

Application Security Verification Standard 3.0

October 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

## 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 © 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.

### Version 3.0, 2015

|  |  |  |
| --- | --- | --- |
| Project Leads | Lead Authors | Contributors and Reviewers |
| Andrew van der Stock  Daniel Cuthbert | Jim Manico | Boy Baukema  Ari Kesäniemi  Colin Watson  François-Eric Guyomarc’h  Cristinel Dumitru  James Holland  Gary Robinson  Stephen de Vries  Glenn Ten Cate  Riccardo Ten Cate  Martin Knobloch  Abhinav Sejpal  David Ryan  Steven van der Baan  Ryan Dewhurst  Raoul Endres  Roberto Martelloni |

### Version 2.0, 2014

| Project Leads | Lead Authors | Contributors and Reviewers |
| --- | --- | --- |
| Daniel Cuthbert  Sahba Kazerooni | Andrew van der Stock  Krishna Raja | 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 | Lead Authors | Contributors and Reviewers |
| Mike Boberski  Jeff Williams  Dave Wichers | Jim Manico | 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 |

# Preface

Welcome to the Application Security Verification Standard (ASVS) version 3.0. The ASVS is a community-effort to establish a framework of security requirements and controls that focus on normalising the functional and non-functional security controls required when designing, developing and testing modern web applications.

ASVS v3.0 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 3.0?

In version 3.0, we have added several new sections, including Configuration, Web Services, Modern (Client) based applications, to make the Standard more applicable to modern applications, which are commonly responsive applications, with an extensive HTML5 front end or mobile client that calls a common set of RESTful web services using SAML authentication.

We have also de-duplicated the standard, for example, to ensure that a mobile developer does not need to re-test the same items multiple times.

We have provided a mapping to the CWE common weakness enumeration (CWE) dictionary. The CWE mapping can be used to identify information such as likelihood of exploitation, consequence of a successful exploitation and broadly speaking to gain insight on what could go wrong if a security control is not used or implemented effectively and how to mitigate the weakness.

Lastly, we reached out to the community and held peer review sessions at AppSec EU 2015 and a final working session at AppSec USA 2015 to include a massive amount of community feedback. During peer review, if edits to the meaning of a control changed substantially, we created a new control and deprecated the old one. We have deliberately chosen to not reuse any deprecated control requirements, as this could be a source of confusion. We have provided a comprehensive mapping of what has changed in Appendix A.

Taken together, v3.0 is the single largest change to the Standard in its history. We hope that you find the update to the standard useful, and use it in ways we can only imagine.

# Using the Application Security Verification Standard

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

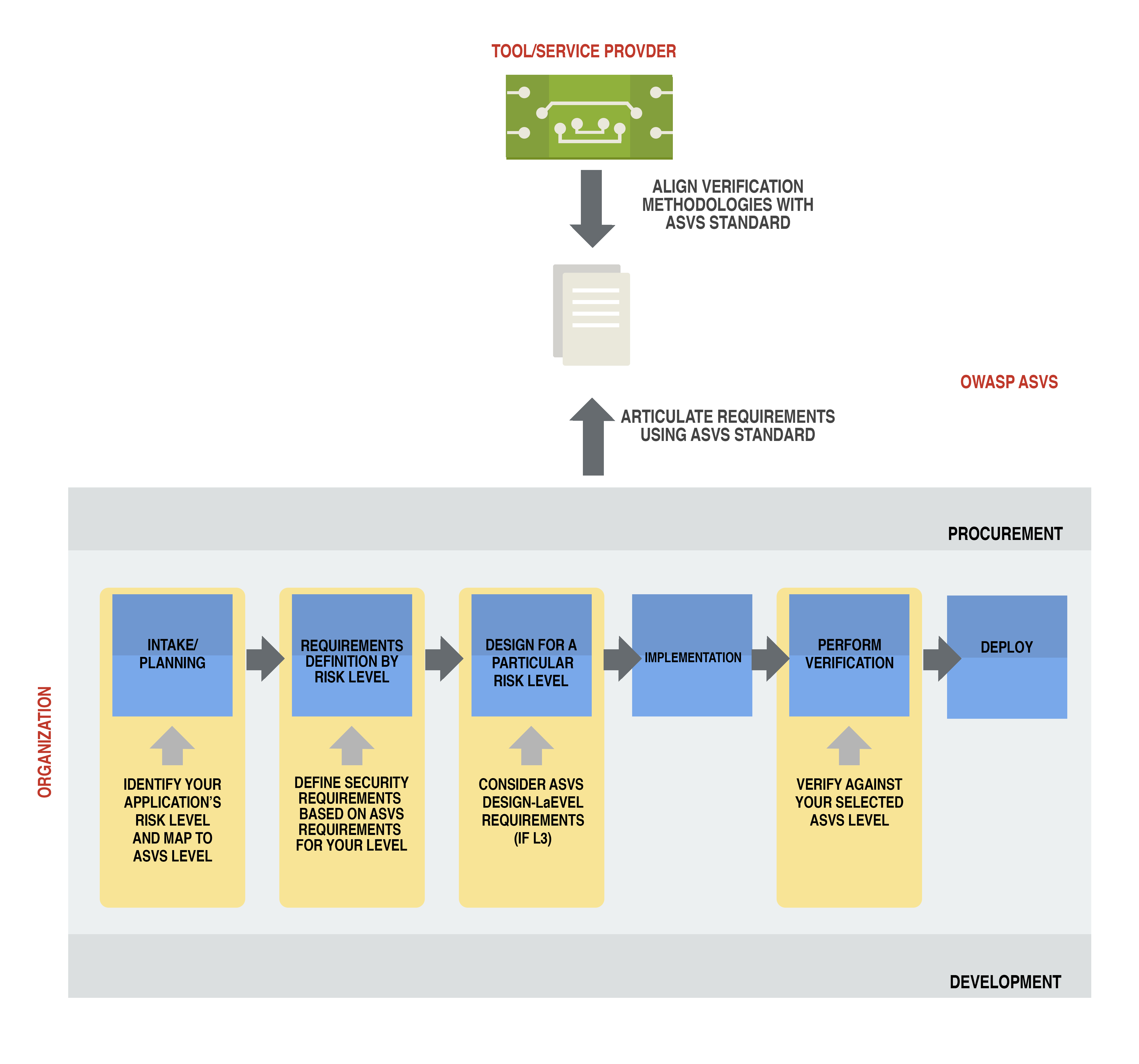


Figure 1 - Uses of ASVS for organizations and tool/service providers

## 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 2 - OWASP Application Security Verification Standard 3.0 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. Organisations 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.

| Industry | Threat Profile | L1 Recommendation | L2 Recommendation | L3 Recommendation |
| --- | --- | --- | --- | --- |
| 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. Some major compliance considerations include Payment Card Industry Data Security Standard (PCI DSS),Gramm Leech Bliley Act and Sarbanes-Oxley Act (SOX). | All network accessible applications. | Applications that contain sensitive information like credit card numbers, personal information, that 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. | Applications that contain large amounts of sensitive information or that allow either rapid transfer of large sums of money (e.g. wire transfers) and/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.  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. | All network accessible applications. | Applications containing internal information or information about employees that may be leveraged in social engineering. Applications containing nonessential, but important intellectual property or trade secrets. | 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 lif |
| 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.  For the US healthcare sector, the Health Insurance Portability and Accountability Act (HIPAA) Privacy, Security, Breach Notification Rules and Patient Safety Rule (<http://www.hhs.gov/ocr/privacy/>=[.](http://www.hhs.gov/ocr/privacy/) | All network accessible applications | Applications with small or moderate amounts of sensitive medical information (Protected Health Information), Personally Identifiable Information, or payment data. | 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. | All network accessible applications. | 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. | 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. |

# Case Studies

## Case Study 1: As a Security Testing Guide

At a private university in Utah, USA, the campus Red Team uses the OWASP ASVS as a guide when performing application penetration tests. It is used throughout the penetration testing process, from initial planning and scoping meetings to guidance for testing activities, and as a way to frame the findings of the final report to clients. The Red Team also organizes training for the team using the ASVS.

The campus Red Team performs network and application penetration testing for various departments on campus as part of the overall university's information security strategy. During initial planning meetings, clients are often reticent to give permission for their application to be tested by a team of students. By introducing the ASVS and explaining to stakeholders that testing activities will be guided by this standard, and that the final report will include how the application performed against the standard, many concerns are immediately resolved. The ASVS is then used during scoping to help determine how much time and effort will be spent on the test. By using the predefined verification levels of the ASVS, the Red Team explains risk-based testing. This helps the client, stakeholders, and the team to come to an agreement on an appropriate scope for the application in question.

Once testing begins, the Red Team uses the ASVS to organize activities and divide up the workload. By tracking which verification requirements have been tested and which are still pending, project managers for the team can easily see how the test is progressing. This leads to improved communication with clients and gives project managers the ability to better manage resources. Because the Red Team is composed primarily of students, most team members have multiple demands on their time from different courses. Well-defined tasks, based on individual verification requirements or entire categories, help team members know exactly what needs to be tested and allow them to provide accurate estimations on how long a task will take to complete. Reporting also benefits from the clear organization of the ASVS, as team members can write up a finding before moving on to the next task, effectively performing the majority of report writing concurrently with the penetration test.

The Red Team organizes the final report around the ASVS, reporting the status of each verification requirement and providing further details where appropriate. This gives clients and stakeholders a good idea of where their application stands as measured by the standard, and is extremely valuable on follow-up engagements because it allows them to see how security has improved or regressed over time. Furthermore, stakeholders interested in how the application performed a specific category or categories can easily find out that information because the report format aligns so closely with the ASVS. The clear organization of the ASVS has also made it easier to train new team members on how to write a report when compared to the previous report format.

Finally, training of the Red Team has improved after adopting the ASVS. Previously, weekly trainings were centered on a topic chosen by the team lead or project manager. These were selected based on requests by team members and perceived need. Training based on these criteria had the potential to broaden the skills of team members, but did not necessarily relate to core Red Team activities. In other words, the team did not get significantly better at penetration testing. After adopting the ASVS, team training now focuses on how to test individual verification requirements. This has led to a significant improvement in the measurable skills of individual team members and the quality of final reports.

## Case Study 2: As a secure SDLC

A startup looking to provide big data analytics to financial institutions realises that security in development is on top of the list of requirements in order to obtain access to and process financial metadata. In this instance, the startup has chosen to use the ASVS as the basis of their agile secure development lifecycle.

The startup uses the ASVS to generate epics and use cases for functional security issues, such as how best to implement login functionality. The startup uses ASVS in a different way than most - it looks through ASVS, picking the requirements that suit the current sprint, and adds them directly to the sprint backlog if it’s a functional requirement, or as a constraint to existing use cases if non-functional. For example, adding TOTP two factor authentication was selected, along with password policies and a web service regulator that doubles as a brute force detection and prevention mechanism. In future sprints, additional requirements will be selected based upon a “just in time”, “you ain’t gonna need it” basis.

The developers use the ASVS as a peer review checklist, which ensures unsafe code does not get checked in, and in retrospective plans to challenge developers who have checked in a new feature to ensure that they have considered likely ASVS requirements and if anything can be improved or reduced in future sprints.

Lastly, the developers use the ASVS as part of their automated verification secure unit and integration test suites to test for use, abuse, and fuzz testing cases. The aim is to reduce the risk from waterfall methodology “penetration testing at the end” causing expensive refactoring when delivering milestone builds into production. As new builds could be promoted after every sprint, it is not sufficient to rely upon a single assurance activity, and so by automating their testing regime, there should be no significant issues that can be found by even a skilled penetration tester with weeks to test the application.

# Assessing software has achieved a verification level

## OWASP’s stance on ASVS Certifications and Trust Marks

OWASP, as a vendor-neutral not-for-profit organisation, does not 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 penetration 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 modelling, 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.

## As 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.

# OWASP Projects using ASVS

## Security Knowledge Framework

<https://www.owasp.org/index.php/OWASP_Security_Knowledge_Framework>

Training developers in writing secure code - SKF is a fully open-source Python-Flask web-application that uses the OWASP Application Security Verification Standard to train you and your team in writing secure code, by design.

## OWASP Zed Attack Proxy

<https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project>

The OWASP Zed Attack Proxy (ZAP) is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications. It is designed to be used by people with a wide range of security experience and as such is ideal for developers and functional testers who are new to penetration testing. ZAP provides automated scanners as well as a set of tools that allow you to find security vulnerabilities manually.

## OWASP Cornucopia

<https://www.owasp.org/index.php/OWASP_Cornucopia>

OWASP Cornucopia is a mechanism in the form of a card game to assist software development teams identify security requirements in Agile, conventional and formal development processes. It is language, platform and technology agnostic. Cornucopia suits were selected based on the structure of the OWASP Secure Coding Practices - Quick Reference Guide (SCP), but with additional consideration of sections in the OWASP Application Security Verification Standard, the OWASP Testing Guide and David Rook’s Principles of Secure Development.

# Detailed Verification Requirements

V1. Architecture, design and threat modelling

V2. Authentication

V3. Session management

V4. Access control

V5. Malicious input handling

V7. Cryptography at rest

V8. Error handling and logging

V9. Data protection

V10. Communications

V11. HTTP security configuration

V13. Malicious controls

V15. Business logic

V16. File and resources

V17. Mobile

V18. Web services (NEW for 3.0)

V19. Configuration (NEW for 3.0)

# V1: Architecture, design and threat modelling

## Control objective

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

* At level 1, components of the application are identified and have a reason for being in the app
* At level 2, the architecture has been defined and the code adheres to the architecture
* At level 3, the architecture and design is in place, in use, and effective

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 |
| --- | --- | --- | --- | --- | --- |
| 1.1 | Verify that all application components are identified and are known to be needed. | ✓ | ✓ | ✓ | 1.0 |
| 1.2 | Verify that all components, such as libraries, modules, and external systems, that are not part of the application but that the application relies on to operate are identified. |  | ✓ | ✓ | 1.0 |
| 1.3 | Verify that a high-level architecture for the application has been defined. |  | ✓ | ✓ | 1.0 |
| 1.4 | Verify that all application components are defined in terms of the business functions and/or security functions they provide. |  |  | ✓ | 1.0 |
| 1.5 | 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 functions, and/or security functions, they provide. |  |  | ✓ | 1.0 |
| 1.6 | Verify that a threat model for the target application has been produced and covers off risks associated with Spoofing, Tampering, Repudiation, Information Disclosure, and Elevation of privilege (STRIDE). |  |  | ✓ | 1.0 |
| 1.7 | Verify all security controls (including libraries that call external security services) have a centralized implementation. |  | ✓ | ✓ | 3.0 |
| 1.8 | Verify that components are segregated from each other via a defined security control, such as network segmentation, firewall rules, or cloud based security groups. |  | ✓ | ✓ | 3.0 |
| 1.9 | Verify the application has a clear separation between the data layer, controller layer and the display layer, such that security decisions can be enforced on trusted systems. |  | ✓ | ✓ |  |
| 1.10 | Verify that there is no sensitive business logic, secret keys or other proprietary information in client side code. |  | ✓ | ✓ |  |

## References

For more information, please see:

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

# V2: Authentication Verification Requirements

## Control objective

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

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

## Requirements

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
| 2.1 | Verify all pages and resources by default require authentication except those specifically intended to be public (Principle of complete mediation). | ✓ | ✓ | ✓ | 1.0 |
| 2.2 | Verify that all password fields do not echo the user’s password when it is entered. | ✓ | ✓ | ✓ | 1.0 |
| 2.4 | Verify all authentication controls are enforced on the server side. | ✓ | ✓ | ✓ | 1.0 |
| 2.6 | Verify all authentication controls fail securely to ensure attackers cannot log in. | ✓ | ✓ | ✓ | 1.0 |
| 2.7 | Verify password entry fields allow, or encourage, the use of passphrases, and do not prevent long passphrases/highly complex passwords being entered. | ✓ | ✓ | ✓ | 3.0 |
| 2.8 | 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. | ✓ | ✓ | ✓ | 2.0 |
| 2.9 | Verify that the changing password functionality includes the old password, the new password, and a password confirmation. | ✓ | ✓ | ✓ | 1.0 |
| 2.12 | Verify that all suspicious authentication decisions are logged. This should include requests with relevant metadata needed for security investigations. |  | ✓ | ✓ | 2.0 |
| 2.13 | Verify that account passwords make use of a sufficient strength encryption routine and that it withstands brute force attack against the encryption routine. |  | ✓ | ✓ | 3.0 |
| 2.16 | Verify that credentials are transported using a suitable encrypted link and that all pages/functions that require a user to enter credentials are done so using an encrypted link. | ✓ | ✓ | ✓ | 3.0 |
| 2.17 | 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 |
| 2.18 | Verify that information enumeration is not possible via login, password reset, or forgot account functionality. | ✓ | ✓ | ✓ | 2.0 |
| 2.19 | 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 |
| 2.20 | 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. | ✓ | ✓ | ✓ | 3.0 |
| 2.21 | Verify that all authentication credentials for accessing services external to the application are encrypted and stored in a protected location. |  | ✓ | ✓ | 2.0 |
| 2.22 | Verify that forgotten password and other recovery paths use a soft token, mobile push, or an offline recovery mechanism. | ✓ | ✓ | ✓ | 3.0 |
| 2.23 | 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. |  | ✓ | ✓ | 3.0 |
| 2.24 | 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 |
| 2.25 | Verify that the system can be configured to disallow the use of a configurable number of previous passwords. |  | ✓ | ✓ | 2.0 |
| 2.26 | 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 |
| 2.27 | Verify that measures are in place to block the use of commonly chosen passwords and weak passphrases. | ✓ | ✓ | ✓ | 3.0 |
| 2.28 | Verify that all authentication challenges, whether successful or failed, should respond in the same average response time. |  |  | ✓ | 3.0 |
| 2.29 | Verify that secrets, API keys, and passwords are not included in the source code, or online source code repositories. |  |  | ✓ | 3.0 |
| 2.30 | Verify that if an application allows users to authenticate, they use a proven secure authentication mechanism. | ✓ | ✓ | ✓ | 3.0 |
| 2.31 | Verify that if an application allows users to authenticate, they can authenticate using two-factor authentication or other strong authentication, or any similar scheme that provides protection against username + password disclosure. |  | ✓ | ✓ | 3.0 |
| 2.32 | Verify that administrative interfaces are not accessible to untrusted parties | ✓ | ✓ | ✓ | 3.0 |

## References

For more information, please see:

* OWASP Testing Guide 4.0: Testing for Authentication <https://www.owasp.org/index.php/Testing_for_authentication>
* Password storage cheat sheet <https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet>
* Forgot password cheat sheet <https://www.owasp.org/index.php/Forgot_Password_Cheat_Sheet>
* Choosing and Using Security Questions at <https://www.owasp.org/index.php/Choosing_and_Using_Security_Questions_Cheat_Sheet>

# V3: Session Management Verification Requirements

## Control objective

TBA

## Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | **Description** | **1** | **2** | **3** | **Since** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## References

For more information, please see:

# V4: Access Control Verification Requirements

## Control objective

TBA

## Requirements

| **#** | **Description** | **1** | **2** | **3** | **Since** |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## References

For more information, please see:

# V5: Malicious input handling verification requirements

## Control objective

TBA

## Requirements

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## References

For more information, please see:

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** |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## References

For more information, please see:

# V8: Error handling and logging verification requirements

## Control objective

TBA

## Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | 1 | 2 | 3 | Since |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## References

For more information, please see:

# V9: Data protection verification requirements

## Control objective

TBA

## Requirements

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## References

For more information, please see:

TBA

# V10: Communications security verification requirements

## Control objective

TBA

## Requirements

| **#** | **Description** | **1** | **2** | **3** | **Since** |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## References

For more information, please see:

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

## References

For more information, please see:

* 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 |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## References

For more information, please see:

# 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

TBA

## Requirements

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## References

For more information, please see:

# V16: Files and resources verification requirements

## Control objective

TBA

## Requirements

| **#** | **Description** | **1** | **2** | **3** | **Since** |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## References

For more information, please see:

# V17: Mobile verification requirements

## Control objective

TBA

## Requirements

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## References

For more information, please see:

# V18: Web services verification requirements

## Control objective

TBA

## Requirements

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## References

For more information, please see:

# V19. Configuration

## Control objective

TBA

## Requirements

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## References

For more information, please see:

# Appendix A: What ever happened to…

# Appendix B: Glossary

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

# Appendix C: References

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

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

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

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

# Appendix D: Standards Mappings

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

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

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