OWASP Application Security Verification Standard 2.0

# Foreword

The OWASP Application Security Verification Standard 2.0 defines three levels of application level security verification:

* **Level 1 – Automated Verification** – for the most part, all applications should be meet this most basic of verification levels. Such verification should be possible using a combination of dynamic and static testing tools, or via manual means in a short period of time. This level is eminently suitable to rapidly determine which applications in a large fleet require Level 2 or Level 3 assessment or further detection and protection if no further development is possible. As this level is mostly automated, the use of a skilled individual to weed out false positives and investigate false negatives is recommended.
* **Level 2 – Design Verification** – suitable for most applications other than those systems with rigorous regulatory requirements such as health or financial applications. For the most part, moderately skilled individuals should be able to complete this verification level using freely available automated and manual tools within a reasonable period of time.
* **Level 3** – **Comprehensive review** – suitable for all applications, including those with rigorous regulatory and legislative requirements, such as health and financial applications. For example, Level 3 can be used to demonstrate compliance with industry regulations such as PCI DSS Section 6.5 as it’s far in excess of Section 6.5’s modest requirements. Skilled individuals should perform such reviews within a reasonable period of time, using specialist manual and automated tools to assist with assessment productivity.

The requirements were developed with the following objectives in mind:

* “*Zero tolerance of the unforgivable*” – there is no room in the modern IT world for SQL injection or cross-site scripting. The OWASP Application Security Verification Standard 2.0 requires all passing applications to be free of these two major issues, even at Level 1.
* *“Control in place, control is effective, and control is used” model*. There is no reason to adhere to industry standards if the required level achieves a checkbox, but the controls are unused or ineffective. ASVS 2.0 aims to ensure that organizations are achieving application security rather than simply assessing if they have security measures.
* *Use as a metric* – Provide application developers and application owners with a yardstick with which to assess the degree of trust that can be placed in their Web applications,
* *Use as guidance* – Provide guidance to security control developers as to what to build into security controls in order to satisfy application security requirements,[[1]](#footnote-1) and
* *Use during procurement* – Provide a basis for specifying application security verification requirements in contracts.[[2]](#footnote-2)

The requirements are easily testable through automated or manual means using a combination of access to source code, configuration, and basic testing techniques. In general, the ASVS security controls are:

* Should be known and understood by the development team
* In place – enabling security controls, such as single sign on, input validation and output encoding libraries should be available to all team members and applications.
* In use – an enabling security control is ineffective if it is present and unused. If an application has a requirement for a particular enabling feature, it should use a well-known, well-tested, secure component.
* Effective - designed and implemented securely, with positive use and negative abuse test cases
* Built with modern tool chains and analysis tools or up to date if third party components or libraries
* Project documentation is used by the team, accurate, and up to date, preferably built from the metadata of the application to minimize errors and staleness; and
* Deployed securely with a minimal attack surface area.

The requirements ensure that the security controls used by an application operate using a deny-by-default strategy, are centralized, are located in a trusted location, and are used where necessary.

# Copyright and License

Copyright © 2008 – 2013 The OWASP Foundation.

This document is released under the [Creative Commons Attribution ShareAlike 3.0](http://creativecommons.org/licenses/by-sa/3.0/) license. For any reuse or distribution, you must make clear to others the license terms of this work.

# Table of Contents

Foreword 2

Copyright and License 3

Table of Contents 5

Introduction 1

Approach 1

Acknowledgements 3

Application Security Verification Levels 4

Level 1 – Automated Verification 4

Dynamic Scan (Partial Automated Verification) 6

Source Code Scan (Partial Automated Verification) 6

Level 2 – Design Verification 1

Level 3 – Comprehensive Verification 3

Requirement Interpretations and Precedents 4

Mapping ASVS 1.0 to 2.0 5

Adopting the standard in your tool 6

Detailed Verification Requirements 1

SL – Secure development lifecycle Requirements 2

V1 – Architecture Verification Requirements 1

BL – Business Logic Verification Requirements 1

V2 – Authentication Verification Requirements 1

V3 – Session Management Verification Requirements 3

V4 – Access Control Verification Requirements 4

V5 – Input Validation Verification Requirements 1

V6 – Output Encoding/Escaping Verification Requirements 1

FR – Files and Remote resource Verification Requirements 1

V7 – Cryptography Verification Requirements 2

V8 – Error Handling and Logging Verification Requirements 1

V9 – Data Protection Verification Requirements 1

V10 – Communication Security Verification Requirements 1

V11 – HTTP Security Verification Requirements 1

V12 – Security Configuration Verification Requirements 1

V13 – Malicious Code Search Verification Requirements 1

V14 – Internal Security Verification Requirements 1

Verification Reporting Requirements 2

R1 – Report Introduction 2

R2 – Application Description 2

R3 – Application Security Architecture 2

R4 – Verification Results 3

Glossary 5

Where To Go From Here 8

# 

# Introduction

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](http://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 primary aim of the OWASP Application Security Verification Standard (ASVS) Project 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 using a commercially-workable open standard. The standard provides a basis for testing 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)](http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet) and [SQL injection](http://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet).[[3]](#footnote-3) This standard can be used to establish a level of confidence in the security of Web applications.

# Approach

The OWASP ASVS defines verification and documentation requirements that are grouped on the basis of related coverage and level of rigor. The Standard defines three hierarchical levels (e.g. Level 2 requires more coverage and rigor than Level 1) as depicted in the figure below.

Figure – OWASP ASVS Levels

Web application security verification is performed from a logical point of view by following (or attempting to follow) paths into and out of a targeted application (called the Target of Verification or TOV) and performing analysis along those paths. More complex applications typically take more time to analyze resulting in longer and more costly verifications. Lines of code are not the only factors that determine the complexity of an application – different technologies will typically require different amounts of analysis. Simple applications may include for example libraries and frameworks. Applications of moderate complexity may include simple Web 1.0 applications. Complex applications may include Web 2.0 applications and new/unique Web technologies.

ASVS defines constituent components for Levels 1 and 2 (e.g. verification at Level 1 requires meeting both Level 1A and 1B requirements). For example, applications may claim compliance to either Level 1A or 1B instead of Level 1, but making such claims is weaker than claiming Level 1. Verification and documentation requirements are defined in this Standard using three types of requirements: High-Level requirements, Detailed requirements, and Reporting requirements. The High-Level requirements define the overall application implementation and verification requirements. The Detailed requirements define low-level application implementation and verification requirements (i.e., specific items to verify). The Reporting requirements define how the results of performing an application verification according to the OWASP ASVS must be documented.

OWASP provides numerous resources, including ASVS, to help organization’s develop and maintain secure applications. The OWASP ASVS, OWASP Contract Annex,[[4]](#footnote-4) and OWASP ESAPI[[5]](#footnote-5) can be used to support your Software Development Life Cycle (SDLC) as depicted in the figure below.

Diagram removed

Figure – One way to introduce verification as an activity into your SDLC[[6]](#footnote-6)

# Acknowledgements

**Version 2.0**

Project Leads: Sahba Kazerooni, Daniel Cuthbert

Lead Author: Andrew van der Stock

Authors: TBA

Reviewers and contributors: TBA

**Version 1.0**

As ASVS 2.0 includes many of the original requirements, the following contributors are recognized for their efforts during the original Application Security Verification Standard effort.

Project Lead: Mike Boberski

Lead Authors: Mike Boberski, Jeff Williams, Dave Wichers

We thank the [OWASP Foundation](http://www.owasp.org) for sponsoring the [OWASP Application Security Verification Standard Project](http://www.owasp.org/index.php/ASVS) during the OWASP Summer of Code 2008.

Acknowledgement is given for the contributions of: Pierre Parrend (OWASP Summer of Code), Andrew van der Stock, Nam Nguyen, John Martin, Gaurang Shah, Theodore Winograd, Stan Wisseman, Barry Boyd, Steve Coyle, Paul Douthit, Ken Huang, Dave Hausladen, Mandeep Khera Scott Matsumoto, John Steven, Stephen de Vries, Dan Cornell, Shouvik Bardhan, Dr. Sarbari Gupta, Eoin Keary, Richard Campbell, Matt Presson, Jeff LoSapio, Liz Fong, George Lawless, Dave van Stein, Terrie Diaz, Ketan Dilipkumar Vyas, Bedirhan Urgun, Dr. Thomas Braun, Colin Watson, Jeremiah Grossman.

And finally, thanks are given to the application security verification community and others interested in trusted web computing for their enthusiastic advice and assistance throughout this effort.

# Application Security Verification Levels

The ASVS defines three levels of verification that increase in both breadth and depth as one moves up the levels. The breadth is defined in each level by a set of security requirements that must be addressed. The depth of the verification is defined by the approach and level of rigor required in verifying each security requirement.

Tools are an important part of every ASVS level. At higher levels in ASVS, the use of tools is encouraged. But to be effective, the tools must be heavily tailored and configured to the application and framework in use. And, at all levels, tool results must be manually verified.

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. This standard uses the term the ‘verifier’ to indicate the person or team that is reviewing the application against these requirements.

The specification for an application may require OWASP ASVS Level *N*, but it could also include other additional detailed requirements such as from a higher ASVS level. For example, a financial organization may have a lower-risk application verified to OWASP ASVS Level 2 but may also want verification that no malicious code (see V13, Level 3 only) has been included. Other organization or business requirements could apply, such as compliance with particular information security policies and regulations.

*There is no verification level 0. Also, to earn a level, vulnerabilities must be remediated (or mitigated), and the application re-verified.*

## Level 1 – Automated Verification

Level 1 (“Automated Verification”) 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. The goal of Level 1 is to be as automated as possible. It is likely that no one tool can satisfy the requirements of Level 1, and thus manual intervention will be required to complete a level 1 review.

The scope of verification includes all code developed or modified for the application as processed in an automated fashion by any combination of static secure source code analysis (SAST) and dynamic web application analysis (DAST), and manual means. The verifier must verify automated findings to remove false positives and to ensure that false negatives are not present.

The coverage of controls is approximately 20% of the comprehensive (Level 3) scan by design to provide a fast and likely mostly automated result in a short time frame. As such, this level cannot be considered a thorough inspection or verification of the application, but more of a quick inspection.

Threats to security[[7]](#footnote-7) will typically be viruses and worms (targets are chosen indiscriminately through wide scans and impact the most vulnerable).

The following are the minimal high-level requirements for Level 1 applications:

|  |  |  |
| --- | --- | --- |
| Ref | Description | Requirement |
| L1.1 | Verification Scope | The scope of the verification includes all code that was developed or modified in order to create the application. |
|  | Security control behavior requirements | There are no requirements for how application security controls make decisions at Level 1. |
|  | Security control use requirements | There are no requirements for where application security controls are used within the application at Level 1. |
|  | Security control implementation requirements | There are no requirements for how the application security controls are built at Level 1. |
| L1.2 | Security control verification requirements | Dynamically scan the application according to the Level 1 requirements in the “Detailed Verification Requirements” section. |
| L1.3 | Security control verification requirements | Perform source code scanning on the application according to the Level 1 requirements in the “Detailed Verification Requirements” section. |
| L1.4 | Reporting requirements | Create a verification report that details the application’s security architecture by listing its components, and includes the results of the verification according to the requirements in the “Verification Reporting Requirements” section. |

Requirements at Level 1 that allow the use of either verification technique only have to be verified with one technique. In addition, if the verifier’s selected tool suite does not have the capability to verify a specified verification requirement, the verifier can perform manual verification to fill this gap.[[8]](#footnote-8) [[9]](#footnote-9)

At Level 1, application components may be defined in terms of either individual or groups of source files, libraries, and/or executables, as depicted in the figure below. At Level 1, the list need not be sorted or otherwise organized other than identifying which components are part of the application, and which components are part of the IT environment. The application can then be treated as groups of components within a single monolithic entity. The path or paths a given end user request may take within the application do not need to be identified and documented.

Diagram removed

Figure – OWASP ASVS Level 1 Security Architecture Example

## Level 2 – Design Verification

Level 2 (“Design Verification”) 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.

Level 2 ensures that evaluated security controls are in place, effective, and used everywhere within the application they need to be used to enforce application-specific policies. The coverage of controls is approximately 80% of the comprehensive (Level 3) scan by design.

The scope of verification includes all code developed or modified for the application, as well as examining the security of all third party components that provide security functionality for the application.

Threats to security will typically be viruses and worms, opportunists, and possibly determined attackers (skilled and motivated attackers focusing on specific targets using tools including purpose-built scanning tools).

The following are the minimal high-level requirements for Level 2 applications:

|  |  |  |
| --- | --- | --- |
| Ref | Description | Requirement |
| L2.1 | Verification Scope | The scope of the verification includes all code that was developed or modified in order to create the application. |
| L2.2 | The scope of the verification includes the code for all third-party framework, library, and service security functionality that is invoked by or supports the security of the application. |
| L2.3 | The scope of the verification includes the code for all third-party frameworks, libraries, and services associated with the application. |
| L2.4 | Security control behavior requirements | Verify that all security controls that perform security checks make decisions using a whitelist approach. |
| L2.5 | Verify that all security controls that perform security checks and security controls that result in security effects cannot be bypassed according to the Level 3 requirements specified in the “Detailed Verification Requirements” section. |
| L2.6 | Security control use requirements | Verify that all security controls are used everywhere within the application they need to be, and the implementations are centralized within the application, on the server side, according to the Level 2 requirements specified in the “Detailed Verification Requirements” section. |
|  | Security control implementation requirements | There are no requirements for how application security controls are built at Level 2. |
| L2.7 | Security control verification requirements | Manually verify the application according to the Level 3 requirements specified in the “Detailed Verification Requirements” section. This augments the manual verification requirements introduced at Level 2. |
| L2.8 | Security control verification requirements | Document a security architecture and use it to verify the proper design and use of all security controls by performing threat modeling. This is a new requirement at Level 3. |
| L2.9 | Reporting requirements | Create a verification report that describes the application’s security architecture by grouping its components into a high-level architecture that includes threat modeling information, and includes the results of the verification according to the requirements in the “Verification Reporting Requirements” section. This augments the reporting requirement from Level 2. |

At level 2, application components may be defined in terms of either individual or groups of source files, libraries, and/or executables that are grouped into a high-level architecture (for example MVC components, business function components, and data layer components). At Level 3, supporting threat modeling information about threat agents and assets must additionally be provided. The path or paths a given end user request may take through a high-level view of the application must be documented, as depicted in the figure below. At Level 3, *all* potential paths through the high-level view of the application must be examined.

Diagram removed

Figure – OWASP ASVS Level 3 Security Architecture Example

## Level 3 – Comprehensive Verification

Level 3 (“Comprehensive Verification”) is typically appropriate for critical applications that protect life and safety, critical infrastructure, or defense functions. Level 3 may also be appropriate for applications that process sensitive assets.

Level 3 ensures that evaluated security controls are in place, effective, and used everywhere within the application they need to be used to enforce application-specific policies. In addition, internal development policies are verified to ensure that secure coding practices are adhered to.

The scope of verification includes all code developed or modified for the application, as well as examining the security of all third party components that provide security functionality for the application.

Threats to security will be from determined attackers (skilled and motivated attackers focusing on specific targets using tools including purpose-built scanning tools). The scope of verification expands beyond the scope of Level 2 to include all code used by the application.

| Ref | Description | Requirement |
| --- | --- | --- |
| L3.1 | Verification Scope | The scope of the verification includes all code that was developed or modified in order to create the application. |
| L3.2 | The scope of the verification includes the code for all third-party framework, library, and service security functionality that is invoked by or supports the security of the application. |
| L3.3 | The scope of the verification includes the code for all third-party frameworks, libraries, and services associated with the application. |
| L3.4 | The scope of the verification includes all remaining code associated with the application, including frameworks, libraries, runtime environments, development tools, build tools, and deployment tools. The scope does not include the code for platform software, such as an application server, database server, virtual machine, or operating system, that has received a substantial amount of public scrutiny. |
| L3.5 | Security control behavior requirements | Verify that all security controls that perform security checks make decisions using a whitelist (“positive”) approach |
| L3.6 | Verify that all security controls that perform security checks and security controls that result in security effects cannot be bypassed according to the Level 4 requirements specified in the “Detailed Verification Requirements” section |
| L3.7 | Security control use requirements | Verify that all security controls are used everywhere within the application they need to be, and that the implementations are centralized within the application, on the server side, according to the Level 4 requirements specified in the “Detailed Verification Requirements” section |
| L3.8 | Security control implementation requirements | Verify that the application does not contain any malicious code according to the Level 4 requirements specified in the “Detailed Verification Requirements” section. This is a new requirement at Level 4 |
| L3.9 | Security control verification requirements | Manually verify the application against the Level 4 requirements specified in the “Detailed Verification Requirements” section. |
| L3.10 | Document a security architecture and use it to verify the proper design and use of all security controls by performing threat modeling |
| L3.11 | Document a security architecture and use it to verify the proper design and use of all security controls by performing threat modeling |
| L3.12 | Manually review all code developed or modified for this application for malicious code[[10]](#footnote-10) according to the Level 4 requirements specified in the “Detailed Verification Requirements” section |
| L3.13 | Reporting requirements | Create a verification report that describes the application’s security architecture according to the Level 3 requirements, which encompasses all application code, and includes the results of the verification according to the requirements in the “Verification Reporting Requirements” section |

## Requirement Interpretations and Precedents

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

[http://www.owasp.org/index.php/ASVS#Articles\_Below\_-\_More\_About\_ASVS\_and\_Using\_It](http://www.owasp.org/index.php/ASVS" \l "Articles_Below_-_More_About_ASVS_and_Using_It) . The articles on the OWASP ASVS project page provide requirement clarifications, requirement verdict precedents, and helpful hints.

# Mapping ASVS 1.0 to 2.0

In general, ASVS 1.0 requirements numbering is left as is so as to not invalidate any previous work against ASVS 1.0 and allow comparison of ASVS 1.0 and ASVS 2.0 reviews.

Where a control has been moved, the previous number of that control is not re-used and a note detailing where the control has moved included.

New controls, revision or explanation of meaning, or changes to level are highlighted in yellow to allow tool vendors and reviewers to identify at a glance 2.0 additions and changes.

Some of the macro changes include:

* Collapsing of Levels 1, 1A and 1B to *Level 1 Automated Verification*, Levels 2, 2A, 2B and 3 to be *Level 2 Design Verification*, and renumbering Level 4 to *Level 3 Comprehensive Verification*. This is to simplify the application security verification standard, whilst making it obvious as to what is being achieved.
* In general, Level 1 has been trimmed to just those controls that are likely to be scannable using contemporary 2013 automated tooling (DAST or SAST or any combination thereof). This does not remove the requirement to validate automated findings manually, as no tool can detect or correctly interpret all of the requirements of this standard
* UPDATED - Malicious code controls from each section have been migrated to the malicious controls section, which is now an optional level 3 only section.
* UPDATED – Controls from the Australian Defense Signals Directorate Top 35 mitigations that are easily performed by an average development team have been incorporated into the internal security section.
* UPDATED - Previously vague requirements are tightened up to be specific and measurable. These are marked “(Clarified Requirement)” and highlighted in yellow.
* UPDATED - architecture requirements, but this is now only a Level 2 and 3 control section
* NEW - Business Logic Verification, primarily a Level 2 and 3 control section
* NEW- File Handling Verification, to eliminate local and remote file inclusion attacks, and virus uploads, which are the leading causes of “drive by” browser attacks
* In ASVS 2.0, there is a requirement **all** applications using credentials will encrypt communications. This is primarily due to the low cost of standard TLS and extended validation (EV) certificates compared to the cost of undergoing Application Security Verification Standard verification even at Level 1.

## Adopting the standard in your tool

We strongly encourage tool vendors to adopt the Application Security Verification Framework:

* Governance, risk and compliance tools are encouraged to adopt and re-use the text of the Standard, with appropriate attribution as per the CC:SA 3.0 license;
* Engagement management tools can assist productivity of verifiers by including work flow features, such as scope, approach and work plan (checklist) creation and maintenance;
* Document management solutions can assist with the collection, searching and secure preservation of documents, interviews, source code and repository collection, and wiki and CMS mirroring;
* SAST and DAST tool vendors are encouraged to create ASVS 2.0 profiles for their scan engines, but cannot claim certified fully automated scans of any ASVS level as manual verification of findings is mandatory
* Reporting tools are encourage to be aligned with the reporting guidelines detailed in Appendix R1.

Tools should assist the verification process in gathering information, assessing the application, finding notorious security defects, hot spots (areas of code that will reward manual inspection), and deserts (areas of code that should contain security controls, but don’t), assisting hybrid reviews, and producing reports.

The Application Security Verification Standard 2.0 requires manual verification of all findings, regardless of if the findings are produced manually or via automated means. Therefore, no fully automated tool can claim to produce certified scans against the OWASP Application Security Verification Standard 2.0 unless a human has certified the findings.

If you are a tool vendor and wish to incorporate the Application Security Verification Standard into your tool, please contact the Project team so we can work with you to produce the best outcomes for your clients. OWASP™ is a registered trademark of the OWASP Foundation and must not be misused, including false claims of automated certification of ASVS Levels.

# Detailed Verification Requirements

This section of the OWASP Application Security Verification Standard (ASVS) defines detailed verification requirements that were derived from the high-level requirements for each of the verification levels defined in this standard. Each section below defines a set of detailed verification requirements grouped into related areas.

The ASVS defines the following security requirements areas:

* SL. Secure development lifecycle
* V1. Security Architecture
* BL. Business Logic
* V2. Authentication
* V3. Session Management
* V4. Access Control
* V5. Input Validation
* V6. Output Encoding/Escaping
* FR. Files and Remote Resource
* V7. Cryptography
* V8. Error Handling and Logging
* V9. Data Protection
* V10. Communication Security
* V11. HTTP Security
* V12. Security Configuration
* V13. Malicious Code Search
* V14. Internal Security

V2.0 substantial change

## SL – Secure development lifecycle Requirements

The Secure Software Development Lifecycle Verification Requirements define a set of requirements for setting out the business requirement for secure software, and enabling processes, policies and technologies to produce secure software. Although it is possible to produce small programs that are provably secure using ad hoc methods, such as no source control or no unit tests, this is the exception and not the rule.

The Secure Software Development Lifecycle Verification Requirements are based on the OWASP Software Assurance Maturity Model (SAMM). SAMM is an open framework to help organizations formulate and implement a strategy for software security that is tailored to the specific risks facing the organization. The resources provided by SAMM will aid in:

* Evaluating an organization’s existing software security practices
* Building a balanced software security assurance program in well-defined iterations
* Demonstrating concrete improvements to a security assurance program
* Defining and measuring security-related activities throughout an organization

No requirement in this table should cost an organization anything other than initial planning effort – all the mandated tools are freely available as open source to run locally on a single workstation through to massive projects, with thousands of contributors and millions of lines of code. The initial planning effort is handsomely paid off within the first few compile cycles through reduced re-work and higher quality.

This is an organizational control, and should apply to all applications acquired or built by an organization. The results should be re-tested and re-used between assessments.

The Secure Software Development Lifecycle Verification Requirements concentrate on the minimal SAMM security practices that are part of the SAMM Governance, Construction and Deployment business functions. The SAMM Verification security practices - Design Review, Code Review and Security Testing – are covered by the ASVS verification activities.

| Ref | OWASP ASVS Secure SDLC Requirements (SL) | 1 | 2 | 3 |
| --- | --- | --- | --- | --- |
| SL.1 | Verify that the organization has established and is maintaining a software security roadmap within the organization, based on its overall business risk profile. This includes allocated security roles and responsibilities. (SAMM-SM1) | ✓ | ✓ | ✓ |
| SL.2 | Verify that the organization has an ongoing process in place for application and data asset risk categorization with aligned SAMM practice objectives per risk category. (SAMM-SM2) |  | ✓ | ✓ |
| SL.3 | Verify that the organization has identified and monitors external compliance drivers with corresponding control statements, which are updated and reviewed regularly with stakeholders. (SAMM-PC1) | ✓ | ✓ | ✓ |
| SL.4 | Verify that organization management and development teams have committed themselves to producing secure software by creating and enforcing a secure development policy, and subsequently adopting secure coding standards, training, and checklists. (SAMM-PC2) | ✓ | ✓ | ✓ |
| SL.5 | Verify that the organization has defined and implemented compliance gates for its application releases. (SAMM-PC3) |  | ✓ | ✓ |
| SL.6 | Verify that the application developers have been given application security awareness training and have access to secure development best practices and guidance. (SAMM-EG1) | ✓ | ✓ | ✓ |
| SL.7 | Verify that all personnel in the software lifecycle have been given role-specific guidance and can rely on support from security coaches for their activities. (SAMM-EG2) |  | ✓ | ✓ |
| SL.8 | Verify that development teams have created and maintain application threat models with a clear selection of measures to mitigate the important threats. (SAMM-TA1) | ✓ | ✓ | ✓ |
| SL.9 | Verify that development teams have specified security requirements derived from business functionality and organization security and compliance policies. (SAMM-SR1) | ✓ | ✓ | ✓ |
| SL.9 | Verify that development teams apply a short list of guiding security principles as a checklist during design. Typically, security principles include defense in depth, securing the weakest link, use of secure defaults, simplicity in design of security functionality, secure failure, balance of security and usability, running with least privilege, avoidance of security by obscurity, etc. (SAMM-SA1) |  | ✓ | ✓ |
| SL.10 | Verify that the organization can attest that their software is secure, via the production of independent secure architecture reviews, secure design reviews, secure code reviews, secure configuration reviews, security testing and operational validation assessments (penetration testing). (SAMM – Verification) | ✓ | ✓ | ✓ |
| SL.11 | Verify that the organization has a plan in place for plan for responding to reported vulnerabilities or incidents. (SAMM-VM1) | ✓ | ✓ | ✓ |
| SL.12 | Verify that the organization identifies and installs critical security upgrades and patches for the complete stack of software running its applications, including built-in programming language libraries, third-party components and development frameworks, base operating systems, etc. (SAMM-EH1) | ✓ | ✓ | ✓ |
| SL.13 | Verify that development teams gather security critical information and communicate this to the users and operators of the software. (SAMM-OE1) | ✓ | ✓ | ✓ |

## V1 – Architecture Verification Requirements

The Architecture Verification Requirements define a set of requirements for application security architecture. This should not replace a full security architecture review (SAR), but is helpful in understanding how the application works and if the team has considered security architecture, produced minimal documentation, and uses risk based decision making in resolving security issues.

| Ref | OWASP ASVS Architecture Requirements (V1) | 1 | 2 | 3 |
| --- | --- | --- | --- | --- |
| 1.1 | (Clarified requirement) Obtain access to all documentation relating to platforms, libraries, components, and custom code. Verify the project’s version of this material is accessible (in place), used by the team, and up to date and accurate. |  | ✓ | ✓ |
| 1.2 | (Clarified requirement) Enumerate all components of the system, including platforms, libraries, and custom code. Verify that the project had this list to hand (control in place), used by the team (such as by build processes), and up to date and accurate. |  | ✓ | ✓ |
| 1.3 | (Clarified requirement) Identify a network diagram, component diagram (such as ER or DFD or similar), data flow (UML use case documentation). Verify that the project had this documentation to hand (control in place), used by the team (such as by build processes), and up to date and accurate. Leading practice is project diagrams and documentation should be built by the build process using code artifacts and other metadata. |  | ✓ | ✓ |
| 1.4 | (Clarified requirement) Verify that the project has a defined software architecture, high level design and detailed design documentation. Verify that the project had this documentation to hand (control in place), used by the team (such as wall diagrams or project wikis), and is up to date and accurate. |  | ✓ | ✓ |
| 1.5 | (Clarified requirement) Verify the provenance of third party libraries, components or code, to ensure that the latest version is obtained prior to any deployment, assurance of security of the third party code is made via static code analysis or via the use of a security contract annex, and the software is deployed with minimal attack surface area. |  | ✓ | ✓ |
| 1.6 | (Clarified requirement) Enumerate any threats and abuse cases that the team have identified and documented, along with automated test cases that prove the application is resilient against such attacks. If there is no threat modeling or abuse cases, create a basic threat model with the project team. |  | ✓ | ✓ |
| 1.7 | Identify all high value data assets. Verify the classifications have been used by the project in developing data protection controls as per the risk classification, including data retention, record based encryption, and audit trails. |  | ✓ | ✓ |
| 1.8 | Identify all audit trails. Verify processes around audit trails are robust against malicious actors, and provide evidence of accountability and continued availability of audit trails except in the most extreme of attacks. |  | ✓ | ✓ |
| 1.9 | Enumerate and verify with the project all high value business flows and transactions within the system. Determine who should perform these transactions. |  | ✓ | ✓ |
| 1.10 | Verify that the project has a risk management system (ISMS) and / or has a defect tracking system, and has used this at least once for treating security and other defects. |  |  | ✓ |
| 1.11 | Verify authentication and communications security between trust boundaries of various components, including clients and all back end systems. Leading practice is that all communication pathways will be authenticated and encrypted. |  |  | ✓ |
| 1.12 | Verify evidence of identity controls for all user roles, including end users, support and administrative staff. Consider two factor authentication, re-authentication for high value transactions, and account management (recovery, registration / new accounts / enrolment, approvals for roles allowing segregation of duties). |  |  | ✓ |
| 1.13 | Verify the application has malicious activity detection controls, such as ESAPI’s governor or AppSensor. Leading practice is an application that can alert and react to unusual events in a configurable way. |  |  | ✓ |
| 1.14 | Verify that the application can be hooked into a SEIM or other log monitoring tool to provide real time feedback. |  |  | ✓ |

## BL – Business Logic Verification Requirements

The Business Logic Verification Requirements define a set of requirements for ensuring that applications process high value transactions in a trusted fashion with adequate detection controls. High value means many different things to different applications. The Application Security Verification Standard considers “high value” to be:

* Personally identifiable information, whether regulated by local or national laws – users and most jurisdictions expect these details to be carefully protected by applications
* Identity information, such as credentials and secondary authentication mechanisms, such as SMS or token information
* Health information that might result in confidentiality loss, such as exposure of highly sensitive patient information
* Financial information that might result in integrity loss, such as fraud or extortion
* Information disclosure that might result in reputation damage or good will loss to users or stakeholders
* Information disclosure that might result in a business loss to users or stakeholders
* Continued availability: Any high value business flow, such as the ability to process transactions in a timely fashion
* Audit trails or detection data: Any ability by normal users to abuse, create, read, update or destroy audit trails or functions is high value
* Any critical data or process relating to the core business asset being exposed by the application

| Ref | OWASP ASVS Business Logic Requirements (BL) | 1 | 2 | 3 |
| --- | --- | --- | --- | --- |
| BL.1 | Verify the application processes or verifies all high value business logic flows in a trusted environment, such as on a protected and monitored server. |  | ✓ | ✓ |
| BL.2 | Verify the application does not allow spoofed high value transactions, such as allowing Attacker User A to process a transaction as Victim User B by tampering with or replaying session, transaction state, transaction or user IDs. |  | ✓ | ✓ |
| BL.3 | Verify the application does not allow high value business logic parameters to be tampered with, such as (but not limited to): price, interest, discounts, PII, balances, stock IDs, etc. |  | ✓ | ✓ |
| BL.4 | Verify the application has defensive measures to protect against repudiation attacks, such as verifiable and protected transaction logs, audit trails or system logs, and in highest value systems real time monitoring of user activities and transactions for anomalies. |  | ✓ | ✓ |
| BL.5 | Verify the application protects against information disclosure attacks, such as direct object reference (see also V4.7 and V4.17) tampering, session brute force or other attacks. |  | ✓ | ✓ |
| BL.6 | Verify the application has sufficient detection and governor controls to protect against brute force (such as continuously using a particular function) or denial of service attacks (see also V4.17). |  | ✓ | ✓ |
| BL.7 | Verify the application has sufficient access controls to prevent elevation of privilege attacks, such as allowing anonymous users from accessing secured data or secured functions, or allowing users to access each other’s details or using privileged functions. |  | ✓ | ✓ |
| BL.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. |  |  | ✓ |
| BL.9 | 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. |  |  | ✓ |
| BL.10 | 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. |  |  | ✓ |

## V2 – Authentication Verification Requirements

The Authentication Verification Requirements define a set of requirements for generating and handling account credentials safely.

| Ref | OWASP ASVS Authentication Requirements (V2) | 1 | 2 | 3 |
| --- | --- | --- | --- | --- |
| 2.1 | Verify all pages and resources require authentication except those specifically intended to be public (Principle of complete mediation). | ✓ | ✓ | ✓ |
| 2.2 | Verify all password fields do not echo the user’s password when it is entered, and that password fields (or the forms that contain them) have autocomplete disabled. | ✓ | ✓ | ✓ |
| 2.3 | Verify that a resource governor is in place to protect against vertical (a single account tested against all possible passwords) and horizontal brute forcing (all accounts tested with the same password e.g. “Password1”). A correct credential entry should incur no delay.  For example, if an attacker tries to brute force all accounts with the single password “Password1”, each incorrect attempt incurs a linear back off (say 5, 25, 125, 625 seconds) with a soft lock of say 15 minutes for that IP address before being allowed to proceed. A similar control should also be in place to protect each account, with a linear back off configurable with a soft lock against the user account of say 15 minutes before being allowed to try again, regardless of source IP address. Both these governor mechanisms should be active simultaneously to protect against diagonal and distributed attacks. | ✓ | ✓ | ✓ |
| 2.4 | Verify all authentication controls are enforced on the server side. |  | ✓ | ✓ |
| 2.5 | Verify all authentication controls (including libraries that call external authentication services) have a centralized implementation. |  | ✓ | ✓ |
| 2.6 | Verify all authentication controls fail securely to ensure attackers cannot log in. |  | ✓ | ✓ |
| 2.7 | (Clarified requirement) Verify password entry fields allow or encourage the use of passphrases, and do not prevent long passphrases or highly complex passwords being entered, and provide a sufficient minimum strength to protect against the use of commonly chosen passwords. |  | ✓ | ✓ |
| 2.8 | Verify all account management functions (such as registration, update profile, forgot username, 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. |  | ✓ | ✓ |
| 2.9 | Verify users can safely change their credentials using a mechanism that is at least as resistant to attack as the primary authentication mechanism. |  | ✓ | ✓ |
| 2.10 | (Clarified requirement) Verify re-authentication, step up or adaptive authentication, SMS or other two factor application, or transaction signing is required before any application-specific sensitive operations are permitted as per the risk profile of the application. |  |  | ✓ |
| 2.11 | (Clarified requirement) Verify authentication credentials can expire after an administratively configurable period of time. |  | ✓ | ✓ |
| 2.12 | Verify that all authentication decisions are logged, including linear back offs and soft-locks. |  | ✓ | ✓ |
| 2.13 | Verify that account passwords are salted using a salt that is unique to that account (e.g., internal user ID, account creation) and hashed before storing. |  | ✓ | ✓ |
| 2.14 | Verify that all authentication credentials for accessing services external to the application are encrypted and stored in a protected location (not in source code). |  | ✓ | ✓ |
| 2.15 | Moved to malicious code section |  |  |  |
| 2.15 | Verify that credentials and all other identity information handled by the application does not traverse unencrypted or weakly encrypted links. | ✓ | ✓ | ✓ |
| 2.16 | Verify that forgot password and other recovery paths do not send the existing or new passwords in clear text to the user. | ✓ | ✓ | ✓ |
| 2.17 | Verify that username enumeration is not possible via login, password reset, or forgot account functionality. | ✓ | ✓ | ✓ |
| 2.18 | Verify there are no default passwords in use for the application framework or any components used by the application (such as “admin/password”). | ✓ | ✓ | ✓ |
| 2.19 | Verify that forgot password and other recovery paths send a time-limited activation token or use two factor proofs (SMS, tokens, mobile application, etc) rather than a password. |  | ✓ | ✓ |
| 2.20 | Verify that forgot password functionality does not lock or otherwise disable the account until after the user has successfully changed their password. |  | ✓ | ✓ |
| 2.21 | Verify that there are no shared knowledge questions/answers (so called "secret" questions and answers). |  | ✓ | ✓ |
| 2.22 | Verify that the system can be configured to disallow the use of a configurable number of previous passwords. |  | ✓ | ✓ |

## V3 – Session Management Verification Requirements

The Session Management Verification Requirements define a set of requirements for safely using HTTP requests, responses, sessions, cookies, headers, and logging to manage sessions properly.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ref | OWASP ASVS Session Management Requirements (V3) | 1 | 2 | 3 |
| 3.1 | Verify that the framework’s default session management control implementation is used by the application. | ✓ | ✓ | ✓ |
| 3.2 | Verify that sessions are invalidated when the user logs out. | ✓ | ✓ | ✓ |
| 3.3 | Verify that sessions timeout after a specified period of inactivity. | ✓ | ✓ | ✓ |
| 3.4 | Verify that sessions timeout after an administratively-configurable maximum time period regardless of activity (an absolute timeout). |  |  | ✓ |
| 3.5 | Verify that all pages that require authentication to access them have logout links. | ✓ | ✓ | ✓ |
| 3.6 | 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. | ✓ | ✓ | ✓ |
| 3.7 | Verify that the session id is changed on login. |  | ✓ | ✓ |
| 3.8 | Verify that the session id is changed on re-authentication. |  | ✓ | ✓ |
| 3.9 | Verify that the session id is changed or cleared on logout. |  | ✓ | ✓ |
| 3.10 | Verify that only session ids generated by the application framework are recognized as valid by the application. |  | ✓ | ✓ |
| 3.11 | Verify that authenticated session tokens are sufficiently long and random to withstand attacks that are typical of the threats in the deployed environment. |  |  | ✓ |
| 3.12 | Verify that authenticated session tokens 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. |  |  | ✓ |
| 3.13 | Moved to malicious code section |  |  |  |
| 3.14 | Verify that authenticated session tokens using cookies are protected by the use of "HttpOnly". | ✓ | ✓ | ✓ |
| 3.15 | Verify that authenticated session tokens using cookies are protected with the "secure" attribute and strict transport security headers (such as Strict-Transport-Security: max-age=60000; includeSubDomains) is present. |  | ✓ | ✓ |

## V4 – Access Control Verification Requirements

The Access Control Verification Requirements assess if access controls at the client (browser), presentation, business logic, and data model are present, in use and effective.

Key principles:

* **Deny by default** – by default, the user should not be able to access resources unless specifically permitted
* **Principle of least privilege** – the user should not be able to execute higher privilege functions or exceed business limits
* **Segregation of duties** – for low value systems, consider SMS authorization for transactions, for high value systems consider initiator / approver / receiver controls.
* **Aggregate access control** – where access to a single secured record or one access to a secured function is acceptable, but scraping the entire database or registering a million users is not desirable or permissible, a resource governor should be used to protect against aggregate access control.

These requirements define verification requirements for access controls for protected URLs, business functions, data, services, and files.

| Ref | OWASP ASVS Access Control Requirements (V4) | 1 | 2 | 3 |
| --- | --- | --- | --- | --- |
| 4.1 | Verify that users can only access secured functions for which they possess specific authorization. | ✓ | ✓ | ✓ |
| 4.2 | Verify that users can only access secured URLs for which they possess specific authorization. | ✓ | ✓ | ✓ |
| 4.3 | Verify that users can only access secured data files for which they possess specific authorization. | ✓ | ✓ | ✓ |
| 4.4 | Verify that direct object references are protected, such that only authorized objects are accessible to each user. | ✓ | ✓ | ✓ |
| 4.5 | Verify that directory browsing is disabled unless deliberately desired. | ✓ | ✓ | ✓ |
| 4.6 | Verify that users can only access protected services for which they possess specific authorization. |  | ✓ | ✓ |
| 4.7 | Verify that users can only access protected data for which they possess specific authorization (for example, protect against direct object reference tampering). | ✓ | ✓ | ✓ |
| 4.8 | Verify that access controls fail securely. |  | ✓ | ✓ |
| 4.9 | (Clarified Requirement) Verify that the same access control rules implied by the presentation layer are enforced on the server side for that user role, such that controls and parameters cannot be re-enabled or re-added from higher privilege users. |  | ✓ | ✓ |
| 4.10 | Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized. |  |  | ✓ |
| 4.11 | Verify that all access controls are enforced on the server side. |  | ✓ | ✓ |
| 4.12 | Verify that there is a centralized mechanism (including libraries that call external authorization services) for protecting access to each type of protected resource. |  | ✓ | ✓ |
| 4.13 | Moved to BL.10 Enforce Business Limits |  |  |  |
| 4.14 | Verify that all access control decisions can be logged and all failed decisions are logged. |  | ✓ | ✓ |
| 4.15 | Moved to malicious code section |  |  |  |
| 4.16 | Verify that the application or framework generates strong random anti-CSRF tokens unique to the user as part of all high value transactions or accessing sensitive data, and that the application verifies the presence of this token with the proper value for the current user when processing these requests. | ✓ | ✓ | ✓ |
| 4.17 | Aggregate access control protection – verify the system can protect against aggregate or continuous access of secured functions, resources, or data. For example, possibly by the use of a resource governor to limit the number of registrations per hour or to prevent the entire database from being scraped by an individual user. |  | ✓ | ✓ |

## 

## V5 – Input Validation Verification Requirements

The Input Validation Requirements define a set of requirements for validating input so that it is safe for use within an application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ref | OWASP ASVS Input Validation Requirements (V5) | 1 | 2 | 3 |
| 5.1 | Verify that the runtime environment is not susceptible to buffer overflows, or that security controls prevent buffer overflows. | ✓ | ✓ | ✓ |
| 5.2 | Verify that a positive validation pattern is defined and applied to all input. | ✓ | ✓ | ✓ |
| 5.3 | Verify that all input validation failures result in input rejection or input sanitization. | ✓ | ✓ | ✓ |
| 5.4 | Verify that a character set, such as UTF-8, is specified for all sources of input. |  | ✓ | ✓ |
| 5.5 | Verify that all input validation is performed on the server side. |  | ✓ | ✓ |
| 5.6 | Verify that a single input validation control is used by the application for each type of data that is accepted. |  | ✓ | ✓ |
| 5.7 | Verify that all input validation failures are logged. |  | ✓ | ✓ |
| 5.8 | Verify that all input data is canonicalized for all downstream decoders or interpreters prior to validation. |  |  | ✓ |
| 5.9 | Moved to malicious code section |  |  |  |
| 5.10 | 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. |  | ✓ | ✓ |
| 5.11 | Verify that the application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (GET, POST, cookies, headers, environment, etc.) |  | ✓ | ✓ |

## 

## V6 – Output Encoding/Escaping Verification Requirements

The Output Encoding/Escaping Validation Requirements define a set of requirements for verifying that output is properly encoded so that it is safe for external applications.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ref | OWASP ASVS Output Encoding/Escaping Requirements (V6) | 1 | 2 | 3 |
| 6.1 | Verify that all untrusted data that are output to HTML (including HTML elements, HTML attributes, javascript data values, CSS blocks, and URI attributes) are properly escaped for the applicable context. | ✓ | ✓ | ✓ |
| 6.2 | Verify that all output encoding/escaping controls are implemented on the server side. |  | ✓ | ✓ |
| 6.3 | Verify that output encoding /escaping controls encode all characters not known to be safe for the intended interpreter. |  | ✓ | ✓ |
| 6.4 | Verify that all untrusted data that is output to SQL interpreters use parameterized interfaces, prepared statements, or are escaped properly. |  | ✓ | ✓ |
| 6.5 | Verify that all untrusted data that are output to XML use parameterized interfaces or are escaped properly. |  | ✓ | ✓ |
| 6.6 | Verify that all untrusted data that are used in LDAP queries are escaped properly. |  | ✓ | ✓ |
| 6.7 | Verify that all untrusted data that are included in operating system command parameters are escaped properly. |  | ✓ | ✓ |
| 6.8 | Verify that all untrusted data that are output to any interpreters not specifically listed above are escaped properly. |  | ✓ | ✓ |
| 6.9 | 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. |  |  | ✓ |
| 6.10 | Moved to malicious code section |  |  |  |

## 

## FR – Files and Remote resource Verification Requirements

The Files and Remote Resource Verification Requirements define a set of requirements for verifying that file handling and remote resource verification prevents common attacks, such as path traversal, local file inclusion (LFI), remote file inclusion (RFI), cross domain resource sharing and malicious file handling.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ref | OWASP ASVS File and Remote Resource Requirements (FR) | 1 | 2 | 3 |
| FR.1 | (Formerly V11.1) Verify that URL redirects and forwards do not include unvalidated data. | ✓ | ✓ | ✓ |
| FR.2 | Verify that filenames and path data obtained from untrusted sources is canonicalized to eliminate path traversal attacks. | ✓ | ✓ | ✓ |
| FR.3 | Verify that files obtained from untrusted sources are stored outside the webroot. | ✓ | ✓ | ✓ |
| FR.4 | Verify that files obtained from untrusted sources are scanned by anti-virus scanners to prevent upload of known malicious content. | ✓ | ✓ | ✓ |
| FR.5 | Verify that parameters obtained from untrusted sources are not used in manipulating filenames, pathnames or any file system object without first being canonicalized and input validated to prevent local file inclusion attacks. | ✓ | ✓ | ✓ |
| FR.6 | Verify that parameters obtained from untrusted sources are canonicalized, input validated, and output encoded to prevent remote file inclusion attacks, particularly where input could be executed, such as header, source, or template inclusion | ✓ | ✓ | ✓ |
| FR.7 | Verify that web or application server is configured by default to deny access to remote resources or systems outside the web or application server. |  | ✓ | ✓ |
| FR.8 | Verify the application code does not execute uploaded data obtained from untrusted sources. |  | ✓ | ✓ |
| FR.9 | Verify if Flash, Silverlight or other rich internet application (RIA) cross domain resource sharing configuration is configured to prevent unauthenticated or unauthorized remote access. | ✓ | ✓ | ✓ |
| FR.10 | Verify remote IFRAMEs and HTML 5 cross-domain resource sharing does not allow inclusion of arbitrary remote content. | ✓ | ✓ | ✓ |

## V7 – Cryptography Verification Requirements

The Encryption Verification Requirements define a set of requirements that can be used to verify an application’s encryption, key management, random number, and hashing operations. Applications should always use FIPS 140-2 validated cryptographic modules, or cryptographic modules validated against an equivalent standard (e.g., a non-U.S. standard).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ref | OWASP ASVS Cryptography Requirements (V7) | 1 | 2 | 3 |
| 7.1 | Verify that all cryptographic functions used to protect secrets from the application user are implemented server side. |  | ✓ | ✓ |
| 7.2 | Verify that all cryptographic modules fail securely. |  | ✓ | ✓ |
| 7.3 | Verify that access to any master secret(s) is protected from unauthorized access (A master secret is an application credential stored as plaintext on disk that is used to protect access to security configuration information). |  | ✓ | ✓ |
| 7.4 | Verify that password hashes are salted when they are created. |  | ✓ | ✓ |
| 7.5 | Verify that cryptographic module failures are logged. |  | ✓ | ✓ |
| 7.6 | Verify that all random numbers, random file names, random GUIDs, and random strings are generated using the cryptographic module’s approved random number generator when these random values are intended to be unguessable by an attacker. |  | ✓ | ✓ |
| 7.7 | Verify that cryptographic modules used by the application have been validated against FIPS 140-2 or an equivalent standard[[11]](#footnote-11). |  |  | ✓ |
| 7.8 | Verify that cryptographic modules operate in their approved mode according to their published security policies. |  |  | ✓ |
| 7.9 | Verify that there is an explicit policy for how cryptographic keys are managed (e.g., generated, distributed, revoked, expired). Verify that this policy is properly enforced. |  |  | ✓ |
| 7.10 | Moved to malicious code section |  |  |  |

## 

## V8 – Error Handling and Logging Verification Requirements

The Error Handling and Logging Verification Requirements define a set of requirements that can be used to verify the tracking of security relevant events and the identification of attack behavior.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ref | OWASP ASVS Error Handling and Logging Requirements (V8) | 1 | 2 | 3 |
| 8.1 | Verify that 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. | ✓ | ✓ | ✓ |
| 8.2 | Verify that all server side errors are handled on the server. |  | ✓ | ✓ |
| 8.3 | Verify that all logging controls are implemented on the server. |  | ✓ | ✓ |
| 8.4 | Verify that error handling logic in security controls denies access by default. |  | ✓ | ✓ |
| 8.5 | Verify security logging controls provide the ability to log both success and failure events that are identified as security-relevant. |  | ✓ | ✓ |
| 8.6 | Verify that each log event includes:   * a time stamp from a reliable source, * severity level of the event, * an indication that this is a security relevant event (if mixed with other logs), * the identity of the user that caused the event (if there is a user associated with the event), * the source IP address of the request associated with the event, * whether the event succeeded or failed, and * a description of the event. |  | ✓ | ✓ |
| 8.7 | Verify that all events that include untrusted data will not execute as code in the intended log viewing software. |  | ✓ | ✓ |
| 8.8 | Verify that security logs are protected from unauthorized access and modification. |  | ✓ | ✓ |
| 8.9 | Verify that there is a single logging implementation that is used by the application. |  | ✓ | ✓ |
| 8.10 | Verify that that the application does not log application-specific sensitive data that could assist an attacker, including user’s session ids and personal or sensitive information. |  | ✓ | ✓ |
| 8.11 | Verify that a log analysis tool is available which allows the analyst to search for log events based on combinations of search criteria across all fields in the log record format supported by this system. |  | ✓ | ✓ |
| 8.12 | Moved to malicious code section |  |  |  |

## 

## V9 – Data Protection Verification Requirements

The Data Protection Verification Requirements define a set of requirements that can be used to verify the protection of sensitive data (e.g., credit card number, passport number, personally identifiable information).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ref | OWASP ASVS Data Protection Requirements (V9) | 1 | 2 | 3 |
| 9.1 | Verify that all forms containing sensitive information have disabled client side caching, including autocomplete features. | ✓ | ✓ | ✓ |
| 9.2 | Verify that the list of sensitive data processed by this application is identified, and that there is an explicit policy for how access to this data must be controlled, and when this data must be encrypted (both at rest and in transit). Verify that this policy is properly enforced. |  | ✓ | ✓ |
| 9.3 | Verify that all sensitive data is sent to the server in the HTTP message body (i.e., URL parameters are never used to send sensitive data). |  |  | ✓ |
| 9.4 | Verify that all cached or temporary copies of sensitive data sent to the client are protected from unauthorized access or purged/invalidated after the authorized user accesses the sensitive data (e.g., the proper no-cache and no-store Cache-Control headers are set). |  | ✓ | ✓ |
| 9.5 | Verify that all cached or temporary copies of sensitive data stored on the server are protected from unauthorized access or purged/invalidated after the authorized user accesses the sensitive data. |  | ✓ | ✓ |
| 9.6 | Verify that there is a method to remove each type of sensitive data from the application at the end of its required retention period. |  |  | ✓ |
| 9.7 | Verify the application minimizes the number of parameters sent to untrusted systems, such as hidden fields, Ajax variables, cookies and header values. |  | ✓ | ✓ |
| 9.8 | 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. |  |  | ✓ |

## 

## V10 – Communication Security Verification Requirements

The Communication Security Verification Requirements define a set of requirements that can be used to verify that all communications with an application are properly secured.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ref | OWASP ASVS Communication Security Requirements (V10) | 1 | 2 | 3 |
| 10.1 | Verify that a path can be built from a trusted CA to each Transport Layer Security (TLS) server certificate, and that each server certificate is valid. | ✓ | ✓ | ✓ |
| 10.2 | Verify that failed TLS connections do not fall back to an insecure connection. |  |  | ✓ |
| 10.3 | Verify that TLS is used for all connections (including both external and backend connections) that are authenticated or that involve sensitive data or functions. |  | ✓ | ✓ |
| 10.4 | Verify that backend TLS connection failures are logged. |  | ✓ | ✓ |
| 10.5 | Verify that certificate paths are built and verified for all client certificates using configured trust anchors and revocation information. |  | ✓ | ✓ |
| 10.6 | Verify that all connections to external systems that involve sensitive information or functions are authenticated. |  | ✓ | ✓ |
| 10.7 | Verify that all connections to external systems that involve sensitive information or functions use an account that has been set up to have the minimum privileges necessary for the application to function properly. |  | ✓ | ✓ |
| 10.8 | 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 (See <http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf> ). |  |  | ✓ |
| 10.9 | Verify that specific character encodings are defined for all connections (e.g., UTF-8). |  |  | ✓ |

## 

## V11 – HTTP Security Verification Requirements

The HTTP Security Verification Requirements define a set of requirements that can be used to verify security related to HTTP requests, responses, sessions, cookies, headers, and logging.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ref | OWASP ASVS HTTP Security Requirements (V11) | 1 | 2 | 3 |
| 11.1 | Moved to Files and Remote Resource Verification Requirements | ✓ | ✓ | ✓ |
| 11.2 | (Clarified requirement) Verify that the application accepts only a defined set of HTTP request methods, such as GET and POST and unused methods are explicitly blocked. | ✓ | ✓ | ✓ |
| 11.3 | Verify that every HTTP response contains a content type header specifying a safe character set (e.g., UTF-8). | ✓ | ✓ | ✓ |
| 11.4 | Verify that the HTTPOnly flag is used on all cookies that do not specifically require access from JavaScript. |  | ✓ | ✓ |
| 11.5 | Verify that the secure flag is used on all cookies that contain sensitive data, including the session cookie. |  | ✓ | ✓ |
| 11.6 | Verify that HTTP headers in both requests and responses contain only printable ASCII characters. |  | ✓ | ✓ |
| 11.7 | (CSRF moved to access control) |  |  |  |
| 11.8 | Verify that HTTP headers and / or other mechanisms for older browsers have been included to protect against click jacking attacks | ✓ | ✓ | ✓ |

## V12 – Security Configuration Verification Requirements

The Security Configuration Verification Requirements define a set of requirements that can be used to verify the secure storage of all configuration information that directs the security-related behavior of the application. Protection of this configuration information is critical to the secure operation of the application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ref | OWASP ASVS Security Configuration Requirements (V12) | 1 | 2 | 3 |
| 12.1 | Verify that all security-relevant configuration information is stored in locations that are protected from unauthorized access. | ✓ | ✓ | ✓ |
| 12.2 | Verify that all access to the application is denied if the application cannot access its security configuration information. |  | ✓ | ✓ |
| 12.3 | Verify that all changes to the security configuration settings managed by the application are logged in the security event log. |  |  | ✓ |
| 12.4 | Verify that the configuration store can be output in a human-readable format to facilitate audit. |  |  | ✓ |
| 12.5 | Verify that detailed error messages are disabled, including stack traces and SOAP faults. |  | ✓ | ✓ |
| 12.6 | Verify that robots.txt and sitemap.xml and any other metadata mechanism does not expose hidden areas of the application that are otherwise unprotected. |  | ✓ | ✓ |
| 12.7 | Verify that all publishing or file update protocols (e.g. WebDAV, sftp, scp) to the webroot are properly protected. |  | ✓ | ✓ |
| 12.8 | Verify that framework opaque state blobs, such as ASP.NET view state, or Spring Web Flow view state is adequately protected from unauthorized replay across users, creation, reading, or updating. |  | ✓ | ✓ |

## 

## V13 – Malicious Code Search Verification Requirements

This section can be performed as an optional extension to a secure code review to validate that on a line-by-line basis that code is not affected by malicious code, such as backdoors, Trojans, salami skimmers, or remote access tools. Examine system clock calls to look for time bombs, functions unrelated to business requirements for back doors, execution paths for Easter eggs, financial transactions for incorrect logic that may indicate a salami attack, other types of malicious code.

The search for malicious code is not easily automated, particularly if looking for obfuscated code.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ref | OWASP ASVS Malicious Code Search Requirements (V13) | 1 | 2 | 3 |
| 13.1 | Verify that no malicious code is in any code that was either developed or modified in order to create the application. |  |  | ✓ |
| 13.2 | Verify that the integrity of interpreted code, libraries, executables, and configuration files is verified using checksums or hashes. |  |  | ✓ |
| 13.3 | Verify that all code implementing or using authentication controls is not affected by any malicious code. |  |  | ✓ |
| 13.4 | Verify that all code implementing or using session management controls is not affected by any malicious code. |  |  | ✓ |
| 13.5 | Verify that all code implementing or using access controls is not affected by any malicious code. |  |  | ✓ |
| 13.6 | Verify that all input validation controls are not affected by any malicious code. |  |  | ✓ |
| 13.7 | Verify that all code implementing or using output validation controls is not affected by any malicious code. |  |  | ✓ |
| 13.8 | Verify that all code supporting or using a cryptographic module is not affected by any malicious code. |  |  | ✓ |
| 13.9 | Verify that all code implementing or using error handling and logging controls is not affected by any malicious code. |  |  | ✓ |

## V14 – Internal Security Verification Requirements

The Internal Security Verification Requirements define a set of requirements that can be used to verify that the application protects itself to an additional degree to guard against implementation flaws.

In ASVS 2.0, a number of mitigations an average development team could easily implement to protect their applications have been drawn directly from the Australian Defense Signals Directorate “Top 35 Strategies to Mitigate Targeted Cyber Intrusions” found at <http://www.dsd.gov.au/publications/Top_35_Mitigations_2012.pdf>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ref | OWASP ASVS Internal Security Requirements (V14) | 1 | 2 | 3 |
| 14.1 | Verify that the application protects user and data attributes and policy information used by access controls from unauthorized access or modification. |  | ✓ | ✓ |
| 14.2 | Verify that security control interfaces are simple enough to use that developers are likely to use them correctly. |  |  | ✓ |
| 14.3 | Verify that the application properly protects shared variables and resources from inappropriate concurrent access. |  |  | ✓ |
| 14.4 | Verify all operating systems, software components, frameworks, libraries, and application platforms are up to date and fully patched so as to have no known security vulnerabilities. (DSD Top 35 #2 and #3) |  | ✓ | ✓ |
| 14.5 | Verify there are a minimal number of named administrator accounts, and all administrative access is fully logged (DSD Top 35 #4) |  | ✓ | ✓ |
| 14.6 | Verify there is no default or shared local administrator accounts (DSD Top 35 #5) |  | ✓ | ✓ |
| 14.7 | Verify remote system administrative access to servers for updates requires multi-factor authentication. DSD Top 35 #6 |  | ✓ | ✓ |
| 14.8 | Verify the use of web application firewalls or similar functional (such as mod-security), and host based firewalls (DSD Top 35 #8 and #9) to protect against common attacks |  | ✓ | ✓ |
| 14.9 | Verify application servers include monitored and regularly updated host based intrusion detection or file integrity monitoring (DSD Top 35 #11) |  |  | ✓ |
| 14.10 | Verify that application platform and operating systems have a time synchronized centralized logging or SEIM capability (DSD Top 35 #12 and #13) for computer and network events |  |  | ✓ |
| 14.11 | Verify that platforms are built from a minimal base to only include features actively required by the application (DSD Top 35 #18) |  |  | ✓ |
| 14.12 | Verify that all operating systems, application server (web, app, database, etc) configurations are hardened as per vendor guidelines (DSD Top 35 #22 and #23) |  | ✓ | ✓ |

# Verification Reporting Requirements

An OWASP ASVS Report contains a description of the application that was analyzed against the OWASP ASVS requirements for a given level. The Report also documents the results of the analysis, including any remediation of vulnerabilities that was required.

The ASVS reporting requirements define the type of information that is required to be present in the report. The ASVS reporting requirements do not define the structure, organization, or format of the report. The ASVS reporting requirements do not preclude additional information from being included in the report.

The type of information that is required by each set of ASVS reporting requirements may be named, formatted, and organized *according to a verifier’s requirements*. *The ASVS reporting requirements are met as long as the required information is present.* A Report should include all material necessary for a reader to understand the analysis that was performed and the results of the analysis, including configuration information and code snippets, as depicted in the adjacent figure, which *may* be used when constructing the Report outline.

Figure 11 – Report Contents

## R1 – Report Introduction

* R1.1 The report introduction shall provide sufficient information to identify both the report and the application that is the subject of the report.
* R1.2 The report introduction shall summarize the overall confidence in the security of the application.
* R1.3 The report introduction shall identify the key business risks associated with operating the application.
* R1.4 The report introduction shall identify the rules of engagement associated with performing the verification or that that may have constrained the scope of the verification.

## R2 – Application Description

* R2.1 The application description shall provide sufficient description of the application to aid the understanding of its operation and the environment that it operates in.

## R3 – Application Security Architecture

* R3.1 The application security architecture shall provide additional detail describing the application as the first step in providing confidence to the reader of the report that the analysis that was performed was both complete and accurate. This part of the Report provides context for the analysis. The information presented in this section will be used in the course of the analysis to identify inconsistencies. This part of the Report shall provide different levels of detail, depending on the OWASP Application Security Verification Standard Level that the analysis was performed. Details will vary according to Level.

## R4 – Verification Results

* R4.1 This verification results shall present the results of the analysis that was performed according to the “Verification Requirements” section of this Standard, including description of any remediation of vulnerabilities that was required, as follows:

Table 2 – OWASP ASVS Report Verification Results Contents

|  |  |  |
| --- | --- | --- |
| Level | Pass | Fail |
| Level 1 Results | Verdict  Tool configuration (if the tool can perform the check) or verdict justification (an argument for completeness and correctness, providing specific evidence)  A mapping of automated tool capabilities to applicable detailed verification requirements  A description of tool configuration and a mapping of tool capabilities need only be provided once as part of the report. | Verdict  Location (URL w/parameters and/or source file path, name and line number(s))  Description (including configuration information as appropriate)  Risk rating[[12]](#footnote-12)  Risk justification |
| A description of tool configuration and a mapping of tool capabilities must also be provided as part of the report. | |
| Levels 2 – 3 Results | Verdict  Verdict justification (an argument for completeness and correctness, providing specific evidence) | Verdict  Location (URL w/parameters and/or source file path, name and line number(s))  Description (including path through application components and steps to reproduce)  Risk rating (see the OWASP Risk Rating Methodology)  Risk justification |

# 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.

*Application Component* – An individual or group of source files, libraries, and/or executables, as defined by the verifier for a particular application.

*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.

*Application Security Verification Standard (ASVS)* – An OWASP standard that defines four levels of application security verification for applications.

*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.

*Common Criteria (CC)* – A multipart standard that can be used as the basis for the verification of the design and implementation of security controls in IT products.

*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.

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

Internal Verification – The technical assessment of specific aspects of the security architecture of an application as defined in the OWASP ASVS.

*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.

*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

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

*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 Validation* – The canonicalization and validation of application output to Web browsers and to external systems.

*OWASP Enterprise Security API (ESAPI)* – A free and open collection of all the security methods that developers need to build secure Web applications. See: <http://www.owasp.org/index.php/ESAPI>

*OWASP Risk Rating Methodology* – A risk rating methodology that has been customized for application security. See: <http://www.owasp.org/index.php/How_to_value_the_real_risk>

*OWASP Testing Guide* – A document designed to help organizations understand what comprises a testing program, and to help them identify the steps needed to build and operate that testing program. See: <http://www.owasp.org/index.php/Category:OWASP_Testing_Project>

*OWASP Top Ten* – A document that represents a broad consensus about what the most critical Web application security flaws are. See: <http://www.owasp.org/index.php/Top10>

*Positive* – See whitelist.

*Salami Attack* – A type of malicious code that is used to redirect small amounts of money without detection in financial transactions.

*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 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).

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

*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 an 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.

*Time Bomb* – A type of malicious code that does not run until a preconfigured time or date elapses.

*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.

# Where To Go From Here

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 Top Ten Project* - <http://www.owasp.org/index.php/Top_10>
* *OWASP Code Review Guide* - <http://www.owasp.org/index.php/Category:OWASP_Code_Review_Project>
* *OWASP Testing Guide* - <http://www.owasp.org/index.php/Testing_Guide>
* *OWASP Enterprise Security API (ESAPI) Project* - <http://www.owasp.org/index.php/ESAPI>
* *OWASP Legal Project* - <http://www.owasp.org/index.php/Category:OWASP_Legal_Project>

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

* *OWASP* - <http://www.owasp.org>
* *MITRE* - Common Weakness Enumeration – Vulnerability Trends, <http://cwe.mitre.org/documents/vuln-trends.html>
* *PCI Security Standards Council* - publishers of the PCI standards, relevant to all organizations processing or holding credit card data, [https://www.pcisecuritystandards.org](https://www.pcisecuritystandards.org/)
* *PCI Data Security Standard (DSS) v1.1* - <https://www.pcisecuritystandards.org/pdfs/pci_dss_v1-1.pdf>

1. For more information about building and using security controls that meet ASVS requirements, see the [*Enterprise Security API (ESAPI)*](http://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API)(OWASP, 2009). [↑](#footnote-ref-1)
2. For more information about using ASVS in contracts, see the [*Contract Annex*](http://www.owasp.org/index.php/Category:OWASP_Legal_Project) (OWASP, 2009). [↑](#footnote-ref-2)
3. For more information about common Web application vulnerabilities, see the *OWASP* [*Top Ten*](http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project) (OWASP, 2013). [↑](#footnote-ref-3)
4. For information about how to specify an ASVS level in a contract, see the OWASP [Contract Annex](http://www.owasp.org/index.php/Category:OWASP_Legal_Project). [↑](#footnote-ref-4)
5. For more information about how to ESAPI-Enable (ES-Enable) your application, see the OWASP [ESAPI](http://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API) project (OWASP 2009). [↑](#footnote-ref-5)
6. For more information about introducing security-related activities into your existing SDLC, see the OWASP CLASP (OWASP 2008) or OWASP SAMM Projects (OWASP 2009). [↑](#footnote-ref-6)
7. For more information about identifying risks and estimating risks associated with vulnerabilities, see the [OWASP Testing Guide](http://www.owasp.org/index.php/Category:OWASP_Testing_Project) (OWASP, 2008). [↑](#footnote-ref-7)
8. For more information about identifying risks, estimating risks associated with vulnerabilities, and performing manual verification by performing manual penetration testing, see the [OWASP Testing Guide](http://www.owasp.org/index.php/Category:OWASP_Testing_Project) (OWASP, 2008). [↑](#footnote-ref-8)
9. For more information about performing manual verification by performing a manual code review, see the [OWASP Code Review Guide](http://www.owasp.org/index.php/Category:OWASP_Code_Review_Project) (OWASP, 2008). [↑](#footnote-ref-9)
10. Malicious code is not the same as malware. See the glossary definition of malicious code. [↑](#footnote-ref-10)
11. See <http://csrc.nist.gov/groups/STM/cmvp/validation.html> [↑](#footnote-ref-11)
12. For more information about identifying risks and estimating risks associated with vulnerabilities, see the [Testing Guide](http://www.owasp.org/index.php/Category:OWASP_Testing_Project) (OWASP, 2008). [↑](#footnote-ref-12)