destinations, or show a warning when redirecting to potentially untrusted content. | ✓ | ✓ | ✓ | 2.0 |
| **16.2** | Verify that untrusted file data submitted to the application is not used directly with file I/O commands, particularly to protect against path traversal, local file include, file mime type, reflective file download, and OS command injection vulnerabilities. | ✓ | ✓ | ✓ | 4.0 |
| **16.3** | Verify that files obtained from untrusted sources are validated to be of expected type and scanned by antivirus scanners to prevent upload of known malicious content. | ✓ | ✓ | ✓ | 2.0 |
| **16.4** | Verify that untrusted data is not used within inclusion, class loader, or reflection capabilities to prevent remote/local code execution vulnerabilities. | ✓ | ✓ | ✓ | 4.0 |
| **16.5** | Verify that untrusted data is not used within cross-domain resource sharing (CORS) to protect against arbitrary remote content. | ✓ | ✓ | ✓ | 2.0 |
| **16.6** | Verify that files obtained from untrusted sources are stored outside the web root, with limited permissions, preferably with strong validation. |  | ✓ | ✓ | 3.0 |
| **16.7** | Verify that the web or application server is configured by default to deny access to remote resources or systems outside the web or application server. |  | ✓ | ✓ | 2.0 |
| **16.8** | Verify the application code does not execute uploaded data obtained from untrusted sources. | ✓ | ✓ | ✓ | 3.0 |
| **16.9** | Verify that unsupported, insecure or deprecated client-side technologies are not used, such as NSAPI plugins, Flash, Shockwave, Active-X, Silverlight, NACL, or client-side Java applets. | ✓ | ✓ | ✓ | 4.0 |
| **16.10** | Verify that the cross-domain resource sharing (CORS) Access-Control-Allow-Origin header does not reflect the request's origin header or support the "null" origin. | ✓ | ✓ | ✓ | 4.0 |

## References

For more information, see also:

[File Extension Handling for Sensitive Information](https://www.owasp.org/index.php/Unrestricted_File_Upload)

[Reflective file download by Oren Hatif](https://www.trustwave.com/Resources/SpiderLabs-Blog/Reflected-File-Download---A-New-Web-Attack-Vector/)

# V17: Mobile Verification Requirements

## ASVS Mobile Removal Notice

This section previously contained controls that are mobile application specific. This section is being removed and replaced by the OWASP Mobile Application Security Verification Standard.

## References

[OWASP Mobile Application Security Verification Standard](https://www.owasp.org/index.php/OWASP_Mobile_Security_Project)

# V18: API and Web Service Verification Requirements

## Control Objective

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

Adequate authentication, session management and authorization of all web services.

Input validation of all parameters that transit from a lower to higher trust level.

Basic interoperability of SOAP web services layer to promote API use.

## Security Verification Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | Since |
| **18.1** | Verify that the same encoding style is used between the client and the server. | ✓ | ✓ | ✓ | 3.0 |
| **18.2** | Verify that access to administration and management functions within the Web Service Application is limited to web service administrators. | ✓ | ✓ | ✓ | 3.0 |
| **18.3** | Verify that XML or JSON schema is in place and verified before accepting input. | ✓ | ✓ | ✓ | 3.0 |
| **18.4** | Verify that all input is limited to an appropriate size limit. | ✓ | ✓ | ✓ | 3.0 |
| **18.5** | Verify that SOAP based web services are compliant with Web Services-Interoperability (WS-I) Basic Profile at minimum. This essentially means TLS encryption. | ✓ | ✓ | ✓ | 3.0.1 |
| **18.7** | Verify that the REST service is protected from Cross-Site Request Forgery via the use of at least one or more of the following: double submit cookie pattern, CSRF nonces, ORIGIN request header checks, and referrer request header checks. | ✓ | ✓ | ✓ | 3.0.1 |
| **18.8** | Verify the REST service explicitly check the incoming Content-Type to be the expected one, such as application/xml or application/json. |  | ✓ | ✓ | 3.0 |
| **18.9** | Verify that the message payload is signed to ensure reliable transport between client and service, using JSON Web Signing or WS-Security for SOAP requests. |  | ✓ | ✓ | 3.0.1 |
| **18.10** | Verify that alternative and less secure access paths do not exist. |  | ✓ | ✓ | 3.0 |

## References

For more information, see also:

[OWASP Testing Guide 4.0: Configuration and Deployment Management Testing](https://www.owasp.org/index.php/Testing_for_configuration_management)

[OWASP Cross-Site Request Forgery cheat sheet](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet)

[JSON Web Tokens (and Signing)](https://jwt.io/)

# V19: Configuration Verification Requirements

## Control Objective

Ensure that a verified application has:

Up to date libraries and platform(s).

A secure by default configuration.

Sufficient hardening that user initiated changes to default configuration do not unnecessarily expose or create security weaknesses or flaws to underlying systems.

## Security Verification Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | Since |
| **19.1** | Verify that all components are up to date with proper security configuration(s) and version(s). This should include removal of unneeded configurations and folders such as sample applications, platform documentation, and default or example users. | ✓ | ✓ | ✓ | 4.0 |
| **19.2** | Verify that communications between components, such as between the application server and the database server, are encrypted, particularly when the components are in different containers or on different systems. |  | ✓ | ✓ | 4.0 |
| **19.3** | Verify that communications between components, such as between the application server and the database server, is authenticated using an account with the least necessary privileges. |  | ✓ | ✓ | 4.0 |
| **19.4** | Verify application deployments are adequately sandboxed, containerized or isolated to delay and deter attackers from attacking other applications. |  | ✓ | ✓ | 3.0 |
| **19.5** | Verify that the application build and deployment processes are performed in a secure and repeatable method, such as CI / CD automation and automated configuration management. |  | ✓ | ✓ | 4.0 |
| **19.6** | Verify that authorized administrators have the capability to verify the integrity of all security-relevant configurations to detect tampering. |  |  | ✓ | 4.0 |
| **19.7** | Verify that all application components are signed. |  |  | ✓ | 3.0 |
| **19.8** | Verify that third party components come from trusted repositories. |  |  | ✓ | 3.0 |
| **19.9** | Verify that build processes for system level languages have all security flags enabled, such as ASLR, DEP, and security checks. |  |  | ✓ | 3.0 |
| **19.10** | 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, sub-resource integrity (SRI) is used to validate the integrity of the asset. |  | ✓ | ✓ | 4.0 |
| **19.11** | Verify that all application components, services, and servers each use their own low privilege service account, that is not shared between applications nor used by administrators. |  | ✓ | ✓ | 4.0 |

## References

For more information, see also:

[OWASP Testing Guide 4.0: Configuration and Deployment Management Testing](https://www.owasp.org/index.php/Testing_for_configuration_management)

# V20: Internet of Things Verification Requirements

This section contains controls that are Embedded/IoT device specific. These controls must be taken in conjunction with all other sections of the relevant ASVS Verification Level.

## Control Objective

Embedded/IoT devices should:

Have the same level of security controls within the device as found in the server, by enforcing security controls in a trusted environment.

Sensitive data stored on the device should be done so in a secure manner using hardware backed storage such as secure elements.

All sensitive data transmitted from the device should utilize transport layer security.

## Security Verification Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | L1 | L2 | L3 | Since |
| **20.1** | Verify that application layer debugging interfaces such USB, UART, and other serial variants are disabled or protected by a complex password. | ✓ | ✓ | ✓ | 4.0 |
| **20.2** | Verify that cryptographic keys and certificats are unique to each individual device. | ✓ | ✓ | ✓ | 4.0 |
| **20.3** | Verify that memory protection controls such as ASLR and DEP are enabled by the embedded/IoT operating system, if applicable. | ✓ | ✓ | ✓ | 4.0 |
| **20.4** | Verify that on-chip debugging interfaces such as JTAG or SWD are disabled or that available protection mechanism is enabled and configured appropriately. | ✓ | ✓ | ✓ | 4.0 |
| **20.5** | Verify that physical debug headers are not present on the device. | ✓ | ✓ | ✓ | 4.0 |
| **20.6** | Verify that sensitive data is not stored unencrypted on the device. | ✓ | ✓ | ✓ | 4.0 |
| **20.7** | Verify that ARM based processors make use of TrustZone and TEE (Trusted Execution Environment) | ✓ | ✓ | ✓ | 4.0 |
| **20.8** | Verify that private keys and certificates are stored securely in a Secure Element, TPM, TEE (Trusted Execution Environment), or protected using strong cryptography. | ✓ | ✓ | ✓ | 4.0 |
| **20.9** | Verify that the firmware apps protect data-in-transit using transport layer security. | ✓ | ✓ | ✓ | 4.0 |
| **20.10** | Verify that the firmware apps validate the digital signature of server connections. | ✓ | ✓ | ✓ | 4.0 |
| **20.11** | Verify that wireless communications are mutually authenticated. | ✓ | ✓ | ✓ | 4.0 |
| **20.12** | Verify that wireless communications are sent over an encrypted channel. | ✓ | ✓ | ✓ | 4.0 |
| **20.13** | Verify that any use of banned C functions are replaced with the appropriate safe equivalent functions. | ✓ | ✓ | ✓ | 4.0 |
| **20.14** | Verify that each firmware maintains a software bill of materials cataloging third-party components, versioning, and published vulnerabilities. | ✓ | ✓ | ✓ | 4.0 |
| **20.15** | Verify all code including third-party binaries, libraries, frameworks are reviewed for hardcoded credentials (backdoors). | ✓ | ✓ | ✓ | 4.0 |
| **20.16** | Verify that the application and firmware components are not susceptible to OS Command Injection by invoking shell command wrappers, scripts, or that security controls prevent OS Command Injection. | ✓ | ✓ | ✓ | 4.0 |
| **20.17** | Verify that the firmware apps pin the digital signature to a trusted server(s). |  | ✓ | ✓ | 4.0 |
| **20.18** | Verify the presence of physical tamper resistance and/or tamper detection features, including epoxy. |  | ✓ | ✓ | 4.0 |
| **20.19** | Verify that identifying markings on chips have been removed. |  | ✓ | ✓ | 4.0 |
| **20.20** | Verify that any available Intellectual Property protection technologies provided by the chip manufacturer are enabled. |  | ✓ | ✓ | 4.0 |
| **20.21** | Verify security controls are in place to hinder firmware reverse engineering (e.g., removal of verbose debugging symbols). |  | ✓ | ✓ | 4.0 |
| **20.22** | Verify the device validates the boot image signature before loading. |  | ✓ | ✓ | 4.0 |
| **20.23** | Verify that the firmware update process is not vulnerable to time-of-check vs time-of-use attacks. |  | ✓ | ✓ | 4.0 |
| **20.24** | Verify the device uses code signing and validates firmware upgrade files before installing. |  | ✓ | ✓ | 4.0 |
| **20.25** | Verify that the device cannot be downgraded to old versions (anti-rollback) of valid firmware. |  | ✓ | ✓ | 4.0 |
| **20.26** | Verify usage of cryptographically secure pseudo-random number generator on embedded device (e.g., using chip-provided random number generators). |  | ✓ | ✓ | 4.0 |
| **20.27** | Verify that firmware has the abilityto perform automatic firmware updates upon a predefined schedule. |  | ✓ | ✓ | 4.0 |
| **20.28** | Verify that the device wipes firmware and sensitive data upon detection of tampering or receipt of invalid message. |  |  | ✓ | 4.0 |
| **20.29** | Verify that only micro controllers that support disabling debugging interfaces (e.g. JTAG, SWD) are used. |  |  | ✓ | 4.0 |
| **20.30** | Verify that only micro controllers that provide substantial protection from de-capping and side channel attacks are used. |  |  | ✓ | 4.0 |
| **20.31** | Verify that sensitive traces are not exposed to outer layers of the printed circuit board. |  |  | ✓ | 4.0 |
| **20.32** | Verify that inter-chip communication is encrypted (e.g. Main board to daughter board commnucation). |  |  | ✓ | 4.0 |
| **20.33** | Verify the device uses code signing and validates code before execution. |  |  | ✓ | 4.0 |
| **20.34** | Verify that sensitive information maintained in memory is overwritten with zeros as soon as it is no longer required. |  |  | ✓ | 4.0 |
| **20.35** | Verify that the firmware apps utilize kernel containers for isolation between apps. |  |  | ✓ | 4.0 |
| **20.36** | Verify that secure compiler flags such as -fPIE, -fstack-protector-all, -Wl,-z,noexecstack, -Wl,-z,noexecheap are configured for firmware builds. |  |  | ✓ | 4.0 |
| **20.37** | Verify that micro controllers are configured with code protection (if applicable). |  |  | ✓ | 4.0 |

## References

For more information, see also:

[OWASP Internet of Things Top 10](https://www.owasp.org/images/7/71/Internet_of_Things_Top_Ten_2014-OWASP.pdf)

[OWASP Embedded Application Security Project](https://www.owasp.org/index.php/OWASP_Embedded_Application_Security)

[OWASP Internet of Things Project](https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project)

[Trudy TCP Proxy Tool](https://github.com/praetorian-inc/trudy)

# Appendix A: Glossary

**2FA** – Two-factor authentication(2FA) adds a second level of authentication to an account log-in.

**Address Space Layout Randomization (ASLR)** – A technique to make exploiting memory corruption bugs more difficult.

**Application Security** – Application-level security focuses on the analysis of components that comprise the application layer of the Open Systems Interconnection Reference Model (OSI Model), rather than focusing on for example the underlying operating system or connected networks.

**Application Security Verification** – The technical assessment of an application against the OWASP MASVS.

**Application Security Verification Report** – A report that documents the overall results and supporting analysis produced by the verifier for a particular application.

**Authentication** – The verification of the claimed identity of an application user.

**Automated Verification** – The use of automated tools (either dynamic analysis tools, static analysis tools, or both) that use vulnerability signatures to find problems.

**Black box testing** – It is a method of software testing that examines the functionality of an application without peering into its internal structures or workings.

**Component** – a self-contained unit of code, with associated disk and network interfaces that communicates with other components.

**Cross-Site Scripting** (XSS) – A security vulnerability typically found in web applications allowing the injection of client-side scripts into content.

**Cryptographic module** – Hardware, software, and/or firmware that implements cryptographic algorithms and/or generates cryptographic keys.

**CWE** - CWE is a community-developed list of common software security weaknesses. It serves as a common language, a measuring stick for software security tools, and as a baseline for weakness identification, mitigation, and prevention efforts.

**DAST** –Dynamic application security testing (DAST) technologies are designed to detect conditions indicative of a security vulnerability in an application in its running state.

**Design Verification** – The technical assessment of the security architecture of an application.

**Dynamic Verification** – The use of automated tools that use vulnerability signatures to find problems during the execution of an application.

**Globally Unique Identifier** (GUID) – a unique reference number used as an identifier in software.

**Hyper Text Transfer Protocol** (HTTPS) – An application protocol for distributed, collaborative, hypermedia information systems. It is the foundation of data communication for the World Wide Web.

**Hardcoded keys** – Cryptographic keys which are stored on the filesystem, be it in code, comments or files.

**IPC** – Inter Process Communications,In IPC Processes communicate with each other and with the kernel to coordinate their activities.

**Input Validation** – The canonicalization and validation of untrusted user input.

**JAVA Bytecode** - Java bytecode is the instruction set of the Java virtual machine(JVM). Each bytecode is composed of one, or in some cases two bytes that represent the instruction (opcode), along with zero or more bytes for passing parameters.

**Malicious Code** – Code introduced into an application during its development unbeknownst to the application owner, which circumvents the application's intended security policy. Not the same as malware such as a virus or worm!

**Malware** – Executable code that is introduced into an application during runtime without the knowledge of the application user or administrator.

**Open Web Application Security Project** (OWASP) – The Open Web Application Security Project (OWASP) is a worldwide free and open community focused on improving the security of application software. Our mission is to make application security "visible," so that people and organizations can make informed decisions about application security risks. See: <http://www.owasp.org/>

**Personally Identifiable Information** (PII) - is information that can be used on its own or with other information to identify, contact, or locate a single person, or to identify an individual in context.

**PIE** – Position-independent executable (PIE) is a body of machine code that, being placed somewhere in the primary memory, executes properly regardless of its absolute address.

**PKI** – A PKI is an arrangement that binds public keys with respective identities of entities. The binding is established through a process of registration and issuance of certificates at and by a certificate authority (CA).

**SAST** – Static application security testing (SAST) is a set of technologies designed to analyze application source code, byte code and binaries for coding and design conditions that are indicative of security vulnerabilities. SAST solutions analyze an application from the “inside out” in a nonrunning state.

**SDLC** – Software development lifecycle.

**Security Architecture** – An abstraction of an application's design that identifies and describes where and how security controls are used, and also identifies and describes the location and sensitivity of both user and application data.

**Security Configuration** – The runtime configuration of an application that affects how security controls are used.

**Security Control** – A function or component that performs a security check (e.g. an access control check) or when called results in a security effect (e.g. generating an audit record).

**SQL Injection (SQLi)** – A code injection technique used to attack data driven applications, in which malicious SQL statements are inserted into an entry point.

**SSO Authentication** – Single Sign On(SSO) occurs when a user logs into one application and is then automatically logged into other applications without having to re-authenticate. For example when you log in in Google, you automatically login in the youtube , docs and mail service.

**Threat Modeling** - A technique consisting of developing increasingly refined security architectures to identify threat agents, security zones, security controls, and important technical and business assets.

**Transport Layer Security** – Cryptographic protocols that provide communication security over the Internet

**URI/URL/URL fragments** – A Uniform Resource Identifier is a string of characters used to identify a name or a web resource. A Uniform Resource Locator is often used as a reference to a resource.

**User acceptance testing (UAT)**– Traditionally a test environment that behaves like the production environment where all software testing is performed before going live.

**Verifier** – The person or team that is reviewing an application against the OWASP ASVS requirements.

**Whitelist** – A list of permitted data or operations, for example a list of characters that are allowed to perform input validation.

**X.509 Certificate** – An X.509 certificate is a digital certificate that uses the widely accepted international X.509 public key infrastructure (PKI) standard to verify that a public key belongs to the user, computer or service identity contained within the certificate.

# Appendix B: References

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

OWASP Mobile Security Project - <https://www.owasp.org/index.php/OWASP_Mobile_Security_Project>

OWASP Mobile Security Testing Guide - <https://www.owasp.org/index.php/OWASP_Mobile_Security_Testing_Guide>

OWASP Mobile Top 10 Risks - <https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks>

OWASP Reverse Engineering and Code Modification Prevention - <https://www.owasp.org/index.php/OWASP_Reverse_Engineering_and_Code_Modification_Prevention_Project>

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

MITRE Common Weakness Enumeration - <http://cwe.mitre.org/>

PCI Security Standards Council - <https://www.pcisecuritystandards.org>

PCI Data Security Standard (DSS) v3.0 Requirements and Security Assessment Procedures <https://www.pcisecuritystandards.org/documents/PCI_DSS_v3.pdf>