

Application Security Verification Standard 3.0

May 2015

Table of Contents

[Acknowledgements 3](#_Toc419822093)

[Version 2.1, 2015 3](#_Toc419822094)

[Version 2.0, 2014 3](#_Toc419822095)

[Version 1.0 – 2009 3](#_Toc419822096)

[Copyright and License 3](#_Toc419822097)

[Introduction 3](#_Toc419822098)

[How to Use This Standard 3](#_Toc419822099)

[Application Security Verification Levels 3](#_Toc419822100)

[Level 0: Cursory 3](#_Toc419822101)

[Level 1: Opportunistic 3](#_Toc419822102)

[Level 2: Standard 3](#_Toc419822103)

[Level 3: Advanced 3](#_Toc419822104)

[Scope of Verification 3](#_Toc419822105)

[Detailed Verification Requirements 3](#_Toc419822106)

[V2: Authentication Verification Requirements 4](#_Toc419822107)

[V3: Session Management Verification Requirements 5](#_Toc419822108)

[V4: Access Control Verification Requirements 6](#_Toc419822109)

[V5: Malicious Input Handling Verification Requirements 7](#_Toc419822110)

[V7: Cryptography at Rest Verification Requirements 8](#_Toc419822111)

[V8: Error Handling and Logging Verification Requirements 9](#_Toc419822112)

[V9: Data Protection Verification Requirements 10](#_Toc419822113)

[V10: Communications Security Verification Requirements 11](#_Toc419822114)

[V11: HTTP Security Verification Requirements 12](#_Toc419822115)

[V13: Malicious Controls Verification Requirements 13](#_Toc419822116)

[V15: Business Logic Verification Requirements 14](#_Toc419822117)

[V16: Files and Resources Verification Requirements 15](#_Toc419822118)

[V17: Mobile Verification Requirements 16](#_Toc419822119)

[Appendix A: Applying ASVS in Practice 17](#_Toc419822120)

[Appendix B: Glossary 18](#_Toc419822121)

[Appendix C: Where To Go From Here 19](#_Toc419822122)

# Acknowledgements

### Version 3.0, 2015

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Project Leads | Andrew van der Stock | Daniel Cuthbert |  |  |
| Lead authors |  |  |  |  |
| Contributors and reviewers | Boy Baukema | Ari Kesäniemi | Colin Watson | Jim Manico |
| François-Eric Guyomarc’h | Cristinel Dumitru | James Holland | Gary Robinson |
|  |  |  |  |  |
|  |  |  |  |  |

### Version 2.0, 2014

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Project Leads | Daniel Cuthbert | Sahba Kazerooni |  |  |
| Lead authors | Andrew van der Stock | Krishna Raja |  |  |
| Contributors and reviewers | Antonio Fontes | Colin Watson | Jeff Sergeant | Pekka Sillanpää |
| Archangel Cuison | Dr Emin Tatli | Jerome Athias | Safuat Hamdy |
|  | Ari Kesäniemi | Etienne Stalmans | Jim Manico | Scott Luc |
|  | Boy Baukema | Evan Gaustad | Mait Peekma | Sebastien Deleersnyder |

### Version 1.0 – 2009

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Project Leads | Mike Boberski | Jeff Williams | Dave Wichers |  |
| Authors and contributors | Andrew van der Stock | Dr. Sarbari Gupta | John Steven | Pierre Parrend |
| Barry Boyd | Dr. Thomas Braun | Ken Huang | Richard Campbell |
|  | Bedirhan Urgun | Eoin Keary | Ketan Dilipkumar Vyas | Scott Matsumoto |
|  | Colin Watson | Gaurang Shah | Liz Fong | Shouvik Bardhan |
|  | Dan Cornell | George Lawless | Mandeep Khera | Stan Wisseman |
|  | Dave Hausladen | Jeff LoSapio | Matt Presson | Stephen de Vries |
|  | Theodore Winograd | Jeremiah Grossman | Nam Nguyen | Steve Coyle |
|  | Dave van Stein | John Martin | Paul Douthit | Terrie Diaz |

## Copyright and License

Copyright © 2008 – 2015 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.

# Preface

Our biggest goal with this version of the standard was to increase adoption.

One of the major challenges of a standard such as this is that it needs to satisfy two distinct, and very different, targets: individuals who are involved in organizing or executing a software security program within their organization, and software security professionals who conduct verification of applications. While both targets seek an industry-accepted standard for verification of applications, they operate under different constraints. For example, one of the most widely voiced criticisms of ASVS 2009 standard was that it specified automated assessments as one of the levels (or sub-levels). Many large organizations see automated assessments as a point of entry into the verification hierarchy, and thus a fully automated level is a convenient concept for them. Information security professionals, however, know that the depth and breadth of such a review will depend on what technology is used to perform the scan, thus leaving too much room for interpretation of the standard. ASVS 2014 introduces a Cursory Level 0 to allow for the flexibility needed to overcome this challenge.

On a similar note, one of the main goals for this version of the standard was to focus on the "what" and not the "how". Whereas the previous version of the standard talked about dynamic scanning, static analysis, Threat Modeling, and design reviews, you will notice that such terms do not appear in this version of the standard. Instead, we essentially define security requirements that must be verified for an application to achieve a certain level. How those requirements are verified is left up to the verifier.

Another major challenge that we overcame is to clearly separate requirements from design from scope. The previous version of the standard did not clearly distinguish between these concepts, leaving room for confusion. In this version, Level 3 is where design considerations are introduced and clearly separated from detailed verification requirements. Furthermore, we have now separated out the concept of scope completely – the new (+) notation allows for a verifier to optionally include third party components and frameworks in their review.

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.

# Introduction

The primary aim of the OWASP Application Security Verification Standard (ASVS) is to normalize the range in the coverage and level of rigor available in the market when it comes to performing web application security verification.

The Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to develop, purchase, and maintain applications that can be trusted. All of the OWASP tools, documents, forums, and chapters are free and open to anyone interested in improving application security. We advocate approaching application security as a people, process, and technology problem, because the most effective approaches to application security include improvements in all of these areas. We can be found at www.owasp.org.

OWASP is a new kind of organization. Our freedom from commercial pressures allows us to provide unbiased, practical, cost-effective information about application security. OWASP is not affiliated with any technology company, although we support the informed use of commercial security technology. Similar to many open-source software projects, OWASP produces many types of materials in a collaborative, open way. The OWASP Foundation is a not-for-profit entity that ensures the project’s long-term success.

The ASVS standard provides a basis for verifying application technical security controls, as well as any technical security controls in the environment that are relied on to protect against vulnerabilities such as Cross-Site Scripting (XSS) and SQL injection. This standard can be used to establish a level of confidence in the security of Web applications

## How to Use This Standard

The ASVS standard can be used by both consumers and service or tool providers.

ASVS has two main goals, as depicted in the figure below: to help organization’s develop and maintain secure applications; and to allow security service/tools providers and consumers to align their requirements and offerings.

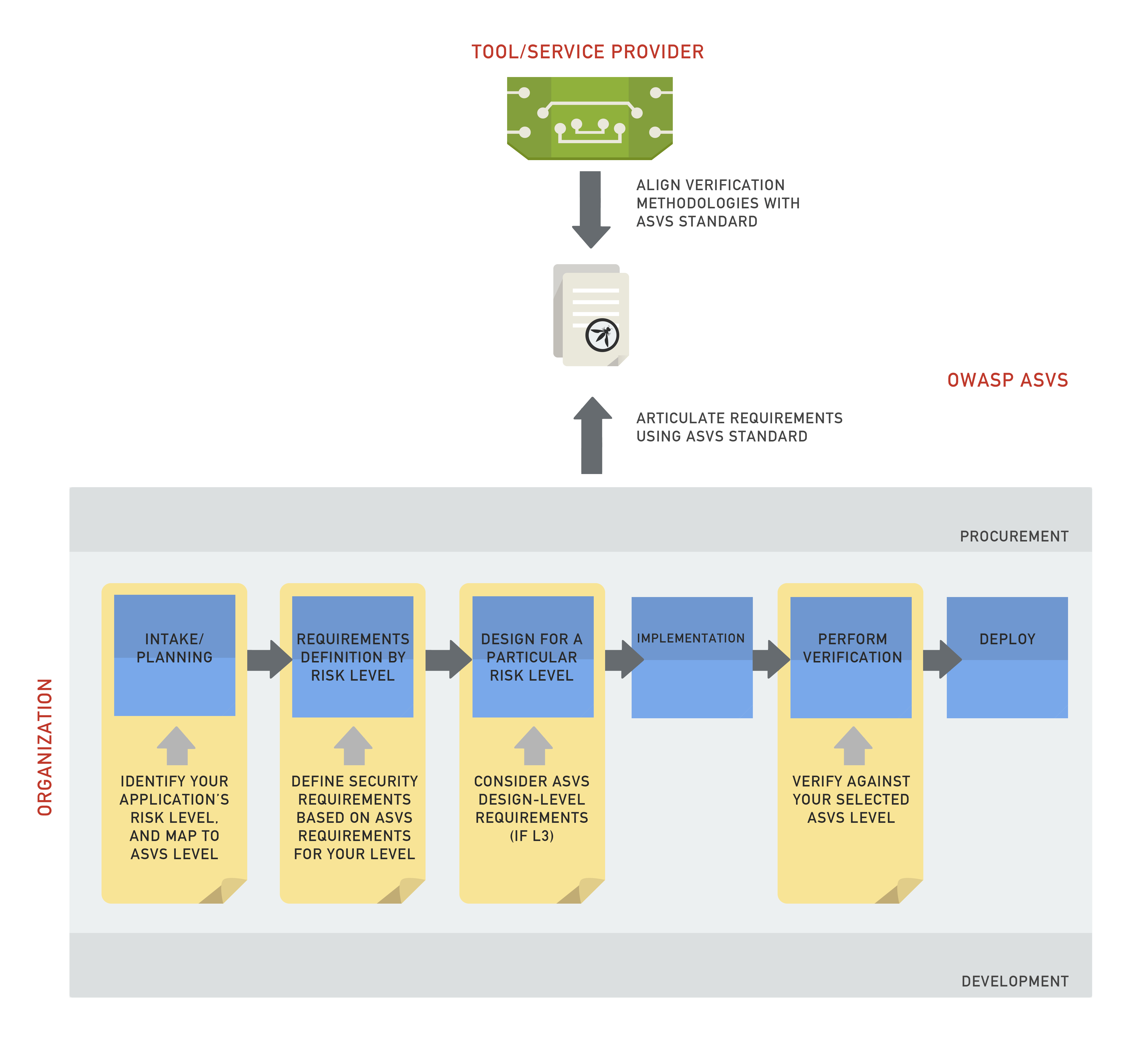


Figure 1 - Uses of ASVS for organizations and tool/service providers

The example scenarios below further demonstrate the common use cases of ASVS using a fictional organization (ACME Bank) and a fictional security services firm (Hack All the Things).

### Use Case 1: Certification of Applications

ACME Bank has developed a new Internet Banking portal. which is due to be deployed into their production environment. The application has followed the bank's SDLC process and should be in a secure state. The internal security team at ACME Bank has been tasked to ensure that once deployed into the production environment. it does not pose a risk to other applications. due to it being hosted on a shared platform and database. After an internal threat modelling exercise was performed. it was agreed that the application had a high-risk associated with it and the data stored within it.

The team makes use of a well-known web application scanning tool and start the process of mapping out the application in preparation for the automated scanning phase. Once complete. the automated scanning tool is started and left to complete. Once the report has been generated. the security analyst tests for false positives (such as SQL injection. or X55) and amends the report as necessary. Any findings discovered are reported back to the system owners and development team. in order to be rectified. Once this has been completed. the re-test of the application is performed to ensure they have been resolved in a suitable manner.

in this example. using the ASVS could allow the internal team to test for common application flaws as well as verify that it had been developed in accordance to the bank's security standard.

### Use Case 2 - Alignment of testing methodology

Hack All the Things (HATT) is a penetration-testing consultancy. whose main area of expertise is performing application security assessments for clients at an infrastructure and application level. They have decided to align their internal testing methodology with that of the OWASP ASVS to offer their clients peace of mind when performing assessments.

In order to achieve this. all staff ate required to manually test the application in question using the detailed verification requirements. as outlined by the ASVS document.

In this instance, adopting the ASVS allows HATT to offer a series of application assessments based on the three ASVS levels. and at the same time. allowing clients to understand what has been assessed.

### Use Case 3: Selection of external supplier

ACME Bank has finally completed all development on their new Internet Banking portal and the banking regulators require them to have an external consultancy perform an assessment of the application to ensure it meets the regulatory requirements with regards to security.

ACME Bank has chosen a supplier from their list of preferred suppliers and asked MATT to perform an assessment. ACME Bank supplied the consultancy with all the source code and documentation and scheduled the assessment. The external test was conducted in a phased approach. with a hilly-automated static analysis code review performed on the source code alongside a manual application security assessment. In addition. business logic was mated to ensure that the application performed as expected. as outlined in the functional specification documentation supplied. Once the assessment was complete. a report was created and delivered to ACME Bank staff.

By both parties adopting the ASVS during this process, the suitable level was chosen and tested for. As a result, both ACME bank and HATT were in sync with what had to be achieved and what the required outcome was.

## What skills do you need to use the ASVS?

I think that is a very good question that we actually should focus more on later editions of the ASVS.

There are at least these things you need to consider:

what are the most essential pieces of general knowledge the team must have, e.g. understanding the concept of injection or validation

what coding principles are followed and why (defensive coding, complying with external or internal architecture standards etc — this of course varies but typically a team should have some guiding security and implementation principles, some of which are relatively universal)

what does the team need to understand about the technology stack they’re using (and I think this is very important thing), that is, how does it work

what parts of the ASVS are handled already by the technology stack (also an important thing to consider)

In practice, I would think that at the very least, the team needs to understand how HTTP works, how things like HTTP parameters and requests get handled in their application, and what an injection is (as a general concept, not just XSS and SQLi). Also they should have a clear concept how authorisation is supposed to work in the application and be able to validate that with the requirements. I’m less concerned about authentication, as it typically is handled by an external component and just kind of plugged in to the application.

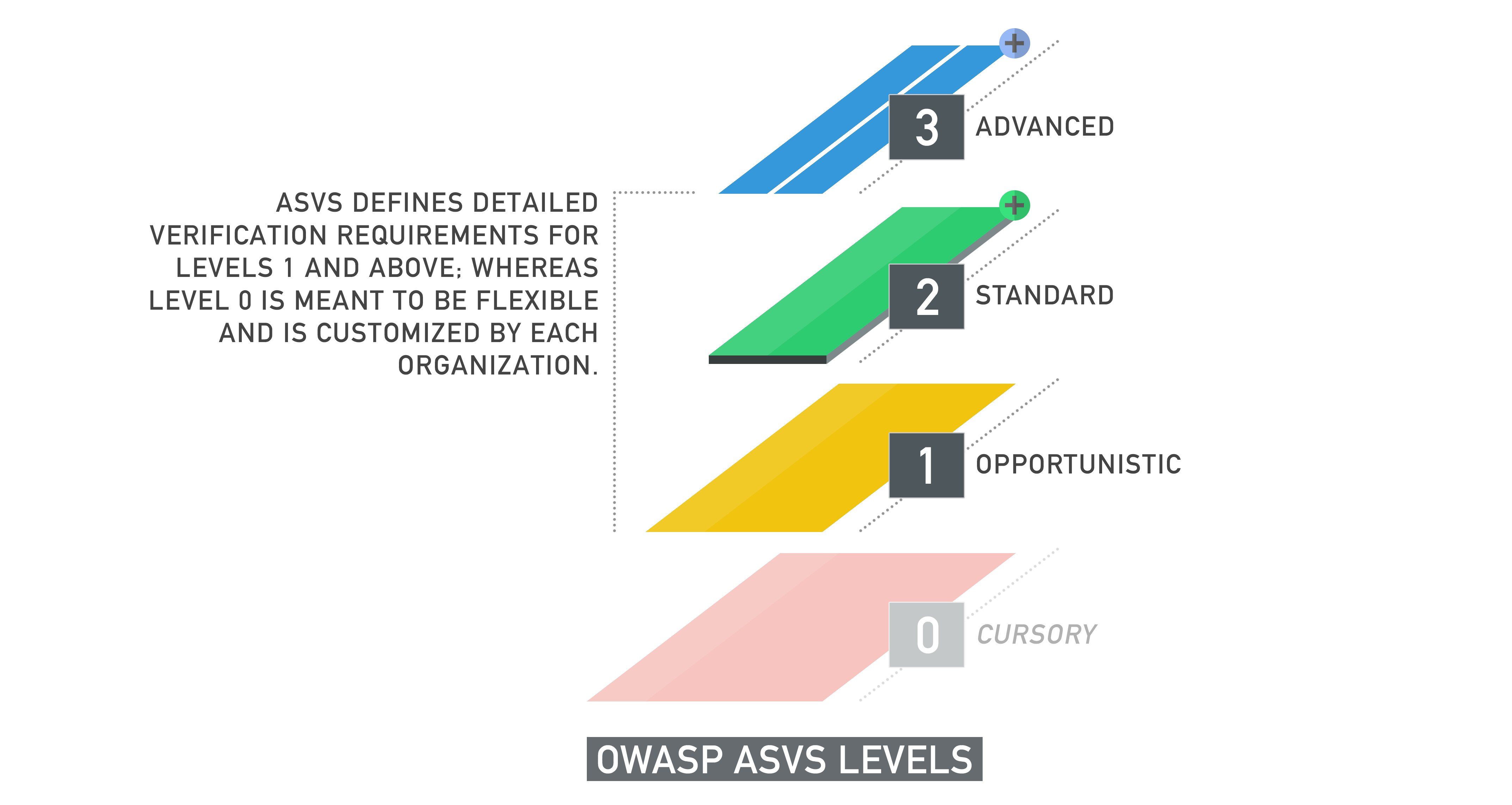
I would put less emphasis on for example session handling (unless you do that in your application, which you shouldn’t) and cryptography (rarely needed). Proper configuration of cookies, secured connections etc. are important, but luckily they can be relatively easily fixed if the team doesn’t get them right already during development.

As for ASVS, would it make sense to somehow categorize verification items based on where they are (or should be) handled, e.g. infrastructure, middleware, program code, centralised libraries etc? Or maybe as part of the ASVS itself, but as a supplemental guide? I can help with this, although it is obvious that there’s no one single categorisation as it depends on what technology is used and how. But I still assert that typically a software development team should only consider a subset of the verification items, while other items are considered by other teams (e.g. infra).

## Application Security Verification Levels

The ASVS defines three levels of verification, with each level increasing in depth as the verification moves up the levels.

The depth is defined in each level by a set of security verification requirements that must be addressed (these are included in the requirements tables towards the end of this document). It is a verifier’s responsibility to determine if a target of verification (TOV) meets all of the requirements at the level targeted by a review. If the application meets all of the requirements for that level, then it can be considered an OWASP ASVS Level N application, where N is the verification level that application complied with. If the application does not meet all the requirements for a particular level, but does meet all the requirements for a lower level of this standard, then it can be considered to have passed the lower level of verification.



The breadth of the verification is defined by what parts of the application are reviewed for each security requirement. For example, the scope of the review may go beyond the application’s custom-built code and include external components. Achieving a verification level under such scrutiny can be represented by annotating a “+” symbol to the verification level.

### Level 0: Cursory

Level 0 (or Cursory) is an optional certification, indicating that the application has passed some type of verification.

Level 0 is designed to be a flexible point of entry into the verification hierarchy; it indicates that some type of review has been done on the application. The detailed verification requirements are not provided by ASVS. Instead, organizations can define their own minimum criteria (such as automated runtime scan, or strong authentication mechanism).

This level is most appropriate for organizations that have a large number of applications, and where a low cost point of entry may be required. One organization may use Level 0 to require a cursory automated scan of all of their external facing applications using the organization’s commercial tool of choice; whereas another organization may define L0 requirements using data from a recent breach.

Unlike the other ASVS levels, Level 0 is not a prerequisite for other levels - an application can jump straight to Level 1 without achieving Level 0 certification (if L0 is not defined by the organization).

When defining Level 0 requirements, it is advised that each requirement be documented in a similar manner to the Detailed Verification Requirements in this document – clear, distinct, realistic, and verifiable.

#### Overview of Verification Requirements

### Level 1: Opportunistic

An application achieves Level 1 (or Opportunistic) certification if it adequately defends against application security vulnerabilities that are easy to discover.

The specific set of vulnerabilities against which Level 1 verification is measured is detailed in the Detailed Verification Requirements, but typically includes vulnerabilities that a verifier can identify with minimal-to-low effort. As such, this level cannot be considered a thorough inspection or verification of the application, but more of a quick inspection.

Level 1 is typically appropriate for applications where some confidence in the correct use of security controls is required, or to provide a quick sweep of a fleet of enterprise applications, to assist in developing a roadmap for more thorough inspections at a later date.

Threats to the application will most likely be from attackers who are using simple 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.

#### Overview of Verification Requirements

### Level 2: Standard

An application achieves Level 2 (or Standard) verification if it also adequately defends against prevalent application security vulnerabilities whose existence poses moderate-to-serious risk.

The specific set of vulnerabilities against which Level 2 verification is measured is detailed in the Detailed Verification Requirements, but would include OWASP Top 10 vulnerabilities and business logic vulnerabilities.

Level 2 ensures that evaluated security controls are in place, effective, and used as needed within the application to enforce application-specific policies.

Level 2 represents an industry standard for which the majority of an organization’s sensitive applications would strive. 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 security will typically be opportunists and possibly determined attackers (skilled and motivated attackers focusing on specific targets using purpose-built scanning tools as well as manual testing techniques).

#### Overview of Level 2 Verification Requirements

Level 2 Applications are assessed according to the following criteria:

L2.1 Controls are assessed and determined to be in place, in use and effective

L2.2 Controls are assessed such that false negatives and false positives are removed from any automated results

### Level 3: Advanced

An application achieves Level 3 (or Advanced) certification if it also adequately defends against all advanced application security vulnerabilities, and also demonstrates principles of good security design. Level 3 is at the very least, is a source code review. The best method of reviewing an application at Level 3 is with both the code and the source code.

The specific set of vulnerabilities against which Level 3 verification is measured is detailed in the Detailed Verification Requirements, but would include more difficult to exploit vulnerabilities, which would most likely be exploited by determined attackers.

Level 3 is the only ASVS level which also requires an inspection of the application’s design. In addition, the following requirements were added:

* Any major security controls which have a cross-cutting impact (such as input validation or authorization) should be implemented in a centralized manner.
* Security controls that perform validation should make decisions using a whitelist (“positive”) approach.
* Input validation should not be relied on as the only defense against injection and scripting vulnerabilities. Rather, input validation should always be the second line of defense, with parameterization and output encoding being the primaries, respectively.

Level 3 verification is typically appropriate for critical applications that protect life and safety, critical infrastructure, or defense functions or have the potential of facilitating substantial damage to the organization. Level 3 may also be appropriate for applications that process sensitive assets.

Threats to security will be from determined attackers (skilled and motivated attackers focusing on specific targets using tools including purpose-built scanning tools).

#### Overview of Level 3 Verification Requirements

L3.1 Application is assessed according to the Level 3 requirements in each of the applicable detailed verification sections

L3.2 Application is verified that implementation of all security controls adhere to the following leading practices:

* Security controls that perform validation make decisions using a positive (“whitelist”) approach
* Data validation controls are complemented by contextually aware output encoding
* All untrusted data that is sent to database layer interpreters either use parameterized interfaces, object relational models not subject to SQL injection, or are otherwise rendered safe for the data access layer.

L3.3 Application reviews are primarily manual with automated assistance for coverage and identification purposes. Controls should be assessed to the following increased minima:

* Consider if identified security controls are designed to be fit for purpose or could be improved
* Security controls are in place, in use and effective
* Security controls are centralized within the application
* All automated results are reviewed in detail, ensuring that coverage is sufficient well above sampling approaches, and that false positive and negatives results are investigated via inspection and/or discussion with the project team

## Scope of Verification

The scope of the verification is separate from the requirements for achieving a level.

By default, ASVS assumes that the scope of the verification includes all code that was developed or modified in order to create the application or release. However, one may decide to include as part of verification the code for all third-party frameworks, libraries, and service security functionality that is invoked by or supports the security of the application. Achieving a verification level under such scrutiny can be represented by annotating a “+” symbol to the verification level. For example, an application may be labelled as ASVS L3+ certified.

Including third party components is optional and is not required to achieve to any ASVS level. Such level of scrutiny may be suitable for highly sensitive or mission critical applications. As such, (+) certification will in most cases be associated with Level 3.

When third party components are included in the verification, it is not required that all detailed verification requirements be applied to third party components. In fact, most detailed verification requirements will not be applicable to third party components and can thus be checked against the base code only. Detailed verification requirements must be verified against the application’s base code, and they are verified against third party components if applicable. Only then can an application achieve the (+) certification for that level.

# Detailed Verification Requirements

V1. Architecture, design and threat modelling

V2. Authentication

V3. Session management

V4. Access control

V5. Malicious input handling

V7. Cryptography at rest

V8. Error handling and logging

V9. Data protection

V10. Communications

V11. HTTP security configuration

V13. Malicious controls

V15. Business logic

V16. File and resources

V17. Mobile

V18. Web services (NEW for 3.0)

V19. Configuration (NEW for 3.0)

V20. Client side Security (NEW for 3.0)

# V1: Architecture, design and threat modelling

## Control objective

TBA

Note: This section has been re-introduced in version 3.0, but is essentially the same architectural controls as version 1.0 of the ASVS.

## Requirements

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
| V1.1 | Verify that all application components (either individual or groups of source files, libraries, and/or executables) that are present in the application are identified. | ✓ | ✓ | ✓ | 1.0 |
| V1.2 | Verify that all components that are not part of the application but that the application relies on to operate are identified. |  | ✓ | ✓ | 1.0 |
| V1.3 | Verify that a high-level architecture for the application has been defined. |  | ✓ | ✓ | 1.0 |
| V1.4 | Verify that all application components are defined in terms of the business functions and/or security functions they provide. |  |  | ✓ | 1.0 |
| V1.5 | Verify that all components that are not part of the application but that the application relies on to operate are defined in terms of the business functions and/or security functions they provide. |  |  | ✓ | 1.0 |
| V1.6 | Verify that threat modelling information has been provided. |  |  | ✓ | 1.0 |

## References

TBA

# V2: Authentication Verification Requirements

## Control objective

TBA

## Requirements

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
| V2.1 | Verify all pages and resources require authentication except those specifically intended to be public (Principle of complete mediation). | ✓ | ✓ | ✓ | 1.0 |
| V2.2 | Verify all password fields do not echo the user’s password when it is entered. | ✓ | ✓ | ✓ | 1.0 |
| V2.3 | Deprecated |  |  |  | 2.0 |
| V2.4 | Verify all authentication controls are enforced on the server side. | ✓ | ✓ | ✓ | 1.0 |
| V2.5 | Verify all authentication controls (including libraries that call external authentication services) have a centralized implementation. |  |  | ✓ | 1.0 |
| V2.6 | Verify all authentication controls fail securely to ensure attackers cannot log in. | ✓ | ✓ | ✓ | 1.0 |
| V2.7 | Verify password entry fields allow or encourage the use of passphrases, and do not prevent long passphrases or highly complex passwords being entered |  | ✓ | ✓ | 1.0 |
| V2.8 | Verify all account identity authentication functions (such as update profile, forgot password, disabled / lost token, help desk or IVR) that might regain access to the account are at least as resistant to attack as the primary authentication mechanism. |  | ✓ | ✓ | 1.0 |
| V2.9 | Verify that the changing password functionality includes the old password, the new password, and a password confirmation, as well as a passphrase strength indication to encourage the adoption of strong passphrases. |  | ✓ | ✓ | 1.0 |
| V2.10 | Deprecated |  |  |  | 2.0 |
| V2.11 | Deprecated |  |  |  | 2.0 |
| V2.12 | Verify that all authentication decisions are logged. This should include requests with missing required information, needed for security investigations. |  | ✓ | ✓ | 1.0 |
| V2.13 | Verify that account passwords are protected using an adaptive key derivation function, salted using a salt that is unique to that account (e.g., internal user ID, account creation) and use bcrypt, scrypt or PBKDF2 before storing the password, with a minimum work factor iteration count of 150,000 loops to eliminate the possibility of brute forcing. |  | ✓ | ✓ | 3.0 |
| V2.14 | Deprecated |  |  |  | 2.0 |
| V2.15 | Deprecated |  |  |  | 2.0 |
| V2.16 | Verify that credentials, and all other identity information handled by the application(s), do not traverse unencrypted or weakly encrypted links, and that the login landing page or login form is sent over an encrypted link. | ✓ | ✓ | ✓ | 2.0 |
| V2.17 | Verify that the forgotten password function and other recovery paths do not reveal the current password and that the new password is not sent in clear text to the user. | ✓ | ✓ | ✓ | 2.0 |
| V2.18 | Verify that username enumeration is not possible via login, password reset, or forgot account functionality if usernames are considered private. | ✓ | ✓ | ✓ | 2.0 |
| V2.19 | Verify there are no default passwords in use for the application framework or any components used by the application (such as “admin/password”). | ✓ | ✓ | ✓ | 2.0 |
| V2.20 | Verify that request throttling is in place to prevent automated attacks against common authentication attacks such as brute force attacks or denial of service attacks. |  | ✓ | ✓ | 2.0 |
| V2.21 | Verify that all authentication credentials for accessing services external to the application are encrypted and stored in a protected location |  | ✓ | ✓ | 2.0 |
| V2.22 | Verify that forgot password and other recovery paths uses a soft token, mobile push, or an offline recovery mechanism. |  | ✓ | ✓ | 2.0 |
| V2.23 | Verify that account lockout is divided into soft and hard lock status, and these are not mutually exclusive. If an account is temporarily soft locked out due to a brute force attack, this should not reset the hard lock status. |  | ✓ | ✓ | 2.0 |
| V2.24 | Verify that if knowledge based questions (also known as "secret questions") are required, the questions should be strong enough to protect the application. |  | ✓ | ✓ | 2.0 |
| V2.25 | Verify that the system can be configured to disallow the use of a configurable number of previous passwords. |  | ✓ | ✓ | 2.0 |
| V2.26 | Verify re-authentication, step up or adaptive authentication, two factor authentication, or transaction signing is required before any application-specific sensitive operations are permitted as per the risk profile of the application. |  |  | ✓ | 2.0 |
| v2.27 | Verify that the use of commonly chosen passwords and weak passphrases (such as “let me in” or “Password1!”) are in place. |  | ✓ | ✓ | 3.0 |
| V2.28 | Verify that all authentication challenges, whether successful or failed, should respond in the same average response time. |  |  | ✓ | 3.0 |
| V2.29 | Verify that secrets, API keys, and passwords are not included in the source code, or end up within the resulting binary. |  |  | ✓ | 3.0 |

## References

Password storage cheat sheet <https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet>

Forgot password cheat sheet <https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet>

Choosing and Using Security Questions at <https://www.owasp.org/index.php/Choosing_and_Using_Security_Questions_Cheat_Sheet>

(See LoA)

# V3: Session Management Verification Requirements

## Control objective

TBA

## Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | **Description** | **1** | **2** | **3** | **Since** |
| V3.1 | Verify that there is no custom session manager, or that the custom session manager is resistant against all common session management attacks. | ✓ | ✓ | ✓ | 1.0 |
| V3.2 | Verify that sessions are invalidated when the user logs out. | ✓ | ✓ | ✓ | 1.0 |
| V3.3 | Verify that sessions timeout after a specified period of inactivity. | ✓ | ✓ | ✓ | 1.0 |
| V3.4 | Verify that sessions timeout after an administratively-configurable maximum time period regardless of activity (an absolute timeout). |  | ✓ | ✓ | 1.0 |
| V3.5 | Verify that all pages that require authentication have easy and visible access to logout functionality. | ✓ | ✓ | ✓ | 1.0 |
| V3.6 | Verify that the session id is never disclosed other than in cookie headers; particularly in URLs, error messages, or logs. This includes verifying that the application does not support URL rewriting of session cookies. | ✓ | ✓ | ✓ | 1.0 |
| V3.7 | Verify that all successful authentication and re-authentication generates a new session and session id. | ✓ | ✓ | ✓ | 1.0 |
| V3.8 | Deprecated |  |  |  | 3.0 |
| V3.9 | Deprecated |  |  |  | 3.0 |
| V3.10 | Verify that only session ids generated by the application framework are recognized as active by the application. |  | ✓ | ✓ | 1.0 |
| V3.11 | Verify that session ids are sufficiently long, random and unique across the correct active session base. |  | ✓ | ✓ | 1.0 |
| V3.12 | Verify that authenticated session ids using cookies have their path set to an appropriately restrictive value for that site. The domain cookie attribute restriction should not be set unless for a business requirement, such as single sign on. (consider dropping second sentence) |  | ✓ | ✓ | 1.0 |
| V3.13 | Deprecated |  |  |  | 2.0 |
| V3.14 | Verify that authenticated session tokens using cookies are protected by the use of "HttpOnly". | ✓ | ✓ | ✓ | 1.0 |
| V3.15 | Verify that authenticated session tokens using cookies are protected with the "secure" attribute and a strict transport security header (such as Strict-Transport-Security: max-age=60000; includeSubDomains) are present. \*\*\* Remove HSTS header requirement, make it a separate item in data protection section... | ✓ | ✓ | ✓ | 1.0 |
| V3.16 | Deprecated |  |  |  | 3.0 |

## References

TBA

# V4: Access Control Verification Requirements

## Control objective

TBA

## Requirements

| **#** | **Description** | **1** | **2** | **3** | **Since** |
| --- | --- | --- | --- | --- | --- |
| V4.1 | Verify that the principle of least privilege - 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 |
| V4.2 | 3is ication rity requirementscation Standard 2.0. cross-site scripting. Deprecated |  |  |  | 3.0 |
| V4.3 | Deprecated |  |  |  | 3.0 |
| V4.4 | Verify that access to sensitive records is protected, such that only authorized objects or data is accessible to each user (for example, protect against users tampering with a parameter to see or alter another user's account). | ✓ | ✓ | ✓ | 1.0 |
| V4.5 | Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow file or directory metadata, such as Thumbs.db, .DS\_Store, .git or .svn folders. | ✓ | ✓ | ✓ | 1.0 |
| V4.8 | Verify that access controls fail securely. | ✓ | ✓ | ✓ | 1.0 |
| V4.9 | Verify that the same access control rules implied by the presentation layer are enforced on the server side. |  | ✓ | ✓ | 1.0 |
| V4.10 | 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 |
| V4.11 | Verify that access controls are enforced on the server side. | ✓ | ✓ | ✓ | 1.0 |
| V4.12 | Verify that there is a centralized mechanism (including libraries that call external authorization services) for protecting access to each type of protected resource. |  |  | ✓ | 1.0 |
| V4.13 | Deprecated |  |  |  | 2.0 |
| V4.14 | Verify that all access control decisions can be logged and all failed decisions are logged. |  | ✓ | ✓ | 2.0 |
| V4.16 | Verify that the application or framework uses strong random anti-CSRF tokens or has another transaction protection mechanism. | ✓ | ✓ | ✓ | 2.0 |
| V4.17 | Verify the system can protect against aggregate or continuous access of secured functions, resources, or data. For example, consider the use of a resource governor to limit the number of edits per hour or to prevent the entire database from being scraped by an individual user. |  | ✓ | ✓ | 2.0 |
| V4.18 | 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 |

## References

TBA

# V5: Malicious input handling verification requirements

## Control objective

TBA

## Requirements

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
| V5.1 | Verify that the runtime environment is not susceptible to buffer overflows, or that security controls prevent buffer overflows. | ✓ | ✓ | ✓ | 1.0 |
| V5.2 | Deprecated |  |  |  | 2.0 |
| V5.3 | Verify that server side input validation failures result in request rejection, and are logged | ✓ | ✓ | ✓ | 1.0 |
| V5.4 | Deprecated |  |  |  | 3.0 |
| V5.5 | Verify that input validation routines are enforced on the server side. | ✓ | ✓ | ✓ | 1.0 |
| V5.6 | Verify that a single input validation control is used by the application for each type of data that is accepted. |  |  | ✓ | 1.0 |
| V5.7 | Deprecated |  |  |  | 3.0 |
| V5.8 | Deprecated |  |  |  | 3.0 |
| V5.9 | Deprecated |  |  |  | 2.0 |
| V5.10 | Verify that all SQL queries, HQL, OSQL, stored procedures, calling of stored procedures are protected by the use of prepared statements or query parameterization. | ✓ | ✓ | ✓ | 2.0 |
| V5.11 | Verify that the application is not susceptible to LDAP Injection, or that security controls prevent LDAP Injection. | ✓ | ✓ | ✓ | 2.0 |
| V5.12 | Verify that the application is not susceptible to OS Command Injection, or that security controls prevent OS Command Injection. | ✓ | ✓ | ✓ | 2.0 |
| V5.13 | Verify that the application is not susceptible to common XML attacks, such as XPath query tampering, XML External Entity attacks, and XML injection attacks. | ✓ | ✓ | ✓ | 2.0 |
| V5.14 | Deprecated |  |  |  | 3.0 |
| V5.15 | Deprecated |  |  |  | 2.0 |
| V5.16 | Verify that data that is output to HTML (including HTML elements, HTML attributes, JavaScript data values, CSS blocks and style parameters, and URL fragments) are properly escaped for the applicable context. | ✓ | ✓ | ✓ | 2.0 |
| V5.17 | If the application framework allows automatic mass parameter assignment (also called automatic variable binding) from the inbound request to a model, verify that security sensitive fields such as “accountBalance”, “role” or “password” are protected from malicious automatic binding. |  | ✓ | ✓ | 2.0 |
| V5.18 | Deprecated |  |  |  | 3.0 |
| V5.19 | Verify that for each type of output encoding/escaping performed by the application, there is a single security control for that type of output for the intended destination. |  |  | ✓ | 2.0 |
| V5.20 | Verify that client side validation is used as a second line of defence, in addition to server side validation |  | ✓ | ✓ | 3.0 |
| V5.21 | "Verify that applications have defined positive validation for all input fields, with permissible exceptions to allow characters that might otherwise be rejected. |  | ✓ | ✓ | 3.0 |

## References

LDAP Injection Cheat Sheet  
<https://www.owasp.org/index.php/LDAP_Injection_Prevention_Cheat_Sheet>

XML vulnerabilities in Python  
<https://pypi.python.org/pypi/defusedxml#how-to-avoid-xml-vulnerabilities>

Cross site scripting cheat sheet <https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet>

# V6: Output encoding / escaping

This section was incorporated into V5 in Application Security Verification Standard 2.0.

# V7: Cryptography at rest verification requirements

## Control objective

The

## Requirements

| **#** | **Description** | **1** | **2** | **3** | **Since** |
| --- | --- | --- | --- | --- | --- |
| **V7.1** | Deprecated |  |  |  | 3.0 |
| **V7.2** | Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable oracle padding. |  | ✓ | ✓ | 1.0 |
| **V7.3** | Deprecated |  |  |  | 3.0 |
| **V7.4** | Deprecated |  |  |  | 2.0 |
| **V7.5** | Deprecated |  |  |  | 2.0 |
| **V7.6** | 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 unguessable by an attacker. |  | ✓ | ✓ | 1.0 |
| **V7.7** | Verify that cryptographic algorithms used by the application have been validated against FIPS 140-2 or an equivalent standard. | ✓ | ✓ | ✓ | 1.0 |
| **V7.8** | Verify that cryptographic modules operate in their approved mode according to their published security policies. |  |  | ✓ | 1.0 |
| **V7.9** | Verify that there is an explicit policy for how cryptographic keys are managed (e.g., generated, distributed, revoked, expired). Verify that this key lifecycle is properly enforced. |  | ✓ | ✓ | 1.0 |
| **V7.10** | Deprecated |  |  |  | 2.0 |
| **V7.11** | Verify that all consumers of cryptographic services do not have direct access to key material. Isolate cryptographic processes, including master secrets and consider the use of a hardware key vault (HSM). |  |  | ✓ | 3.0 |
| **V7.12** | Only store sensitive data that you need to store. | ✓ | ✓ | ✓ | 3.0 |
| **V7.13** | PII shall not be stored in the clear nor communicated in the clear within unprotected communication channels. |  | ✓ | ✓ | 3.0 |
| **V7.14** | Verify that where possible, keys and secrets are zeroed when destroyed |  | ✓ | ✓ | 3.0 |
| **V7.15** | Verify that all keys and passwords are replaceable, and are generated or replaced at installation time |  | ✓ | ✓ | 3.0 |

## References

TBA

# V8: Error handling and logging verification requirements

## Control objective

TBA

## Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | 1 | 2 | 3 | Since |
| V8.1 | Verify that the application does not output error messages or stack traces containing sensitive data that could assist an attacker, including session id and personal information. | ✓ | ✓ | ✓ | 1.0 |
| V8.2 | Deprecated |  |  |  | 3.0 |
| V8.3 | Deprecated |  |  |  | 3.0 |
| V8.4 | Verify that error handling logic in security controls denies access by default. |  | ✓ | ✓ | 1.0 |
| V8.5 | Verify security logging controls provide the ability to log success and particularly failure events that are identified as security-relevant. |  | ✓ | ✓ | 1.0 |
| V8.6 | Verify that each log event includes necessary information that would allow for a detailed investigation of the timeline when an event happens. |  | ✓ | ✓ | 1.0 |
| V8.7 | Verify that all events that include untrusted data will not execute as code in the intended log viewing software. |  |  | ✓ | 1.0 |
| V8.8 | Verify that security logs are protected from unauthorized access and modification. |  | ✓ | ✓ | 1.0 |
| V8.9 | Deprecated |  |  |  | 3.0 |
| V8.10 | Verify that the application does not log sensitive data that could assist an attacker, including user’s session identifiers and personal or sensitive information. The length and existence of sensitive data can be logged. | ✓ | ✓ | ✓ | 1.0 |
| V8.11 | Deprecated |  |  |  | 3.0 |
| V8.12 | Deprecated |  |  |  | 2.0 |
| V8.13 | Verify that all non-printable symbols and field separators are properly encoded in log entries, to prevent log injection. |  |  | ✓ | 2.0 |
| V8.14 | Verify that log fields from trusted and untrusted sources are distinguishable in log entries. |  |  | ✓ | 2.0 |
| V8.15 | Deprecated |  |  |  | 3.0 |
| V8.16 | Verify that an audit log or similar allows for non-repudiation of key transactions. |  | ✓ | ✓ | 3.0 |
| V8.17 | Verify that security logs have some form of integrity checking or controls to prevent unauthorized modification. |  |  | ✓ | 3.0 |

## References

TBA

# V9: Data protection verification requirements

## Control objective

TBA: Treatment plans

TBA: There is an unstated assumption in the Data Protection chapter that data protection is occurring on a trusted device. This is not always true.

I feel that we either need to make it clear that there is an assumption, or make it a requirement for Levels 2 and 3 that Data Protection is enforced on a trusted device, much in the same way as all other chapters.

## Requirements

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
| V9.1 | Verify that all forms containing sensitive information have disabled client side caching, including autocomplete features. | ✓ | ✓ | ✓ | 1.0 |
| V9.2 | Verify that the list of sensitive data processed by the application is identified, and that there is an explicit policy for how access to this data must be controlled, encrypted and enforced under relevant data protection directives | ✓ | ✓ | ✓ | 1.0 |
| V9.3 | Verify that all sensitive data is sent to the server in the HTTP message body or headers (i.e., URL parameters are never used to send sensitive data). | ✓ | ✓ | ✓ | 1.0 |
| V9.4 | Verify that the application sets appropriate anti-caching headers as per the risk of the application, such as the following:  Expires: Tue, 03 Jul 2001 06:00:00 GMT Last-Modified: {now} GMT Cache-Control: no-store, no-cache, must-revalidate, max-age=0 Cache-Control: post-check=0, pre-check=0 Pragma: no-cache |  | ✓ | ✓ | 1.0 |
| V9.5 | 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 |
| V9.6 | Verify that there is a method to remove each type of sensitive data from the application at the end of the required retention policy. |  |  | ✓ | 1.0 |
| V9.7 | Verify the application minimizes the number of parameters in a request, such as hidden fields, Ajax variables, cookies and header values. |  | ✓ | ✓ | 2.0 |
| V9.8 | Verify the application has the ability to detect and alert on abnormal numbers of requests for information or processing high value transactions for that user role, such as screen scraping, automated use of web service extraction, or data loss prevention. For example, the average user should not be able to access more than 5 records per hour or 30 records per day, or add 10 friends to a social network per minute. This might be achieved using an application attack and response engine, web application firewall, OWASP AppSensor, or custom logic in the controller. |  |  | ✓ | 2.0 |
| V9.9 | Verify that data stored in client side storage, such as HTML5 local storage or Flash cookies, does not contain sensitive or peronally identified data. |  | ✓ | ✓ | 3.0 |
| V9.10 | 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 |
| V9.11 | Verify that sensitive data or primary keys, such as personally identifiable information or credit cards are anonymized, masked or truncated on the server before transmission to the client | ✓ | ✓ | ✓ | 3.0 |
| V9.12 | Verify that sensitive data is rapidly sanitized from memory as soon as it is no longer needed and handled in accordance to functions and techniques supported by the framework/library/operating system. |  |  | ✓ | 3.0 |

## References

TBA

# V10: Communications security verification requirements

## Control objective

TBA

## Requirements

| **#** | **Description** | **1** | **2** | **3** | **Since** |
| --- | --- | --- | --- | --- | --- |
| V10.1 | 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 |
| V10.2 | Deprecated |  |  |  | 3.0 |
| V10.3 | Verify that TLS is used for all connections (including both external and backend connections) that are authenticated or that involve sensitive data or functions, and does not fall back to insecure or unencrypted protocols. Ensure the strongest alternative is the preferred algorithm. | ✓ | ✓ | ✓ | 3.0 |
| V10.4 | Verify that backend TLS connection failures are logged. |  |  | ✓ | 1.0 |
| V10.5 | Verify that certificate paths are built and verified for all client certificates using configured trust anchors and revocation information. |  |  | ✓ | 1.0 |
| V10.6 | Verify that all connections to external systems that involve sensitive information or functions are authenticated. |  | ✓ | ✓ | 1.0 |
| V10.7 | Deprecated |  |  |  | 3.0 |
| V10.8 | Verify that there is a single standard TLS implementation that is used by the application that is configured to operate in an approved mode of operation |  |  | ✓ | 1.0 |
| V10.9 | Deprecated |  |  |  | 3.0 |
| V10.10 | Verify that TLS certificate public key pinning is implemented with production and backup public keys. For more information, please see the references below. |  |  | ✓ | 3.0 |
| V10.11 | 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 |
| V10.12 | Verify that production website URL has been submitted to preloaded list of Strict Transport Security domains maintained by web browser vendors. Please see the references below. |  |  | ✓ | 3.0 |
| V10.13 | Ensure forward secrecy ciphers are in use to mitigate passive attackers recording traffic. | ✓ | ✓ | ✓ | 3.0 |
| V10.14 | Verify that proper certification revocation, such as OSCP Stapling, is enabled and configured. |  | ✓ | ✓ | 3.0 |
| V10.15 | Verify that only strong algorithms, ciphers, and protocols are used, through all the certificate hierarchy, including root and intermediary certificates of your selected certifying authority. | ✓ | ✓ | ✓ | 3.0 |
| V10.16 | Verify that the TLS settings are in line with current leading practice, particularly as common configurations, ciphers, and algorithms become insecure. |  | ✓ | ✓ | 3.0 |

## References

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

# V11: HTTP security configuration verification requirements

## Control objective

TBA

## Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Description** | **1** | **2** | **3** | **Since** |
| V11.1 | Deprecated |  |  |  | 2.0 |
| V11.2 | Verify that the application accepts only a defined set of HTTP request methods, such as GET and POST are accepted, and unused methods (e.g. TRACE, PUT, DELETE) are explicitly blocked. | ✓ | ✓ | ✓ | 1.0 |
| V11.3 | Verify that every HTTP response contains a content type header specifying a safe character set (e.g., UTF-8, ISO 8859-1). | ✓ | ✓ | ✓ | 1.0 |
| V11.4 | Deprecated |  |  |  | 2.0 |
| V11.5 | Deprecated |  |  |  | 2.0 |
| V11.6 | Deprecated |  |  |  | 3.0 |
| V11.7 | Deprecated |  |  |  | 2.0 |
| V11.8 | Deprecated |  |  |  | 3.0 |
| V11.9 | Verify that HTTP headers added by a trusted proxy or SSO devices, such as a bearer token, are authenticated by the application. |  | ✓ | ✓ | 2.0 |
| V11.10 | Verify that the HTTP header, X-Frame-Options is in use for sites where content should not be viewed in a 3rd-party X-Frame. A common middle ground is to send SAMEORIGIN, meaning only websites of the same origin may frame it. |  | ✓ | ✓ | 2.0 |
| V11.12 | Verify that the HTTP headers or any part of the HTTP response do not expose detailed version information of system components. |  | ✓ | ✓ | 2.0 |
| V11.13 | Verify that all API responses contain X-Content-Type-Options: nosniff and Content-Disposition: attachment; filename="api.json" (or other appropriate file name for the content type). | ✓ | ✓ | ✓ | 3.0 |

## References

* 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>

# V12: Security configuration verification requirements

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

# V13: Malicious controls verification requirements

## Control objective

TBA

## Requirements

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
| V13.1 | Deprecated |  |  |  | 3.0 |
| V13.2 | Deprecated |  |  |  | 3.0 |
| V13.3 | Deprecated |  |  |  | 3.0 |
| V13.4 | Deprecated |  |  |  | 3.0 |
| V13.5 | Deprecated |  |  |  | 3.0 |
| V13.6 | Deprecated |  |  |  | 3.0 |
| V13.7 | Deprecated |  |  |  | 3.0 |
| V13.8 | Deprecated |  |  |  | 3.0 |
| V13.9 | Deprecated |  |  |  | 3.0 |
| V13.10 | Verify all malicious activity is adequately sandboxed, containerized or isolated to delay and deter attackers from attacking other applications. |  |  | ✓ | 2.0 |
| V13.11 | Deprecated |  |  |  | 3.0 |
| V13.12 | Verify that a code review looks for malicious code, back doors, Easter eggs, and logic flaws. |  |  | ✓ | 3.0 |

## References

TBA

# V14: Internal security verification requirements

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

# V15: Business logic verification requirements

## Control objective

Verify the application has business limits and enforces them in a trusted location (as on a protected server) on a per user, per day or daily basis, with configurable alerting and automated reactions to automated or unusual attack. Examples include (but not limited to): ensuring new SIM users don’t exceed $10 per day for a new phone account, a forum allowing more than 100 new users per day or preventing posts or private messages until the account has been verified, a health system should not allow a single doctor to access more patient records than they can reasonably treat in a day, or a small business finance system allowing more than 20 invoice payments or $1000 per day across all users. In all cases, the business limits and totals should be reasonable for the business concerned. The only unreasonable outcome is if there are no business limits, alerting or enforcement.

## Requirements

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
| V15.1 | Deprecated |  |  |  | 3.0 |
| V15.2 | Deprecated |  |  |  | 3.0 |
| V15.3 | Deprecated |  |  |  | 3.0 |
| V15.4 | Deprecated |  |  |  | 3.0 |
| V15.5 | Deprecated |  |  |  | 3.0 |
| V15.6 | Deprecated |  |  |  | 3.0 |
| V15.7 | Deprecated |  |  |  | 3.0 |
| V15.8 | 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 |
| V15.9 | Deprecated |  |  |  | 3.0 |
| V15.10 | 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 |
| V15.11 | Verify that the application covers off risks associated with Spoofing, Tampering, Repudiation, Information Disclosure, and Elevation of privilege (STRIDE). |  | ✓ | ✓ | 3.0 |

## References

TBA

# V16: Files and resources verification requirements

## Control objective

TBA

## Requirements

| **#** | **Description** | **1** | **2** | **3** | **Since** |
| --- | --- | --- | --- | --- | --- |
| V16.1 | Verify that URL redirects and forwards only allow whitelisted destinations. | ✓ | ✓ | ✓ | 2.0 |
| V16.2 | Verify that untrusted file data obtained from the user is either not used directly with file I/O commands, particularly to protect against path traversal, local file include, file mime type, and OS command injection vulnerabilities. | ✓ | ✓ | ✓ | 2.0 |
| V16.3 | Deprecated |  |  |  | 3.0 |
| V16.4 | Deprecated |  |  |  | 3.0 |
| V16.5 | Verify that untrusted data is not used within inclusion, class loader, or reflection capabilities to prevent remote file inclusion vulnerabilities. | ✓ | ✓ | ✓ | 2.0 |
| V16.6 | Verify that untrusted data is not used within cross-domain resource sharing (CORS) to protect against arbitrary remote content. | ✓ | ✓ | ✓ | 2.0 |
| V16.7 | Verify that files obtained from untrusted sources are stored outside the webroot, with limited permissions, preferably with strong validation. |  | ✓ | ✓ | 2.0 |
| V16.8 | Verify that web or application server is configured by default to deny access to remote resources or systems outside the web or application server. |  | ✓ | ✓ | 2.0 |
| V16.9 | Verify the application code does not execute uploaded data obtained from untrusted sources. |  | ✓ | ✓ | 2.0 |
| V16.10 | Do not use Flash, Active-X, Silverlight, NACL, client-side Java or other client side technologies not supported natively via W3C browser standards. | ✓ | ✓ | ✓ | 2.0 |

## References

TBA

# V17: Mobile verification requirements

## Control objective

Provide unqiue security requirements for mobile applications, aligned with the OWASP Mobile Top 10.

Note: All of the deprecated requirements in version 3.0 are due to mobile requirements being duplicates of requirements found elsewhere within this standard. These have been removed as duplicates. A full mobile application assessment at each level requires consideration of all other applicable requirements throughout this standard.

## Requirements

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
| V17.1 | Deprecated |  |  |  | 3.0 |
| V17.2 | Verify that ID values stored on the device and retrievable by other applications, such as the UDID or IMEI number are not used as authentication tokens | ✓ | ✓ | ✓ | 2.0 |
| V17.3 | Verify that the mobile app does not store sensitive data onto potentially unencrypted shared resources on the device (e.g. SD card or shared folders) | ✓ | ✓ | ✓ | 2.0 |
| V17.4 | Verify that sensitive data is not stored unprotected on the device, even in system protected areas such as key chains | ✓ | ✓ | ✓ | 2.0 |
| V17.5 | Verify that secret keys, API tokens, or passwords are not hard-coded in mobile applications. |  | ✓ | ✓ | 2.0 |
| V17.6 | Verify that the mobile app prevents leaking of sensitive data where screenshots are saved of the current application, or as the application is backgrounded or the device is restarted. |  | ✓ | ✓ | 2.0 |
| V17.7 | Deprecated |  |  |  | 3.0 |
| V17.8 | Deprecated |  |  |  | 3.0 |
| V17.9 | Verify the permissions being requested as well as the resources that it is authorized to access are a minimal set of permissions required for the application to function. |  | ✓ | ✓ | 2.0 |
| V17.10 | Deprecated |  |  |  | 3.0 |
| V17.11 | Deprecated |  |  |  | 3.0 |
| V17.12 | Deprecated |  |  |  | 3.0 |
| V17.13 | Deprecated |  |  |  | 3.0 |
| V17.14 | Deprecated |  |  |  | 3.0 |
| V17.15 | Deprecated |  |  |  | 3.0 |
| V17.16 | Deprecated |  |  |  | 3.0 |
| V17.17 | Deprecated |  |  |  | 3.0 |
| V17.18 | Deprecated |  |  |  | 3.0 |
| V17.19 | Deprecated |  |  |  | 3.0 |
| V17.20 | Deprecated |  |  |  | 3.0 |
| V17.21 | Verify that the application makes use of Address Space Layout Randomization (ASLR). | ✓ | ✓ | ✓ | 2.0 |
| V17.22 | Deprecated |  |  |  | 3.0 |
| V17.23 | Deprecated |  |  |  | 3.0 |
| V17.24 | Deprecated |  |  |  | 3.0 |
| V17.25 | Verify that anti-debugging techniques sufficient to deter or delay likely attackers for the risk of the application are implemented within the app. |  |  | ✓ | 2.0 |
| V17.26 | Verify that the app does not export sensitive activities, intents, content providers etc. on Android. | ✓ | ✓ | ✓ | 2.0 |
| V17.27 | Verify that mutable structures have been used for sensitive strings such as account numbers and are overwritten when not used. (Mitigate damage from memory analysis attacks). |  |  | ✓ | 2.0 |
| V17.28 | Verify that any exposed intents, content providers, services, and broadcast receivers perform full data validation on input (Android). | ✓ | ✓ | ✓ | 2.0 |

## References

TBA

# V18: Web services verification requirements

## Control objective

TBA

## Requirements

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
| v18.1 | Verify that the same encoding style is used between the client and the server | ✓ | ✓ | ✓ | 3.0 |
| v18.2 | Verify that access to administration and management functions within the Web Service Application is limited to web service administrators | ✓ | ✓ | ✓ | 3.0 |
| v18.3 | Verify that XML or JSON schema are in place and verified before accepting input. | ✓ | ✓ | ✓ | 3.0 |
| v18.4 | Verify that all input is limited to an appropriate size limit | ✓ | ✓ | ✓ | 3.0 |
| v18.5 | Verify that the XML and JSON parsing processes are robust to XML and JSON based attacks | ✓ | ✓ | ✓ | 3.0 |
| v18.6 | Verify that Web services is compliant with Web Services-Interoperability (WS-I) Basic Profile at minimum |  | ✓ | ✓ | 3.0 |
| v18.7 | Verify the use of session-based authentication and authorization. Please refer to sections 2, 3 and 4 for further guidance. Avoid the use of static "API keys" and similar. | ✓ | ✓ | ✓ | 3.0 |
| v18.8 | Verify that the REST service is protected from Cross-Site Request Forgery. | ✓ | ✓ | ✓ | 3.0 |
| v18.9 | Verify the REST service explicitly check the incoming Content-Type to be the expected one, such as application/xml or application/json. |  | ✓ | ✓ | 3.0 |

## References

TBA

# V19. Configuration

## Control objective

TBA

## Requirements

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
| v19.1 | All components should be up to date with proper security configuration(s) and version(s). This should include unneeded configurations and folders (sample applications). | ✓ | ✓ | ✓ | 3.0 |
| v19.2 | Communications between components, such as between the application server and the database server, should be encrypted. |  | ✓ | ✓ | 3.0 |
| v19.3 | Communications between components, such as between the application server and the database server should be authenticated using an account with the least necessary privileges. |  | ✓ | ✓ | 3.0 |
| v19.4 | Verify application deployments are adequately sandboxed, containerized or isolated to delay and deter attackers from attacking other applications. |  | ✓ | ✓ | 3.0 |
| v19.5 | Verify that the application is securely built, installed, deployed or started. |  | ✓ | ✓ | 3.0 |
| v19.6 | Verify that authorised administrators have the capability to verify the integrity of all security-relevant configurations to ensure that they have not been tampered with. |  |  | ✓ | 3.0 |
| v19.7 | Verify that all application components are signed |  |  | ✓ | 3.0 |
| v19.8 | Verify that third party components come from trusted repositories |  |  | ✓ | 3.0 |

## References

TBA

# V20. Client side verification requirements

## Control objective

TBA

## Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Description** | **1** | **2** | **3** | **Since** |
| V20.1 | Ensure that Content Security Policy headers are delivered in a way that either disables inline JavaScript or provides an integrity check on inline JavaScript with CSP noncing or hashing. |  | ✓ | ✓ | 3.03.0 |
| V20.2 | Ensure that all string variables placed into HTML or other web client code is either properly contextually encoded manually, or utilized templates that auto-encode contextually. | ✓ | ✓ | ✓ | 3.0 |
| V20.3 | Make sure untrusted HTML from WYSIWYG editors or similar are properly sanitized with an HTML sanitizer. | ✓ | ✓ | ✓ | 3.0 |
| V20.4 | Ensure that if UI escaping is disabled, ensure that HTML sanitization is enabled instead. |  | ✓ | ✓ | 3.0 |
| V20.5 | Verify that data transferred from one DOM context to another uses safe JavaScript methods, such as using .innerText and .val |  | ✓ | ✓ | 3.0 |
| V20.6 | Verify that the X-XSS-Protection: 1; mode=block header is in place. |  | ✓ | ✓ | 3.0 |
| V20.7 | Verify that there is no sensitive business logic, secret keys or other proprietary information in client side code. |  | ✓ | ✓ | 3.0 |
| V20.8 | Verify when parsing JSON in browsers, that JSON.parse is used to parse JSON on the client. Do not use eval() to parse JSON on the client. |  | ✓ | ✓ | 3.0 |

## References

TBA

# Appendix A: Applying ASVS in Practice

Different threats have different motivations, and some industries have unique information and technology assets as well as 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 vulnerable applications reachable through the Internet, which is why ASVS Level 1 is recommended for all Internet-accessible applications regardless of industry. This is a suggested starting point, considering a small number of risk factors. 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.

| INDUSTRY SEGMENT | THREAT PROFILE | SUGGESTED ASVS LEVEL |
| --- | --- | --- |
| Finance and insurance | Although this segment will experience attempts from opportunistic attackers, it is often viewed as a high value target by motivated attackers and attacks are often financially motivated. Commonly, attackers are looking for sensitive data or account credentials that can be used to commit fraud or to benefit directly by leveraging money movement functionality built into applications. Techniques often include stolen credentials, application-level attacks, and social engineering. | Level 1: all Internet-accessible applications. |
|  | Some major compliance considerations include Payment Card Industry Data Security Standard (PCI DSS), Gramm-Leech Bliley act, Sarbanes Oxley (SOX). | Level 2: applications that contain sensitive information like credit card numbers, personal information, can move limited amounts of money in limited ways. Examples include: (i) transfer money between accounts at the same institution or (ii) a slower form of money movement (e.g. ACH) with transaction limits or (iii) wire transfers with hard transfer limits within a period of time. |
|  |  | Level 3: applications that contain large amounts of sensitive information or that allow either rapid transfer of large sums of money (e.g. wire transfers) or transfer of large sums of money in the form of individual transactions or as a batch of smaller transfers. |
| Manufacturing, professional, transportation, technology, utilities, infrastructure, and defense | These industries may not appear to have very much in common, but the threat actors who are likely to attack organizations in this segment are more likely to perform focused attacks with more time, skill, and resources. Often the sensitive information or systems are not easy to locate and require leveraging insiders and social engineering techniques. Attacks may involve insiders, outsiders, or be collusion between the two. Their goals may include gaining access to intellectual property for strategic or technological advantage. We also do not want to overlook attackers looking to abuse application functionality influence the behaviour of or disrupt sensitive systems. | Level 1: all Internet-accessible applications. |
|  | Most attackers are looking for sensitive data that can be used to directly or indirectly profit from to include personally identifiable information and payment data. Often the data can be used for identity theft, fraudulent payments, or a variety of fraud schemes. | Level 2: applications containing internal information or information about employees that may be leveraged in social engineering. Applications containing non-essential, but important intellectual property or trade secrets. |
|  |  | Level 3: applications containing valuable intellectual property, trade secrets, or government secrets (e.g. in the United States this may be anything classified at Secret or above) that is critical to the survival or success of the organization. Applications controlling sensitive functionality (e.g. transit, manufacturing equipment, control systems) or that have the possibility of threatening safety of life. |
| Healthcare | Most attackers are looking for sensitive data that can be used to directly or indirectly profit from to include personally identifiable information and payment data. Often the data can be used for identity theft, fraudulent payments, or a variety of fraud schemes. | Level 1: all Internet-accessible applications. |
|  |  | Level 2: applications with small or moderate amounts of sensitive medical information (Protected Health Information), Personally Identifiable Information, or payment data. |
|  |  | Level 3: Applications used to control medical equipment, devices, or records that may endanger human life. Payment and Point of Sale systems (POS) that contain large amounts of transaction data that could be used to commit fraud. This includes any administrative interfaces for these applications. |
| Retail, food, hospitality | Many of the attackers in this segment utilize opportunistic "smash and grab" tactics. However, there is also a regular threat of specific attacks on applications known to contain payment information, perform financial transactions, or store personally identifiable information. Although less likely than the threats mentioned above, there is also the possibility of more advanced threats attacking this industry segment to steal intellectual property, gain competitive intelligence, or gain an advantage with the target organization or a business partner in negotiations. | Level 1: all Internet-accessible applications. |
|  |  | Level 2: Suitable for business applications, product catalogue information, internal corporate information, and applications with limited user information (e.g. contact information). Applications with small or moderate amounts of payment data or checkout functionality. |
|  |  | Level 3: Payment and Point of Sale systems (POS) that contain large amounts of transaction data that could be used to commit fraud. This includes any administrative interfaces for these applications. Applications with a large volume of sensitive information like full credit card numbers, mother's maiden name, social security numbers etc. |

# Appendix B: Glossary

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

# Appendix C: Where to go from here

The OWASP ASVS is a living document. If you are performing an application security verification according to this standard, then you should always review the articles that can be found on the OWASP ASVS project page.

OWASP is the premier site for Web application security. The OWASP site hosts many projects, forums, blogs, presentations, tools, and papers. Additionally, OWASP hosts two major Web application security conferences per year, and has over 80 local chapters. The OWASP ASVS project page can be found here http://www.owasp.org/index.php/ASVS

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

* OWASP Code Review Guide - <http://www.owasp.org/index.php/Category:OWASP_Code_Review_Project>
* OWASP Enterprise Security API (ESAPI) Project - <http://www.owasp.org/index.php/ESAPI>
* OWASP Mobile Top 10   
  <https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks>
* OWASP Testing Guide - <https://www.owasp.org/index.php/OWASP_Testing_Project>
* OWASP Top Ten Project - <http://www.owasp.org/index.php/Top_10>

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

* OWASP - <http://www.owasp.org>   
  Contains thousands of pages of application security related information
* MITRE Common Weakness Enumeration - <http://cwe.mitre.org/>   
  This site provides a thorough mapping of all known weaknesses present in software
* PCI Security Standards Council - <https://www.pcisecuritystandards.org>   
  Publishers of PCI data security standards, relevant to all organizations processing or holding credit card data
* PCI Data Security Standard (DSS) v3.0 Requirements and Security Assessment Procedures <https://www.pcisecuritystandards.org/documents/PCI_DSS_v3.pdf>