

Application Security Verification Standard 3.0.1

2015年12月

謝辞 6

本標準について 6

著作権とライセンス 6

バージョン 3.0 (2015) 6

バージョン 2.0 (2014) 6

バージョン 1.0 (2009) 7

序文 9

バージョン 3.0 の更新内容 9

Application Security Verification Standard の使用 11

アプリケーションセキュリティの検証レベル 12

本標準の採用方法 12

レベル 1: 導入 (Opportunistic) 12

レベル 2: 標準 13

レベル 3: 上級 13

ASVS 適用の実際 14

採用事例 17

採用事例 1: セキュリティテストの指針として 17

採用事例 2: セキュアな開発ライフサイクルとして 18

検証レベルの達成を評価する 19

ASVS 認定と信頼マークに対する OWASPの見解 19

認定組織に対する指針 19

自動侵入検査ツールの役割 19

侵入検査の役割 20

セキュリティアーキテクチャの詳細な指針として 20

既存のセキュアコーディングチェックリストに代わるものとして 20

自動の単体および統合テストのガイドとして 20

セキュア開発のためのトレーニングとして 21

ASVSを採用しているOWASP のプロジェクト 22

Security Knowledge Framework 22

OWASP Zed Attack Proxy 22

OWASP Cornucopia 22

検証要件の詳細 23

V1: アーキテクチャ，設計，脅威モデリング 24

管理目標 24

要件 24

参考情報 25

V2: 認証に関する検証要件 26

管理目標 26

要件 26

参考情報 28

V3: セッション管理に関する検証要件 29

管理目標 29

要件 29

参考情報 30

V4: アクセス制御に関する検査要件 31

管理目標 31

要件 31

参考情報 32

V5: 悪性入力の処理に関する検証要件 33

管理目標 33

要件 33

参考情報 35

V6: 出力のエンコード / エスケープ 37

V7: 暗号化に関する検証要件 38

管理目標 38

要件 38

参考情報 39

V8: エラー処理とログの保存に関する検証要件 40

管理目標 40

要件 40

参考情報 41

V9: データの保護に関する検証要件 42

管理目標 42

要件 42

参考情報 44

V10: 通信のセキュリティに関する検証要件 45

管理目標 45

要件 45

参考情報 46

V11: HTTP のセキュリティ設定に関する検証要件 48

管理目標 48

要件 48

参考情報 49

V12: セキュリティ設計に関する検証要件 50

V13: 悪性活動の管理に関する検証要件 51

管理目標 51

要件 51

参考情報 51

V14: 内部セキュリティに関する検証要件 52

V15: ビジネスロジックに関する検証要件 53

管理目標 53

要件 53

参考情報 53

V16: ファイルとリソースに関する検証要件 55

管理目標 55

要件 55

参考情報 56

V17: モバイルに関する検証要件 57

管理目標 57

要件 57

参考情報 58

V18: Web サービスに関する検証要件 59

管理目標 59

要件 59

参考情報 60

V19. 構成 61

管理目標 61

要件 61

参考情報 62

付録 A: 廃止された要件 63

付録 B: 用語集 69

付録 C: 参考情報 73

付録 D: 他の標準との対応 74

# 謝辞

## 本標準について

Application Security Verification Standard (アプリケーションセキュリティ検証標準) とは，アプリケーションのセキュリティ要件またはセキュリティテストの項目です．アプリケーションの設計担当者，開発者，テスト担当者，セキュリティプロフェッショナル，ユーザーは，本標準を使用することでセキュアなアプリケーションとは何かを定義することができます．

## 著作権とライセンス

Copyright © 2008 – 2015 The OWASP Foundation. 本書は，クリエイティブコモンズ表示—継承 3.0 ライセンスに基づいて公開されています．再使用または頒布する場合は，他者に対して本著作物のライセンス条項を明らかにする必要があります．

### バージョン 3.0 (2015)

|  |  |  |
| --- | --- | --- |
| プロジェクトリーダー | 主執筆者 | 共同執筆者およびレビュー担当者 |
| Andrew van der Stock  Daniel Cuthbert | Jim Manico | Boy Baukema  Ari Kesäniemi  Colin Watson  François-Eric Guyomarc’h  Cristinel Dumitru  James Holland  Gary Robinson  Stephen de Vries  Glenn Ten Cate  Riccardo Ten Cate  Martin Knobloch  Abhinav Sejpal  David Ryan  Steven van der Baan  Ryan Dewhurst  Raoul Endres  Roberto Martelloni |

### バージョン 2.0 (2014)

| プロジェクトリーダー | 主執筆者 | 共同執筆者およびレビュー担当者 |
| --- | --- | --- |
| Daniel Cuthbert  Sahba Kazerooni | Andrew van der Stock  Krishna Raja | Antonio Fontes  Colin Watson  Jeff Sergeant  Pekka Sillanpää  Archangel Cuison  Dr Emin Tatli  Jerome Athias  Safuat Hamdy  Ari Kesäniemi  Etienne Stalmans  Jim Manico  Scott Luc  Boy Baukema  Evan Gaustad  Mait Peekma  Sebastien Deleersnyder |

### バージョン 1.0 (2009)

|  |  |  |
| --- | --- | --- |
| プロジェクトリーダー | 主執筆者 | 共同執筆者およびレビュー担当者 |
| Mike Boberski  Jeff Williams  Dave Wichers | Jim Manico | Andrew van der Stock  Dr. Sarbari Gupta  John Steven  Pierre Parrend  Barry Boyd  Dr. Thomas Braun  Ken Huang  Richard Campbell  Bedirhan Urgun  Eoin Keary  Ketan Dilipkumar Vyas  Scott Matsumoto  Colin Watson  Gaurang Shah  Liz Fong Shouvik Bardhan  Dan Cornell  George Lawless  Mandeep Khera  Stan Wisseman  Dave Hausladen  Jeff LoSapio  Matt Presson  Stephen de Vries  Theodore Winograd  Jeremiah Grossman  Nam Nguyen  Steve Coyle  Dave van Stein  John Martin  Paul Douthit  Terrie Diaz |

# 序文

Application Security Verification Standard (ASVS) バージョン 3.0 へようこそ。ASVS は、最新の Web アプリケーションを設計、開発、テストするときに必要となる，セキュリティ要件および管理策のフレームワークを確立することを目指した、コミュニティによる取組みであり、特に、機能的および非機能的なセキュリティ管理策の標準化に重点をおいています。

ASVS v3.0 は、コミュニティにおける取組みと業界からのフィードバックの成果です。本リリースでは、ASVS の採用に関する事例を示すことが重要であると考えました。事例をみることで、ASVS に初めて接する企業，本標準の採用計画を容易に行えるようになり、既に採用している企業は他の採用企業の経験から学ぶことができるでしょう。

本標準の内容に 100% 同意いただけるとは我々も考えていません。リスク分析には主観が常につきものです．あらゆる対象に適用可能な標準として一般化することには困難が伴います。しかし我々は、この版で行った最新の更新が、正しい方向へ踏み出す一歩となること、そして、この重要な業界標準に導入された概念を、尊重しつつもさらに強化することを期待しています。

## バージョン 3.0 の更新内容

バージョン 3.0 では、最新なアプリケーションへの本標準の適用可能性を強化するため、構成、Web サービス、モダンな (クライアントベースの) アプリケーションなどに関する新たなセクションを追加しました。モダンなアプリケーションとは、一般に応答性を備えるアプリケーションであり、HTML5を活用したフロントエンドやモバイルクライアントから、SAML 認証を使用して共通の RESTful Web サービス群を呼び出すようなものを想定しています。

また、たとえば、モバイル開発者が同じ項目を何度も再テストしなくて済むように、標準内の重複を排除しました。

CWE 共通脆弱性一覧 (Common Weakness Enumeration: CWE) との対応表を作成しました。CWE 対応表を活用することで、脆弱性の悪用可能性や悪用された場合の影響を明らかにすることができます。平たく言えば、セキュリティ管理策が用いられていない、あるいは効果的に実装されていない場合に，どのような問題が発生するか、また、脆弱性の影響をどのように軽減できるかの洞察を得る助けとなるということです。

これまで我々は、コミュニティに働きかけ、AppSec EU 2015 ではピアレビューセッションを、AppSec USA 2015 では最終的なワーキングセッションを行い、コミュニティのフィードバックを大量に取り込んできました。ピアレビュー時に管理策の意味が大幅に変更されたときは、新たな管理策を作成し、既存の管理策は廃止しました。廃止した管理策は、混乱を招く可能性があるため、あえて再利用しないこととしました。変更内容は、付録 A に包括的な対応表として収録しています。

まとめると、v3.0 ではこれまでで最も大きな変更が行われています。今回の更新が皆様にとって有益なものとなり、さまざまな方法で活用されることを期待しています。

# Application Security Verification Standard の使用

ASVS には次の2つの主目的があります．

* 組織におけるセキュアなアプリケーション開発と保守を支援する
* セキュリティサービス，セキュリティツールベンダ，およびユーザが，製品やサービスをセキュリティ要件に合致させる



図 1 – 組織およびツール/サービス提供者におけるASVSの活用

## アプリケーションセキュリティの検証レベル

Application Security Verification Standard では、3 段階のセキュリティ検査レベルを定義しており、レベルが上がるごとに深度が増します。

* ASVS レベル 1 は、すべてのソフトウェアを対象とする
* ASVS レベル 2 は、保護を必要とする機微なデータを含むアプリケーションを対象とする
* ASVS レベル 3 は、最も重要度が高いアプリケーション、つまり、最高レベルの信頼性を必要とするあらゆるアプリケーション (高い価値を伴う取引を行うものや、機微な医療データを含むものなど) を対象とする

各 ASVS レベルでは、セキュリティ要件を列挙しています。また、個々の要件は、ソフトウェアに組み込まれるべきセキュリティ上の機能に対応付けることができます。



図 2 - OWASP Application Security Verification Standard 3.0 のレベル

## 本標準の採用方法

Application Security Verification Standard の最適な使用方法の 1 つは、自社のアプリケーション、プラットフォーム、または組織固有のセキュアコーディングチェックリスト作成の雛形として使用することです。ユースケースに合うようにASVS を変更することで、プロジェクトや環境で最も重要なセキュリティ要件により注力することができます。

### レベル 1: 導入 (Opportunistic)

OWASP Top 10 や同類のチェックリストに含まれる容易に検出可能な脆弱性に対する防御が適切に行われているアプリケーションは、ASVS レベル 1 (導入) を満たします。

通常、レベル 1 は、セキュリティ管理策を正しく使用しているが、それほど厳密さが求められないアプリケーションに適しています。また、エンタープライズアプリケーションの簡易分析のためや、複数のフェーズからなる活動の1つとして，セキュリティ要件の優先順位リストの作成を補助するために適しています。レベル 1 の管理策は、ツールを使用して自動的に確認することもできれば、手動で (ソースコードにアクセスすることなく) 確認することもできます。レベル 1 は、すべてのアプリケーションが最低限満たすべきレベルであると考えています。

アプリケーションに対する脅威の多くは、単純で容易に使える技術を用い、簡単に発見できて悪用が容易な脆弱性を発見する攻撃者に由来するものです。これとは対照的に、明確な動機を持つ攻撃者は、全エネルギーを集中させて特定のアプリケーションに狙いを定めてきます。アプリケーションで価値の高い重要なデータを処理している場合、レベル 1 では不十分でしょう。

### レベル 2: 標準 (Standard)

今日のソフトウェアが直面するほとんどのリスクに適切に対応するアプリケーションは、ASVS レベル 2 (標準) を満たします。

レベル 2 では、アプリケーションにセキュリティ管理策が存在し、有効であり、活用されていることを保証します。通常、レベル 2 は、企業間のトランザクションを大量に処理するアプリケーション (医療情報を処理する、事業上重要もしくは機微な機能を実装する、その他の機微な資産を処理するなど) に適用されます。

レベル 2 アプリケーションに対する脅威は、高いスキルと明確な動機を持つ攻撃者によるものであり、攻撃者は，アプリケーションの脆弱性の検出と攻略に定評のあるツールや技法を使い、特定の標的を攻撃します。

### レベル 3: 上級 (Advanced)

レベル 3 はASVSで最も高い検査レベルです。レベル 3 は、軍事、保健と安全、重要インフラストラクチャなどの分野で使われる高度なセキュリティ検査レベルを必要とするアプリケーションを対象としています。障害の発生が組織の運営、さらにはその存続に大きな影響を及ぼしうる重要な機能を担うアプリケーションに対しては、ASVS レベル 3 が必要となるでしょう。ASVS レベル 3 の適用に関するいくつかの指針を後段に示します。高度な脆弱性に対する対策が適切に行われた、正しいセキュリティ設計の原則を実践しているアプリケーションは、ASVS レベル 3 (上級) を満たします。

ASVS レベル 3 のアプリケーションには、他のどのレベルよりも綿密な分析、アーキテクチャ、コーディング、およびテストが必要です。セキュアなアプリケーションは、(弾力性やスケーラビリティの確保、また多層的セキュリティの実現のために)適切にモジュール化されており、(ネットワーク接続によって、または物理的に分離された)各モジュールは与えられたセキュリティ上の役割を果たします (多層防御)。また，各モジュールの役割は，適切に文書化される必要があります。セキュリティ上の役割には、機密性(暗号化など)、完全性 (トランザクション、入力検証など)、可用性 (負荷の適切な処理など)、認証 (システム間の認証など)、否認防止、認可、および監査 (ログの取得) などの確保があります。

## ASVS 適用の実際

脅威が異なればその動機も異なります。業界によっては、固有の情報資産や技術資産を持ち、その分野における法令遵守要件が適用されるものがあります。

以下の表に、業界別の推奨される ASVS レベルに関する指針を示します。業界における脅威にはそれぞれの独自の基準があり，脅威の種類も異なるかもしれませんが、すべての業界に共通する次の問題があります。つまり，日和見的な攻撃者は容易に悪用できる脆弱なアプリケーションを探す、ということです。ASVS レベル 1 が業界を問わずあらゆるアプリケーションに推奨されるのはそのためであり、容易に見つかるリスクを管理する出発点として推奨されます．組織は、事業の性質に基づいて、自社に固有のリスク特性をより綿密に調査することが強く求められます．対極にあるのが ASVS レベル 3 です。レベル 3 は、生命の安全を脅かしかねない場合やアプリケーションが完全に侵害された際に組織へ甚大な影響が及ぼしうる場合に対応しています。

| 業界 | 業界のプロファイル | L1 推奨対象 | L2 推奨対象 | L3 推奨対象 |
| --- | --- | --- | --- | --- |
| 金融，保険 | 日和見的な攻撃者からの攻撃にもさらされますが、金銭詐取という明確な動機を持つ攻撃者が、高い価値を持つ標的を狙うことが多くなっています。攻撃者は通常、機密データや口座資格情報を探し、不正行為を働いたり、アプリケーション組込みの送金機能を使って直接的に利益を得ようとします。多くの場合、窃盗された認証情報の使用、アプリケーションに対する攻撃、ソーシャルエンジニアリングなどが攻撃手法です。  遵守すべき主要な法令として、PCI データセキュリティスタンダード (PCI DSS)、グラムリーチブライリー法、米国企業改革法 (SOX) などがあります。 | ネットワークアクセス可能なすべてのアプリケーション。 | 一定の金額を限定的な方法で移動することができる、クレジットカード番号や個人情報などの機密情報を含むアプリケーション。例として、次の場合があります。  (i) 同一機関の口座間の送金  (ii) 取引限度額がある、時間のかかる資金移動形式 (自動決済機関など)  (iii) 一定期間内における送金額の限度額がある電信送金 | 大量の機密情報を含むアプリケーション、あるいは多額の迅速な送金 (電信送金など) や個別取引形式、少額送金の一括処理形式による多額の送金を実行できるアプリケーション。 |
| 製造、輸送、テクノロジ、公益事業、インフラ、防衛 | これらの業界にはさほど共通点がないと思うかもしれませんが、これら業界の組織を狙う攻撃者は、より多くの時間と技術力とリソースを使って集中的な攻撃を行う傾向が強まっています。多くの場合、機密情報やシステムの特定は容易ではなく、内通者やソーシャルエンジニアリング技法の活用を必要とします。攻撃は、内通者、部外者、あるいはその両方の共謀によって行われる場合があります。攻撃者の目的には、知的財産にアクセスして戦略的または技術的な利益を得ることなどがあります。また、アプリケーションの機能を悪用して機密性の高いシステムの動作に影響を及ぼしたり、このようなシステムを中断させようとする攻撃者も見逃すわけにはいきません。  攻撃者の大半は、個人識別情報や支払データなど直接的または間接的な利用によって利益につながる機密データを探しています。多くの場合、このようなデータは、なりすまし犯罪や不正支払いなど、さまざまな不正行為の計画に利用されます。 | ネットワークアクセス可能なすべてのアプリケーション。 | ソーシャルエンジニアリングに使われる可能性がある内部情報や従業員情報を含むアプリケーション。必須ではないが重要な知的財産または企業秘密を含むアプリケーション。 | 組織の存続や成功に不可欠な、高い価値を持つ知的財産、企業秘密、国家機密 (たとえば、米国の場合、"Secret" 以上のカテゴリに分類されるあらゆる情報) を含むアプリケーション。セキュリティに影響のある機能 (輸送、製造装置、制御システムなど) を制御するアプリケーション、または人命に関わるアプリケーション。 |
| 医療 | ほとんどの攻撃者は、直接的または間接的な利用によって利益を得ようと、個人識別情報や支払データなどの機密データを探しています。多くの場合、このようなデータは、なりすまし犯罪や不正支払いなど、さまざまな不正行為の計画に利用されます。  米国の医療業界の場合、医療保険の相互運用性と説明責任に関する法律 (HIPAA) のプライバシー、セキュリティ、侵害開示に関する規則と患者の安全性に関する規則 (<http://www.hhs.gov/ocr/privacy/>) があります。 | ネットワークアクセス可能なすべてのアプリケーション。 | センシティブな医療情報 (保護医療情報)、個人識別情報、または支払データを小・中規模含むアプリケーション。 | 人命に関わる医療装置、機器、または記録の制御に使用されるアプリケーション。詐欺に利用できるトランザクションデータを大量に含む支払システムや POS システム。これらのアプリケーションの管理インターフェイスも含まれます。 |
| 小売、食品、接客 | 行き当たりばったりに "力尽くで金品を奪う" のが、この分野の攻撃者によく見られる攻撃方法です。一方で、支払情報の保有、金融取引、個人識別情報の保存を行うアプリケーションに対する、特定の攻撃の脅威も存在します。前述の脅威よりも頻度は低いものの、業界を狙ったより高度な攻撃の可能性があり、知的財産の詐取、競合他社の情報収集、交渉中の組織やビジネスパートナーを出し抜くことを目的に行われます。 | ネットワークアクセス可能なすべてのアプリケーション。 | ビジネスアプリケーション、製品カタログ情報、社内情報に適するほか、限定的なユーザー情報 (連絡先情報など) を含むアプリケーションに適します。小・中規模の支払データまたは精算機能を含むアプリケーション。 | 詐欺に利用されうる、トランザクションデータが大量に含まれる支払システムや POS システム。これらのアプリケーションの管理インターフェイスも含まれます。完全なクレジットカード番号、母親の旧姓、社会保障番号などの大量の機密情報が含まれるアプリケーション。 |

# 採用事例

## 採用事例 1: セキュリティテストの指針として

## 米国ユタ州のある私立大学では、学内の Red Team が、アプリケーションのペネトレーションテスト実施のガイドとして OWASP ASVS を使用しています。OWASP ASVS は，ペネトレーションテストのプロセス全体（最初の計画と対象検討のためのミーティングからテストの実施指針まで）において、また、クライアント向け最終報告書にテスト結果まとめるための枠組みとして，使用されています。また、Red Team はチームのトレーニング計画にも ASVS を活用しています。

## 学内 Red Team は、大学全体の情報セキュリティ戦略の一環として、校内のさまざまな学部に対して、ネットワークとアプリケーションのペネトレーションテストを実施しています。最初の計画ミーティングの際、クライアントは多くの場合、学生チームがアプリケーションをテストすることに後ろ向きです。しかし、ASVS を紹介し、テストが ASVS 標準に則って行われ、最終報告書には標準に対する達成度合いが記されることを説明すると、ほとんどのクライアントは安心して任せてくれます。次に、テスト範囲の検討において、ASVSを使ってテストに要する時間と作業が見積もられます。Red Team は、あらかじめ定義されたASVSの検査レベルを使い，リスクに基づいたテストが行われることを説明します。ASVSの活用により，クライアント、関係者、そしてRed Team はテストの対象となるアプリケーションの適切なスコープについて合意します。

## ひとたびテストが始まると、Red Team は ASVS を使って活動を計画し、作業量の分割を行います。チームのプロジェクトマネージャーは、各検証要件のテストの状況 (完了、保留中など) を追跡することで、テストの進捗状況を容易に把握できます。これにより、クライアントとよりスムーズにコミュニケーションを取れるようになり、プロジェクトマネージャーはより適切にリソースを管理できるようになります。Red Team は主に学生から構成されているため、チームメンバーの大半は複数の講義に時間を割かなくてはなりません。しかし、個々の検証要件またはカテゴリ全体に基づいてタスクが適切に定義されているため、チームメンバーは、テストすべき内容を正確に把握することができ、また個々のタスク完了までにかかる時間を正確に見積もることができます。報告書作成も，ASVS は構成が明確であるため，簡単です。チームメンバーは、検査結果を記述してから次のタスクに進むことができ、ペネトレーションテストの進行に合わせて報告書の大半を効果的に作成することができます。

## Red Team は ASVS に沿って最終報告書を構成し、各検証要件のステータスを報告し、必要に応じて詳細を提供します。そのため、クライアントや関係者は、ASVS 標準に照らし合わせてアプリケーションの状況を適切に把握できます。また、最終報告書は、その後のセキュリティの向上や低下を知る上での目安になるので、あとに続く取組みにも非常に役立ちます。さらに、報告書形式が ASVS と密接に連携しているため、特定のカテゴリにおけるアプリケーションの達成度合いに関心がある関係者は、その情報を容易に見つけ出すことができます。また、ASVS は構成が明確なため、新たなチームメンバーに報告書の作成方法を教えることが以前の報告書形式よりも容易になりました。

## また、ASVS を採用することで、Red Team のトレーニングは改善しています。以前は、チームリーダーまたはプロジェクトマネージャーが選択したトピックを中心とするトレーニングが週に 1 回実施されていました。トピックは、チームメンバーからのリクエストや、ニーズに基づいて選択されていました。このような基準に基づくトレーニングは、チームメンバーのスキルを広げる可能性があるとはいえ、Red Team の中心的な活動に必ずしも関連しているわけではありませんでした。つまり、ペネトレーションテストにおけるチームのスキルはそれほど向上していませんでした。ASVS の採用後、チームのトレーニングは、個々の検証要件のテスト方法に焦点を絞って実施されるようになりました。その結果、個々のチームメンバーの測定可能なスキルと最終報告書の品質が大幅に向上しました。

## 採用事例 2: セキュアな開発ライフサイクルとして

金融機関向けにビッグデータ分析機能の提供を目論むスタートアップ企業は、金融メタデータへのアクセスとデータ処理のための最優先要件は開発におけるセキュリティであることに気付きました。そこで、アジャイル SDLC (ソフトウェア開発ライフサイクル) の基盤として ASVS を採用しました。

スタートアップ企業はASVS を使用し、ログイン機能の最適な実装方法などの機能的なセキュリティの問題について、ユーザーストーリーとユースケースを生成します。スタートアップ企業のASVSの活用方法は通常とは異なるものでした。まず ASVS を精査し、現在のスプリントに適合する要件を拾い出します。そして、機能的要件はスプリントのバックログに直接追加し、非機能的要件は制約として既存のユースケースに追加します。たとえば、TOTP 2 要素認証の追加を選択し，同時にパスワードポリシーとブルートフォースの検出および防止機構の効果を倍増させる Web サービスレギュレータを追加するという具合です。以降のスプリントでは、その他の要件を "JIT" (just in time = 土壇場で追加する)、"YAGNT" (You ain't gonna need it = 必要でないなら追加しない) 方式で選択します。

開発者は ASVS をピアレビュー用チェックリストとして使用し、安全でないコードがチェックインされないようにします。また、ASVS を使って、過去に遡って新機能をチェックインした開発者を綿密に調査して，適切な ASVS 要件を考慮していることを確認し、将来のスプリントで改善または緩和できるものはないかをチェックします。

最後に、開発者は、ASVS を自動検査のセキュアな単体 / 統合テストの一部として使用し、ユーステストケース、悪用テストケース、ファジーテストケースをそれぞれテストします。目的は、マイルストーンのビルドを製品としてリリースする場合に、ウォーターフォール手法における "開発の最終段階のペネトレーションテスト" で生じるリスクを軽減することにあります。各スプリント後に新規のビルドが昇格される可能性があるため、単一の保証に頼るのは不十分です。テスト体制を自動化することで、熟練したペネトレーションテスト担当者が数週間テストしても重大な問題は見つからないでしょう．

# 検証レベルの達成を評価する

## ASVS 認定と信頼マークに対する OWASPの見解

OWASP は、ベンダー中立の非営利組織であり、ベンダー、認証者、ソフトウェアの認定は一切行っていません。

ASVS に関する保証の表明、信頼マーク、認定は、いずれも OWASP によって正式に検査や登録、または認定されたものではありません。組織は、ASVS 認定を主張するあらゆる第三者または信頼マークについて、その信頼性を慎重に検討する必要があります。

ただし、OWASP の正式な認定であると主張しない限り、組織がこのような保証サービスを提供することを妨げるものではありません。

## 認定組織に対する指針

## Application Security Verification Standard は、アプリケーションの公開検証にも活用することができます。公開検証においては、設計担当者や開発者、プロジェクトドキュメント、ソースコード、特に L2 および L3 検査用に使用できるテストシステムへの認証アクセス (各役割について最低 1 つのアカウントへのアクセス) など、アプリケーションの重要リソースに対して制限なく自由にアクセスすることができます。

## 歴史的に、ペネトレーションテストとセキュアなコードレビューでは問題を例外として取り扱ってきました。つまり、テストに不合格であった問題のみが最終報告書に現れるというわけです。認定機関は、報告書に、検証の範囲 (特に、SSO 認証などの重要な構成要素が範囲外である場合) 、検査結果の要約（合格したテストと不合格だったテスト）、および不合格のテストに対する解決策の明確な指示を必ず含める必要があります。

## 詳細な作業文書、スクリーンショットや動画、問題を確実かつ反復的に再現するスクリプト、テストの電子的記録 (たとえば、プロキシのログや、クリーンアップリストなどの関連するメモ) を保存しておくことは、業界の標準的な慣行と見なされており、懐疑的な開発者に対する検証結果の証拠として非常に役立つことがあります。単にツールを実行してエラーを報告するだけでは不十分です。それだけでは、認定対象のレベルのすべての問題を完全にテストしたと裏付ける十分な証明には (まったく) なりません。論争が発生したときは、検証対象の要件のすべてをそれぞれテストしたことを確実に実証する十分な証明が必要になります。

## 自動ペネトレーションテストツールの役割

自動のペネトレーションテストツールは、できるだけ広いカバレッジを提供し、かつ多くの異なる形式の悪性入力をテストできることが推奨されます。

## 自動ペネトレーションテストツールのみを使用して、ASVS の検査のすべてを完全に実行することはできません。L1 の要件については、その大半を自動テストで検査できますが、全体的には、ほとんどの要件が自動ペネトレーションテストにはなじみません。

## ただし、自動テストと手動テストとの間の線引きは、アプリケーションセキュリティ産業の成熟化が進むにつれて、不明確になってきています。自動ツールは多くの場合、技術担当者によって手動調整され、また手動テストの担当者がさまざまな自動ツールを活用することもよくあります。

## ペネトレーションテストの役割

## ソースコードにアクセスせず、手動によるペネトレーションテストを通じてL1 のすべての問題を検査することは可能ですが、主流の手法ではありません。L2 では、開発者、ドキュメント、コードへの一定のアクセスと、システムへの認証アクセスが必要になります。レベル 3 では、ペネトレーションテストですべてをカバーするのは不可能です。なぜなら、レベル 3 でカバーされる問題の大半が、システム構成のレビュー、悪性コードのレビュー、脅威モデリングといったペネトレーションテスト以外の成果物を必要とするからです。

## セキュリティアーキテクチャの詳細な指針として

Application Security Verification Standard の一般的な使用法の 1 つは、セキュリティ設計担当者がリソースとして使用することです。セキュリティアーキテクチャフレームワークとして代表的な SABSA と TOGAF には、アプリケーションセキュリティアーキテクチャをレビューする際に必要な多くの情報が不足しています。ASVS を採用することで、セキュリティ設計担当者は、データ保護パターンや入力検証戦略などの一般的な問題をより適切に管理できるようになるため、これらのフレームワークとのギャップを埋めることができます。

## 既存のセキュアコーディングチェックリストに代わるものとして

多くの組織はASVS を採用することでメリットを得られるでしょう．採用の方法としては， 3 つのレベルのいずれかを選択する方法もあれば， ASVS をフォークし，各アプリケーションリスクレベルで必要とされる要件を分野に合った形に改変するという方法もあるでしょう．この種のフォークは奨励されますが、前提条件として、追跡可能性が維持される必要があります。つまり、たとえばアプリケーションが要件 4.1 に合格している場合、フォークが進んでいっても、フォークしたコピーが ASVS 標準と同じ要件を満たすことがわかります。

## 自動の単体および統合テストのガイドとして

## ASVS は、アーキテクチャに関する要件と悪性コードに関する要件を除き、テスト可能性がきわめて高いものとなっています。特定の関連するファズデータや悪用ケースに対するテストを行う単体テストと統合テストを構築することによって、アプリケーションはすべてのビルドでほぼ自己検証可能になります。たとえば、ログインコントローラー用のテストスイートの追加テストを作成し、一般的なユーザー名、アカウントの列挙、ブルートフォース攻撃、LDAP/SQL インジェクション、および XSS に対してユーザー名パラメーターをテストすることができます。同様に、パスワードパラメーターに関するテストには、よくあるパスワード、パスワード長、ナルバイトインジェクション、パラメーターの削除、XSS、アカウントの列挙などが含まれるべきでしょう。

## セキュア開発のためのトレーニングとして

ASVS はセキュアなソフトウェアの特性を定義するためにも使用できます。"セキュアコーディング" コースの多くは、コーディングのヒントをほんの少し付け足した、単なる倫理的ハッキングコースにすぎません。これでは開発者の助けにはならないでしょう。やってはいけないトップ 10 項目を取り上げるのではなく，ASVS の事前対処の管理策により重点をおいたコースにするとよいでしょう．

# ASVSを採用しているOWASP のプロジェクト

## Security Knowledge Framework

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

開発者のセキュアコーディングトレーニング。SKF はオープンソースの Python-Flask Web アプリケーションで，OWASP Application Security Verification Standard を使用し、チームメンバーに対してセキュアコーディングのトレーニングを行うためのもの．

## OWASP Zed Attack Proxy

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

OWASP Zed Attack Proxy (ZAP) は、Web アプリケーションの脆弱性を検出するための使い勝手の良い統合ペネトレーションテストツールです。ZAP は、セキュリティの経験が様々なユーザが使うことを想定して設計されており、ペネトレーションテストを初めて行う開発者や機能テスト担当者に最適なツールとなっています。ZAP は、自動スキャナーに加え、脆弱性を手動で検出するための一連のツールを備えている。

## OWASP Cornucopia

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

OWASP Cornucopia は、カードゲームの形式で，ソフトウェア開発チームがアジャイル、従来の，あるいは形式的開発プロセスにけるセキュリティ要件の設定を助けるものです。ゲームは開発言語やプラットフォーム、テクノロジに依存しません。Cornucopia のスーツ（トランプのマーク）は、OWASP Secure Coding Practices - Quick Reference Guide (SCP) の構成に基づいて選ばれていますが、OWASP Application Security Verification Standard、OWASP Testing Guide、および David Rook の Principles of Secure Development のセクションの内容も考慮されています。

# 検証要件の詳細

V1. アーキテクチャ，設計，脅威モデリング

V2. 認証

V3. セッション管理

V4. アクセス制御

V5. 悪性入力の処理

V7. 暗号化

V8. エラー処理とログの保存

V9. データの保護

V10. 通信

V11. HTTP に関するセキュリティ設定

V13. 悪性活動の管理

V15. ビジネスロジック

V16. ファイルとリソース

V17. モバイル

V18. Web サービス (3.0で追加)

V19. 構成 (3.0で追加)

# V1: アーキテクチャ，設計，脅威モデリング

## 管理目標

検証対象のアプリケーションが次の高次の要件を満たすことを確認します。

* レベル 1: アプリケーションの構成要素を把握しており，各構成要素に存在理由がある
* レベル 2: アーキテクチャが定義されており，コードがアーキテクチャに準拠している
* レベル 3: アーキテクチャと設計が存在し、使用され、機能している

注: このセクションはバージョン 3.0 で再導入されましたが、基本的には、バージョン 1.0 の ASVS におけるアーキテクチャの管理策と同じ内容です。

## 要件

| # | 説明 | 1 | 2 | 3 | 導入 |
| --- | --- | --- | --- | --- | --- |
| 1.1 | アプリケーションのすべての構成要素を把握し，それらが必要とされている | ✓ | ✓ | ✓ | 1.0 |
| 1.2 | ライブラリ，モジュール，外部システムなど，アプリケーションに内包されてはいないがその動作に必要な構成要素をすべて把握している |  | ✓ | ✓ | 1.0 |
| 1.3 | アプリケーションの高次のアーキテクチャが定義されている |  | ✓ | ✓ | 1.0 |
| 1.4 | アプリケーションのすべての構成要素が，業務上の機能，セキュリティ上の機能の観点で定義されている |  |  | ✓ | 1.0 |
| 1.5 | アプリケーションに内包されてはいないがその動作に必要となるすべての構成要素が，業務上の機能，セキュリティ上の機能の観点で定義されている |  |  | ✓ | 1.0 |
| 1.6 | 対象となるアプリケーションの脅威モデルが作成されており，STRIDE，つまり，なりすまし (Spoofing)，改ざん (Tampering)，否認 (Repudiation)，情報漏洩 (Information Disclosure)，権限昇格 (Elevation of privilege) に関連するリスクが網羅されている |  |  | ✓ | 1.0 |
| 1.7 | すべてのセキュリティ管理策（外部のセキュリティサービスを呼び出すライブラリを含む）が集中実装されている |  | ✓ | ✓ | 3.0 |
| 1.8 | ネットワークセグメント、ファイアウォールルール、クラウドベースのセキュリティグループなどの、定義されたセキュリティ管理を通じて、構成要素が相互に切り離されている |  | ✓ | ✓ | 3.0 |
| 1.9 | アプリケーションでは、データレイヤー、コントローラーレイヤー、表示レイヤーが明確に分離されており、信頼されたシステム上でセキュリティに関する意思決定を適用できる |  | ✓ | ✓ | 3.0 |
| 1.10 | クライアント側のコードにセンシティブなビジネスロジック，秘密鍵，その他の機密情報が含まれていない |  | ✓ | ✓ | 3.0 |

## 参考情報

詳しくは以下の情報を参照してください．

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

# V2: 認証に関する検証要件

## 管理目標

認証 (Authentication) とは，対象が本物(あるいは本人) であることを立証もしくは確認する行為です．つまり対象についてなされた主張が真実であることを立証または確認する行為です．検証対象のアプリケーションが次の高次の要件を満たすことを確認します．

* 通信の送信元のデジタルIDを検証する
* 許可された者だけが認証でき，認証情報がセキュアな方法で転送される

## 要件

| # | 説明 | 1 | 2 | 3 | 導入 |
| --- | --- | --- | --- | --- | --- |
| 2.1 | 意図して公開しているものを除き，すべてのページとリソースがデフォルトで認証を必要とする（完全仲介の原則） | ✓ | ✓ | ✓ | 1.0 |
| 2.2 | すべてのパスワードフィールドについて，ユーザが入力したパスワードがそのまま表示されない | ✓ | ✓ | ✓ | 1.0 |
| 2.4 | すべての認証がサーバ側で行われる | ✓ | ✓ | ✓ | 1.0 |
| 2.6 | すべての認証機構がフェイルセキュアであり，攻撃者がログインできない | ✓ | ✓ | ✓ | 1.0 |
| 2.7 | パスワード入力フィールドでパスフレーズの使用を許可または推奨し、長いパスフレーズや複雑なパスワードの入力を拒否しない | ✓ | ✓ | ✓ | 3.0 |
| 2.8 | アカウントへのアクセス回復に使用できるすべてのアカウント ID 認証機能 (プロファイルの更新機能、パスワードの再設定、トークンの無効化や紛失、ヘルプデスク、IVR など) が、一義的な認証機構として攻撃耐性を持つ | ✓ | ✓ | ✓ | 2.0 |
| 2.9 | パスワード変更機能に、古いパスワード、新しいパスワード、パスワードの確認が含まれている | ✓ | ✓ | ✓ | 1.0 |
| 2.12 | 疑わしい認証がすべてのログに保存されている。ログには、セキュリティ調査に必要な関連するメタデータを含むリクエストが含まれていること |  | ✓ | ✓ | 2.0 |
| 2.13 | アカウントパスワードに十分な強度を持つ暗号化ルーチンを使われており、この暗号化ルーチンがブルートフォース攻撃に耐えられる |  | ✓ | ✓ | 3.0 |
| 2.16 | 認証情報は暗号化された適切なリンクを使用して転送され、ユーザに資格情報の入力を求めるすべてのページや機能は暗号化されたリンクを使用して転送されている | ✓ | ✓ | ✓ | 3.0 |
| 2.17 | パスワード再設定機能やその他のパスワード回復経路から現在のパスワードが露呈せず、新しいパスワードが平文でユーザーに送信されない | ✓ | ✓ | ✓ | 2.0 |
| 2.18 | ログイン機能、パスワード再設定機能、またはアカウントを忘れた場合の機能を使用しもユーザ情報の取得ができない | ✓ | ✓ | ✓ | 2.0 |
| 2.19 | アプリケーションのフレームワークやアプリケーションが使用するすべての構成要素でデフォルトパスワード ("admin/password" など) を使用しない | ✓ | ✓ | ✓ | 2.0 |
| 2.20 | ブルートフォース攻撃や DoS 攻撃など認証機構に対する一般的な自動攻撃を防止するために，リクエスト制限機能が存在する | ✓ | ✓ | ✓ | 3.0 |
| 2.21 | 外部サービスにアクセスするための認証証明がすべて暗号化されており、安全な場所に保管されている |  | ✓ | ✓ | 2.0 |
| 2.22 | パスワードの再設定やその他のパスワード復帰方法において、ソフトトークン、モバイルプッシュ、またはオフラインのパスワード回復機構が使用されている | ✓ | ✓ | ✓ | 3.0 |
| 2.23 | アカウントロックが、ソフトロック状態とハードロック状態に分かれていて、相互に排他的でない。つまり，ブルートフォース攻撃によりアカウントが一時的にソフトロックされた場合、ハードロック状態はリセットされない |  | ✓ | ✓ | 3.0 |
| 2.24 | 知識に基づく質問 ("秘密の質問") を実装する必要がある場合、アプリケーションを保護できる十分強固な質問である | ✓ | ✓ | ✓ | 2.0 |
| 2.25 | 過去に使用されたパスワードの再使用禁止について，何世代前までを禁止するかシステムで設定できる |  | ✓ | ✓ | 2.0 |
| 2.26 | アプリケーション固有のセンシティブな処理がアプリケーションのリスク特性に基づいて許可される前に、再認証、ステップアップ認証または適応認証、2 要素認証、あるいはトランザクション署名が要求される |  | ✓ | ✓ | 2.0 |
| 2.27 | よくあるパスワードや弱いパスフレーズの使用を禁止する対策がとられている | ✓ | ✓ | ✓ | 3.0 |
| 2.28 | 成功、失敗にかかわらず、認証要求に対する応答が同じ平均応答時間で返される |  |  | ✓ | 3.0 |
| 2.29 | シークレット、API 鍵、パスワードはソースコードやオンラインのソースコードリポジトリ中に含まれていない |  |  | ✓ | 3.0 |
| 2.30 | ユーザーが認証を行うことをアプリケーションが許可する場合、安全性が実証されている認証メカニズムを使用する | ✓ | ✓ | ✓ | 3.0 |
| 2.31 | ユーザーが認証を行うことをアプリケーションが許可する場合、ユーザは，ユーザ名とパスワードの漏洩に対策するために，2 要素認証や別の強力な認証等の機構用いることができる |  | ✓ | ✓ | 3.0 |
| 2.32 | 信頼できない第三者が管理インターフェイスにアクセスできない | ✓ | ✓ | ✓ | 3.0 |

## 参考情報

詳しくは以下の情報を参照してください．

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

# V3: セッション管理に関する検証要件

## 管理目標

One of the core components of any web-based application is the mechanism by which it controls and maintains the state for a user interacting with it. This is referred to this as Session Management and is defined as the set of all controls governing state-full interaction between a user and the web-based application.

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

* Sessions are unique to each individual and cannot be guessed or shared
* Sessions are invalidated when no longer required and timed out during periods of inactivity.

## 要件

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | **Description** | **1** | **2** | **3** | **Since** |
| **3.1** | Verify that there is no custom session manager, or that the custom session manager is resistant against all common session management attacks. | ✓ | ✓ | ✓ | 1.0 |
| **3.2** | Verify that sessions are invalidated when the user logs out. | ✓ | ✓ | ✓ | 1.0 |
| **3.3** | Verify that sessions timeout after a specified period of inactivity. | ✓ | ✓ | ✓ | 1.0 |
| **3.4** | Verify that sessions timeout after an administratively-configurable maximum time period regardless of activity (an absolute timeout). |  |  | ✓ | 1.0 |
| **3.5** | Verify that all pages that require authentication have easy and visible access to logout functionality. | ✓ | ✓ | ✓ | 1.0 |
| **3.6** | Verify that the session id is never disclosed in URLs, error messages, or logs. This includes verifying that the application does not support URL rewriting of session cookies. | ✓ | ✓ | ✓ | 1.0 |
| **3.7** | Verify that all successful authentication and re-authentication generates a new session and session id. | ✓ | ✓ | ✓ | 1.0 |
| **3.10** | Verify that only session ids generated by the application framework are recognized as active by the application. |  | ✓ | ✓ | 1.0 |
| **3.11** | Verify that session ids are sufficiently long, random and unique across the correct active session base. | ✓ | ✓ | ✓ | 1.0 |
| **3.12** | Verify that session ids stored in cookies have their path set to an appropriately restrictive value for the application, and authentication session tokens additionally set the “HttpOnly” and “secure” attributes | ✓ | ✓ | ✓ | 3.0 |
| **3.16** | Verify that the application limits the number of active concurrent sessions. | ✓ | ✓ | ✓ | 3.0 |
| **3.17** | Verify that an active session list is displayed in the account profile or similar of each user. The user should be able to terminate any active session. | ✓ | ✓ | ✓ | 3.0 |
| **3.18** | Verify the user is prompted with the option to terminate all other active sessions after a successful change password process. | ✓ | ✓ | ✓ | 3.0 |

## 参考情報

For more information, please see:

* OWASP Testing Guide 4.0: Session Management Testing <https://www.owasp.org/index.php/Testing_for_Session_Management>
* OWASP Session Management Cheat Sheet: <https://www.owasp.org/index.php/Session_Management_Cheat_Sheet>

# V4: アクセス制御に関する検査要件

## 管理目標

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

* Persons accessing resources holds valid credentials to do so.
* Users are associated with a well-defined set of roles and privileges.
* Role and permission metadata is protected from replay or tampering.

## 要件

| **#** | **Description** | **1** | **2** | **3** | **Since** |
| --- | --- | --- | --- | --- | --- |
| **4.1** | Verify that the principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege. | ✓ | ✓ | ✓ | 1.0 |
| **4.4** | Verify that access to sensitive records is protected, such that only authorized objects or data is accessible to each user (for example, protect against users tampering with a parameter to see or alter another user's account). | ✓ | ✓ | ✓ | 1.0 |
| **4.5** | Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS\_Store, .git or .svn folders. | ✓ | ✓ | ✓ | 1.0 |
| **4.8** | Verify that access controls fail securely. | ✓ | ✓ | ✓ | 1.0 |
| **4.9** | Verify that the same access control rules implied by the presentation layer are enforced on the server side. | ✓ | ✓ | ✓ | 1.0 |
| **4.10** | Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized. |  | ✓ | ✓ | 1.0 |
| **4.11** | Verify that there is a centralized mechanism (including libraries that call external authorization services) for protecting access to each type of protected resource. |  |  | ✓ | 1.0 |
| **4.12** | Verify that all access control decisions can be logged and all failed decisions are logged. |  | ✓ | ✓ | 2.0 |
| **4.13** | Verify that the application or framework uses strong random anti-CSRF tokens or has another transaction protection mechanism. | ✓ | ✓ | ✓ | 2.0 |
| **4.14** | Verify the system can protect against aggregate or continuous access of secured functions, resources, or data. For example, consider the use of a resource governor to limit the number of edits per hour or to prevent the entire database from being scraped by an individual user. |  | ✓ | ✓ | 2.0 |
| **4.15** | Verify the application has additional authorization (such as step up or adaptive authentication) for lower value systems, and / or segregation of duties for high value applications to enforce anti-fraud controls as per the risk of application and past fraud. |  | ✓ | ✓ | 3.0 |
| **4.16** | Verify that the application correctly enforces context-sensitive authorisation so as to not allow unauthorised manipulation by means of parameter tampering. | ✓ | ✓ | ✓ | 3.0 |

## 参考情報

For more information, please see:

* OWASP Testing Guide 4.0: Authorization <https://www.owasp.org/index.php/Testing_for_Authorization>
* OWASP Cheat Sheet: Access Control <https://www.owasp.org/index.php/Access_Control_Cheat_Sheet>

# V5: 悪性入力の処理に関する検証要件

## 管理目標

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

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

* All input is validated to be correct and fit for the intended purpose.
* Data from an external entity or client should never be trusted and should be handled accordingly.

## 要件

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
| **5.1** | Verify that the runtime environment is not susceptible to buffer overflows, or that security controls prevent buffer overflows. | ✓ | ✓ | ✓ | 1.0 |
| **5.3** | Verify that server side input validation failures result in request rejection and are logged. | ✓ | ✓ | ✓ | 1.0 |
| **5.5** | Verify that input validation routines are enforced on the server side. | ✓ | ✓ | ✓ | 1.0 |
| **5.6** | Verify that a single input validation control is used by the application for each type of data that is accepted. |  |  | ✓ | 1.0 |
| **5.10** | Verify that all SQL queries, HQL, OSQL, NOSQL and stored procedures, calling of stored procedures are protected by the use of prepared statements or query parameterization, and thus not susceptible to SQL injection | ✓ | ✓ | ✓ | 2.0 |
| **5.11** | Verify that the application is not susceptible to LDAP Injection, or that security controls prevent LDAP Injection. | ✓ | ✓ | ✓ | 2.0 |
| **5.12** | Verify that the application is not susceptible to OS Command Injection, or that security controls prevent OS Command Injection. | ✓ | ✓ | ✓ | 2.0 |
| **5.13** | Verify that the application is not susceptible to Remote File Inclusion (RFI) or Local File Inclusion (LFI) when content is used that is a path to a file. | ✓ | ✓ | ✓ | 3.0 |
| **5.14** | Verify that the application is not susceptible to common XML attacks, such as XPath query tampering, XML External Entity attacks, and XML injection attacks. | ✓ | ✓ | ✓ | 2.0 |
| **5.15** | Ensure that all string variables placed into HTML or other web client code is either properly contextually encoded manually, or utilize templates that automatically encode contextually to ensure the application is not susceptible to reflected, stored and DOM Cross-Site Scripting (XSS) attacks. | ✓ | ✓ | ✓ | 2.0 |
| **5.16** | If the application framework allows automatic mass parameter assignment (also called automatic variable binding) from the inbound request to a model, verify that security sensitive fields such as “*accountBalance*”, “*role*” or “*password*” are protected from malicious automatic binding. |  | ✓ | ✓ | 2.0 |
| **5.17** | Verify that the application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (GET, POST, cookies, headers, environment, etc.) |  | ✓ | ✓ | 2.0 |
| **5.18** | Verify that client side validation is used as a second line of defense, in addition to server side validation. |  | ✓ | ✓ | 3.0 |
| **5.19** | Verify that all input data is validated, not only HTML form fields but all sources of input such as REST calls, query parameters, HTTP headers, cookies, batch files, RSS feeds, etc; using positive validation (whitelisting), then lesser forms of validation such as greylisting (eliminating known bad strings), or rejecting bad inputs (blacklisting). |  | ✓ | ✓ | 3.0 |
| **5.20** | Verify that structured data is strongly typed and validated against a defined schema including allowed characters, length and pattern (e.g. credit card numbers or telephone, or validating that two related fields are reasonable, such as validating suburbs and zip or post codes match). |  | ✓ | ✓ | 3.0 |
| **5.21** | Verify that unstructured data is sanitized to enforce generic safety measures such as allowed characters and length, and characters potentially harmful in given context should be escaped (e.g. natural names with Unicode or apostrophes, such as ねこ or O'Hara) |  | ✓ | ✓ | 3.0 |
| **5.22** | Make sure untrusted HTML from WYSIWYG editors or similar are properly sanitized with an HTML sanitizer and handle it appropriately according to the input validation task and encoding task. | ✓ | ✓ | ✓ | 3.0 |
| **5.23** | For auto-escaping template technology, if UI escaping is disabled, ensure that HTML sanitization is enabled instead. |  | ✓ | ✓ | 3.0 |
| **5.24** | Verify that data transferred from one DOM context to another, uses safe JavaScript methods, such as using .innerText and .val. |  | ✓ | ✓ | 3.0 |
| **5.25** | Verify when parsing JSON in browsers, that JSON.parse is used to parse JSON on the client. Do not use eval() to parse JSON on the client. |  | ✓ | ✓ | 3.0 |
| **5.26** | Verify that authenticated data is cleared from client storage, such as the browser DOM, after the session is terminated. |  | ✓ | ✓ | 3.0 |

## 参考情報

For more information, please see:

* OWASP Testing Guide 4.0: Input Validation Testing  
  <https://www.owasp.org/index.php/Testing_for_Input_Validation>
* OWASP Cheat Sheet: Input Validation <https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet>
* OWASP Testing Guide 4.0: Testing for HTTP Parameter Pollution <https://www.owasp.org/index.php/Testing_for_HTTP_Parameter_pollution_%28OTG-INPVAL-004%29>
* OWASP LDAP Injection Cheat Sheet <https://www.owasp.org/index.php/LDAP_Injection_Prevention_Cheat_Sheet>
* OWASP Testing Guide 4.0: Client Side Testing <https://www.owasp.org/index.php/Client_Side_Testing>
* OWASP Cross Site Scripting Prevention Cheat Sheet <https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet>
* OWASP Java Encoding Project  
  <https://www.owasp.org/index.php/OWASP_Java_Encoder_Project>

For more information on auto-escaping, please see

* Reducing XSS by way of Automatic Context-Aware Escaping in Template Systems <http://googleonlinesecurity.blogspot.com/2009/03/reducing-xss-by-way-of-automatic.html>
* AngularJS Strict Contextual Escaping <https://docs.angularjs.org/api/ng/service/$sce>

# V6: 出力のエンコード / エスケープ

**This section was incorporated into V5 in Application Security Verification Standard 2.0. ASVS requirement 5.16 addresses contextual output encoding to help prevent Cross Site Scripting.**

# V7: 暗号化に関する検証要件

## 管理目標

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

* That all cryptographic modules fail in a secure manner and that errors are handled correctly.
* That a suitable random number generator is used when randomness is required.
* That access to keys is managed in a secure way.

## 要件

| **#** | **Description** | **1** | **2** | **3** | **Since** |
| --- | --- | --- | --- | --- | --- |
| **7.2** | Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable oracle padding. | ✓ | ✓ | ✓ | 1.0 |
| **7.6** | Verify that all random numbers, random file names, random GUIDs, and random strings are generated using the cryptographic module’s approved random number generator when these random values are intended to be not guessable by an attacker. |  | ✓ | ✓ | 1.0 |
| **7.7** | Verify that cryptographic algorithms used by the application have been validated against FIPS 140-2 or an equivalent standard. | ✓ | ✓ | ✓ | 1.0 |
| **7.8** | Verify that cryptographic modules operate in their approved mode according to their published security policies. |  |  | ✓ | 1.0 |
| **7.9** | Verify that there is an explicit policy for how cryptographic keys are managed (e.g., generated, distributed, revoked, and expired). Verify that this key lifecycle is properly enforced. |  | ✓ | ✓ | 1.0 |
| **7.11** | Verify that all consumers of cryptographic services do not have direct access to key material. Isolate cryptographic processes, including master secrets and consider the use of a hardware key vault (HSM). |  |  | ✓ | 3.0 |
| **7.12** | *Personally Identifiable Information* should be stored encrypted at rest and ensure that communication goes via protected channels. |  | ✓ | ✓ | 3.0 |
| **7.13** | Verify that where possible, keys and secrets are zeroed when destroyed. |  | ✓ | ✓ | 3.0 |
| **7.14** | Verify that all keys and passwords are replaceable, and are generated or replaced at installation time. |  | ✓ | ✓ | 3.0 |
| **7.15** | Verify that random numbers are created with proper entropy even when the application is under heavy load, or that the application degrades gracefully in such circumstances. |  |  | ✓ | 3.0 |

## 参考情報

For more information, please see:

* OWASP Testing Guide 4.0: Testing for weak Cryptography <https://www.owasp.org/index.php/Testing_for_weak_Cryptography>
* OWASP Cheat Sheet: Cryptographic Storage <https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet>

# V8: エラー処理とログの保存に関する検証要件

## 管理目標

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

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

* Not collecting or logging sensitive information if not specifically required.
* Ensuring all logged information is handled securely and protected as per its data classification.
* Ensuring that logs are not forever, but have an absolute lifetime that is as short as possible.

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

## 要件

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | 1 | 2 | 3 | Since |
| **8.1** | Verify that the application does not output error messages or stack traces containing sensitive data that could assist an attacker, including session id, software/framework versions and personal information | ✓ | ✓ | ✓ | 1.0 |
| **8.2** | Verify that error handling logic in security controls denies access by default. |  | ✓ | ✓ | 1.0 |
| **8.3** | Verify security logging controls provide the ability to log success and particularly failure events that are identified as security-relevant. |  | ✓ | ✓ | 1.0 |
| **8.4** | Verify that each log event includes necessary information that would allow for a detailed investigation of the timeline when an event happens. |  | ✓ | ✓ | 1.0 |
| **8.5** | Verify that all events that include untrusted data will not execute as code in the intended log viewing software. |  |  | ✓ | 1.0 |
| **8.6** | Verify that security logs are protected from unauthorized access and modification. |  | ✓ | ✓ | 1.0 |
| **8.7** | Verify that the application does not log sensitive data as defined under local privacy laws or regulations, organizational sensitive data as defined by a risk assessment, or sensitive authentication data that could assist an attacker, including user’s session identifiers, passwords, hashes, or API tokens. |  | ✓ | ✓ | 3.0 |
| **8.8** | Verify that all non-printable symbols and field separators are properly encoded in log entries, to prevent log injection. |  |  | ✓ | 2.0 |
| **8.9** | Verify that log fields from trusted and untrusted sources are distinguishable in log entries. |  |  | ✓ | 2.0 |
| **8.10** | Verify that an audit log or similar allows for non-repudiation of key transactions. |  | ✓ | ✓ | 3.0 |
| **8.11** | Verify that security logs have some form of integrity checking or controls to prevent unauthorized modification. |  |  | ✓ | 3.0 |
| **8.12** | Verify that the logs are stored on a different partition than the application is running with proper log rotation. |  |  | ✓ | 3.0 |

## 参考情報

For more information, please see:

* OWASP Testing Guide 4.0 content: Testing for Error Handling <https://www.owasp.org/index.php/Testing_for_Error_Handling>

# V9: データの保護に関する検証要件

## 管理目標

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

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

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

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

## 要件

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
| **9.1** | Verify that all forms containing sensitive information have disabled client side caching, including autocomplete features. | ✓ | ✓ | ✓ | 1.0 |
| **9.2** | Verify that the list of sensitive data processed by the application is identified, and that there is an explicit policy for how access to this data must be controlled, encrypted and enforced under relevant data protection directives. |  |  | ✓ | 1.0 |
| **9.3** | Verify that all sensitive data is sent to the server in the HTTP message body or headers (i.e., URL parameters are never used to send sensitive data). | ✓ | ✓ | ✓ | 1.0 |
| **9.4** | Verify that the application sets appropriate anti-caching headers as per the risk of the application, such as the following:  Expires: Tue, 03 Jul 2001 06:00:00 GMT  Last-Modified: {now} GMT  Cache-Control: no-store, no-cache, must-revalidate, max-age=0  Cache-Control: post-check=0, pre-check=0  Pragma: no-cache | ✓ | ✓ | ✓ | 1.0 |
| **9.5** | Verify that on the server, all cached or temporary copies of sensitive data stored are protected from unauthorized access or purged/invalidated after the authorized user accesses the sensitive data. |  | ✓ | ✓ | 1.0 |
| **9.6** | Verify that there is a method to remove each type of sensitive data from the application at the end of the required retention policy. |  |  | ✓ | 1.0 |
| **9.7** | Verify the application minimizes the number of parameters in a request, such as hidden fields, Ajax variables, cookies and header values. |  | ✓ | ✓ | 2.0 |
| **9.8** | Verify the application has the ability to detect and alert on abnormal numbers of requests for data harvesting for an example screen scraping. |  |  | ✓ | 2.0 |
| **9.9** | Verify that data stored in client side storage - such as HTML5 local storage, session storage, IndexedDB, regular cookies or Flash cookies - does not contain sensitive or PII). | ✓ | ✓ | ✓ | 3.0 |
| **9.10** | Verify accessing sensitive data is logged, if the data is collected under relevant data protection directives or where logging of accesses is required. |  | ✓ | ✓ | 3.0 |
| **9.11** | Verify that sensitive data is rapidly sanitized from memory as soon as it is no longer needed and handled in accordance to functions and techniques supported by the framework/library/operating system. |  | ✓ | ✓ | 3.0 |

## 参考情報

For more information, please see:

* User Privacy Protection Cheat Sheet: <https://www.owasp.org/index.php/User_Privacy_Protection_Cheat_Sheet>

# V10: 通信のセキュリティに関する検証要件

## 管理目標

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

* That TLS is used where sensitive data is transmitted
* That strong algorithms and ciphers are used at all times.

## 要件

| **#** | **Description** | **1** | **2** | **3** | **Since** |
| --- | --- | --- | --- | --- | --- |
| **10.1** | Verify that a path can be built from a trusted CA to each Transport Layer Security (TLS) server certificate, and that each server certificate is valid. | ✓ | ✓ | ✓ | 1.0 |
| **10.3** | Verify that TLS is used for all connections (including both external and backend connections) that are authenticated or that involve sensitive data or functions, and does not fall back to insecure or unencrypted protocols. Ensure the strongest alternative is the preferred algorithm. | ✓ | ✓ | ✓ | 3.0 |
| **10.4** | Verify that backend TLS connection failures are logged. |  |  | ✓ | 1.0 |
| **10.5** | Verify that certificate paths are built and verified for all client certificates using configured trust anchors and revocation information. |  |  | ✓ | 1.0 |
| **10.6** | Verify that all connections to external systems that involve sensitive information or functions are authenticated. |  | ✓ | ✓ | 1.0 |
| **10.8** | Verify that there is a single standard TLS implementation that is used by the application that is configured to operate in an approved mode of operation. |  |  | ✓ | 1.0 |
| **10.10** | Verify that TLS certificate public key pinning is implemented with production and backup public keys. For more information, please see the references below. |  |  | ✓ | 3.0 |
| **10.11** | Verify that HTTP Strict Transport Security headers are included on all requests and for all subdomains, such as Strict-Transport-Security: max-age=15724800; includeSubdomains | ✓ | ✓ | ✓ | 3.0 |
| **10.12** | Verify that production website URL has been submitted to preloaded list of Strict Transport Security domains maintained by web browser vendors. Please see the references below. |  |  | ✓ | 3.0 |
| **10.13** | Ensure forward secrecy ciphers are in use to mitigate passive attackers recording traffic. | ✓ | ✓ | ✓ | 3.0 |
| **V10.14** | Verify that proper certification revocation, such as Online Certificate Status Protocol (OSCP) Stapling, is enabled and configured. | ✓ | ✓ | ✓ | 3.0 |
| **V10.15** | Verify that only strong algorithms, ciphers, and protocols are used, through all the certificate hierarchy, including root and intermediary certificates of your selected certifying authority. | ✓ | ✓ | ✓ | 3.0 |
| **V10.16** | Verify that the TLS settings are in line with current leading practice, particularly as common configurations, ciphers, and algorithms become insecure. | ✓ | ✓ | ✓ | 3.0 |

## 参考情報

For more information, please see:

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

# V11: HTTP のセキュリティ設定に関する検証要件

## 管理目標

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

* The application server is suitably hardened from a default configuration
* HTTP responses contain a safe character set in the content type header.

## 要件

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Description** | **1** | **2** | **3** | **Since** |
| **11.1** | Verify that the application accepts only a defined set of required HTTP request methods, such as GET and POST are accepted, and unused methods (e.g. TRACE, PUT, and DELETE) are explicitly blocked. | ✓ | ✓ | ✓ | 1.0 |
| **11.2** | Verify that every HTTP response contains a content type header specifying a safe character set (e.g., UTF-8, ISO 8859-1). | ✓ | ✓ | ✓ | 1.0 |
| **11.3** | Verify that HTTP headers added by a trusted proxy or SSO devices, such as a bearer token, are authenticated by the application. |  | ✓ | ✓ | 2.0 |
| **11.4** | Verify that the Content Security Policy V2 (CSP) is in use for sites where content should not be viewed in a 3rd-party X-Frame. |  | ✓ | ✓ | 2.0 |
| **11.5** | Verify that the HTTP headers or any part of the HTTP response do not expose detailed version information of system components. | ✓ | ✓ | ✓ | 2.0 |
| **11.6** | Verify that all API responses contain X-Content-Type-Options: nosniff and Content-Disposition: attachment; filename="api.json" (or other appropriate filename for the content type). | ✓ | ✓ | ✓ | 3.0 |
| **11.7** | Verify that the Content Security Policy V2 (CSP) is in use in a way that either disables inline JavaScript or provides an integrity check on inline JavaScript with CSP noncing or hashing. | ✓ | ✓ | ✓ | 3.0 |
| **11.8** | Verify that the X-XSS-Protection: 1; mode=block header is in place. | ✓ | ✓ | ✓ | 3.0 |

## 参考情報

For more information, please see:

* OWASP Testing Guide 4.0: Testing for HTTP Verb Tampering <https://www.owasp.org/index.php/Testing_for_HTTP_Verb_Tampering_%28OTG-INPVAL-003%29>
* Adding Content-Disposition to API responses helps prevent many attacks based on misunderstanding on the MIME type between client and server, and the "filename" option specifically helps prevent Reflected File Download attacks.   
  <https://www.blackhat.com/docs/eu-14/materials/eu-14-Hafif-Reflected-File-Download-A-New-Web-Attack-Vector.pdf>

# V12: セキュリティ設計に関する検証要件

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

# V13: 悪性活動の管理に関する検証要件

## 管理目標

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

* Malicious activity is handled securely and properly as to not affect the rest of the application.
* Do not have time bombs or other time based attacks built into them
* do not “phone home” to malicious or unauthorized destinations
* Applications do not have back doors, Easter eggs, salami attacks, or logic flaws that can be controlled by an attacker

Malicious code is extremely rare, and is difficult to detect. Manual line by line code review can assist looking for logic bombs, but even the most experienced code reviewer will struggle to find malicious code even if they know it exists.

## 要件

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
| **13.1** | Verify all malicious activity is adequately sandboxed, containerized or isolated to delay and deter attackers from attacking other applications. |  |  | ✓ | 2.0 |
| **13.2** | Verify that a code review looks for malicious code, back doors, Easter eggs, and logic flaws. |  |  | ✓ | 3.0 |

## 参考情報

For more information, please see:

* <http://www.dwheeler.com/essays/apple-goto-fail.html>

# V14: 内部セキュリティに関する検証要件

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

# V15: ビジネスロジックに関する検証要件

## 管理目標

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

* The business logic flow is sequential and in order
* Business logic includes limits to detect and prevent automated attacks, such as continuous small funds transfers, or adding a million friends one at a time, and so on.
* High value business logic flows have considered abuse cases and malicious actors, and have protections against spoofing, tampering, repudiation, information disclosure, and elevation of privilege attacks.

## 要件

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
| **V15.1** | Verify the application will only process business logic flows in sequential step order, with all steps being processed in realistic human time, and not process out of order, skipped steps, process steps from another user, or too quickly submitted transactions. |  | ✓ | ✓ | 2.0 |
| **V15.2** | Verify the application has business limits and correctly enforces on a per user basis, with configurable alerting and automated reactions to automated or unusual attack. |  | ✓ | ✓ | 2.0 |

## 参考情報

For more information, please see:

* OWASP Testing Guide 4.0: Business Logic Testing <https://www.owasp.org/index.php/Testing_for_business_logic>
* OWASP Cheat Sheet: <https://www.owasp.org/index.php/Business_Logic_Security_Cheat_Sheet>

# V16: ファイルとリソースに関する検証要件

## 管理目標

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

* Untrusted file data should be handled accordingly and in a secure manner
* Obtained from untrusted sources are stored outside the webroot and limited permissions.

## 要件

| **#** | **Description** | **1** | **2** | **3** | **Since** |
| --- | --- | --- | --- | --- | --- |
| **16.1** | Verify that URL redirects and forwards only allow whitelisted destinations, or show a warning when redirecting to potentially untrusted content. | ✓ | ✓ | ✓ | 2.0 |
| **16.2** | Verify that untrusted file data submitted to the application is not used directly with file I/O commands, particularly to protect against path traversal, local file include, file mime type, and OS command injection vulnerabilities. | ✓ | ✓ | ✓ | 2.0 |
| **16.3** | Verify that files obtained from untrusted sources are validated to be of expected type and scanned by antivirus scanners to prevent upload of known malicious content. | ✓ | ✓ | ✓ | 2.0 |
| **16.4** | Verify that untrusted data is not used within inclusion, class loader, or reflection capabilities to prevent remote/local file inclusion vulnerabilities. | ✓ | ✓ | ✓ | 2.0 |
| **16.5** | Verify that untrusted data is not used within cross-domain resource sharing (CORS) to protect against arbitrary remote content. | ✓ | ✓ | ✓ | 2.0 |
| **16.6** | Verify that files obtained from untrusted sources are stored outside the webroot, with limited permissions, preferably with strong validation. |  | ✓ | ✓ | 3.0 |
| **16.7** | Verify that the web or application server is configured by default to deny access to remote resources or systems outside the web or application server. |  | ✓ | ✓ | 2.0 |
| **16.8** | Verify the application code does not execute uploaded data obtained from untrusted sources. | ✓ | ✓ | ✓ | 3.0 |
| **16.9** | Do not use Flash, Active-X, Silverlight, NACL, client-side Java or other client side technologies not supported natively via W3C browser standards. | ✓ | ✓ | ✓ | 2.0 |

## 参考情報

For more information, please see:

* File Extension Handling for Sensitive Information: <https://www.owasp.org/index.php/Unrestricted_File_Upload>

# V17: モバイルに関する検証要件

## 管理目標

This section contains controls that are mobile application specific. These controls have been de-duplicated from 2.0, so must be taken in conjunction with all other sections of the relevant ASVS Verification Level.

Mobile applications should:

* Should have the same level of security controls within the mobile client as found in the server, by enforcing security controls in a trusted environment
* Sensitive information assets stored on the device should be done so in a secure manner
* All sensitive data transmitted from the device should be done so with transport layer security in mind.

## 要件

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
| **17.1** | Verify that ID values stored on the device and retrievable by other applications, such as the UDID or IMEI number are not used as authentication tokens. | ✓ | ✓ | ✓ | 2.0 |
| **17.2** | Verify that the mobile app does not store sensitive data onto potentially unencrypted shared resources on the device (e.g. SD card or shared folders). | ✓ | ✓ | ✓ | 2.0 |
| **17.3** | Verify that sensitive data is not stored unprotected on the device, even in system protected areas such as key chains. | ✓ | ✓ | ✓ | 2.0 |
| **17.4** | Verify that secret keys, API tokens, or passwords are dynamically generated in mobile applications. |  | ✓ | ✓ | 2.0 |
| **17.5** | Verify that the mobile app prevents leaking of sensitive information (for example, screenshots are saved of the current application as the application is backgrounded or writing sensitive information in console) . |  | ✓ | ✓ | 2.0 |
| **17.6** | Verify that the application is requesting minimal permissions for required functionality and resources. |  | ✓ | ✓ | 2.0 |
| **17.7** | Verify that the application sensitive code is laid out unpredictably in memory (For example ASLR). | ✓ | ✓ | ✓ | 2.0 |
| **17.8** | Verify that there are anti-debugging techniques present that are sufficient enough to deter or delay likely attackers from injecting debuggers into the mobile app (For example GDB). |  |  | ✓ | 2.0 |
| **17.9** | Verify that the app does not export sensitive activities, intents, content providers etc., for other mobile apps on the same device to exploit. | ✓ | ✓ | ✓ | 2.0 |
| **17.10** | Verify that mutable structures have been used for sensitive strings such as account numbers and are overwritten when not used. (Mitigate damage from memory analysis attacks). |  |  | ✓ | 2.0 |
| **17.11** | Verify that the app’s exposed activities, intents, content providers etc. validate all inputs. | ✓ | ✓ | ✓ | 2.0 |

## 参考情報

For more information, please see:

* OWASP Mobile Security Project: <https://www.owasp.org/index.php/OWASP_Mobile_Security_Project>
* iOS Developer Cheat Sheet: <https://www.owasp.org/index.php/IOS_Developer_Cheat_Sheet>

# V18: Web サービスに関する検証要件

## 管理目標

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

* Adequate authentication, session management and authorization of all web services
* Input validation of all parameters that transit from a lower to higher trust level
* Basic interoperability of SOAP web services layer to promote API use

## 要件

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
| **18.1** | Verify that the same encoding style is used between the client and the server. | ✓ | ✓ | ✓ | 3.0 |
| **18.2** | Verify that access to administration and management functions within the Web Service Application is limited to web service administrators. | ✓ | ✓ | ✓ | 3.0 |
| **18.3** | Verify that XML or JSON schema is in place and verified before accepting input. | ✓ | ✓ | ✓ | 3.0 |
| **18.4** | Verify that all input is limited to an appropriate size limit. | ✓ | ✓ | ✓ | 3.0 |
| **18.5** | Verify that SOAP based web services are compliant with Web Services-Interoperability (WS-I) Basic Profile at minimum. | ✓ | ✓ | ✓ | 3.0 |
| **18.6** | Verify the use of session-based authentication and authorization. Please refer to sections 2, 3 and 4 for further guidance. Avoid the use of static "API keys" and similar. | ✓ | ✓ | ✓ | 3.0 |
| **18.7** | Verify that the REST service is protected from Cross-Site Request Forgery. | ✓ | ✓ | ✓ | 3.0 |
| **18.8** | Verify the REST service explicitly check the incoming Content-Type to be the expected one, such as application/xml or application/json. |  | ✓ | ✓ | 3.0 |
| **18.9** | Verify that the message payload is signed to ensure reliable transport between client and service. |  | ✓ | ✓ | 3.0 |
| **18.10** | Verify that alternative and less secure access paths do not exist. |  | ✓ | ✓ | 3.0 |

## 参考情報

For more information, please see:

* OWASP Testing Guide 4.0: Configuration and Deployment Management Testing <https://www.owasp.org/index.php/Testing_for_configuration_management>

# V19. 構成

## 管理目標

Ensure that a verified application has:

* Up to date libraries and platform(s).
* A secure by default configuration.
* Sufficient hardening that user initiated changes to default configuration do not unnecessarily expose or create security weaknesses or flaws to underlying systems.

## 要件

| # | Description | 1 | 2 | 3 | Since |
| --- | --- | --- | --- | --- | --- |
| **19.1** | All components should be up to date with proper security configuration(s) and version(s). This should include removal of unneeded configurations and folders such as sample applications, platform documentation, and default or example users. | ✓ | ✓ | ✓ | 3.0 |
| **19.2** | Communications between components, such as between the application server and the database server, should be encrypted, particularly when the components are in different containers or on different systems. |  | ✓ | ✓ | 3.0 |
| **19.3** | Communications between components, such as between the application server and the database server should be authenticated using an account with the least necessary privileges. |  | ✓ | ✓ | 3.0 |
| **19.4** | Verify application deployments are adequately sandboxed, containerized or isolated to delay and deter attackers from attacking other applications. |  | ✓ | ✓ | 3.0 |
| **19.5** | Verify that the application build and deployment processes are performed in a secure fashion. |  | ✓ | ✓ | 3.0 |
| **19.6** | Verify that authorised administrators have the capability to verify the integrity of all security-relevant configurations to ensure that they have not been tampered with. |  |  | ✓ | 3.0 |
| **19.7** | Verify that all application components are signed. |  |  | ✓ | 3.0 |
| **19.8** | Verify that third party components come from trusted repositories. |  |  | ✓ | 3.0 |
| **19.9** | Ensure that build processes for system level languages have all security flags enabled, such as ASLR, DEP, and security checks. |  |  | ✓ | 3.0 |

## 参考情報

For more information, please see:

* OWASP Testing Guide 4.0: Configuration and Deployment Management Testing <https://www.owasp.org/index.php/Testing_for_configuration_management>

# 付録 A: 廃止された要件

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Original #** | **Description** | **Status** | **Removed** | **Reason** |
| **2.3** | Verify that if a maximum number of authentication attempts is exceeded, the account is locked for a period of time long enough to deter brute force attacks. | Deprecated | 2.0 | A more complex requirement replaced it (v2.20) |
| **2.5** | Verify that all authentication controls (including libraries that call external authentication services) have a centralized implementation. | Merged | 3.0 | Genericized to include all security controls and moved to 1.10 |
| **2.10** | Verify that re-authentication is required before any application- specific sensitive operations are permitted. | Deprecated | 2.0 | Re-authentication is so rarely observed that we decided to remove the control |
| **2.11** | Verify that after an administratively- configurable period of time, authentication credentials expire. | Deprecated | 2.0 | Absolute timeouts and credential expiry removed as not being an effective control. |
| **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). | Updated | 2.0 | Became V2.21 |
| **2.15** | Verify that all code implementing or using authentication controls is not affected by any malicious code. | Moved | 2.0 | Moved to V13 - Malicious Code |
| **3.8** | Verify that the session id is changed upon re-authentication | Updated | 3.0 | Rolled into 3.7 |
| **3.9** | Verify that the session id is changed or cleared on logout | Updated | 3.0 | Rolled into 3.7 |
| **3.13** | Verify that all code implementing or using session management controls is not affected by any malicious code | Moved | 2.0 | Moved to V13 - Malicious code |
| **3.14** | Verify that authenticated session tokens using cookies are protected by the use of "HttpOnly". | Updated | 3.0 | Moved into 3.13 |
| **3.15** | Verify that authenticated session tokens using cookies are protected with the "secure" attribute. | Updated | 3.0 | Moved into 3.13 |
| **4.2** | Verify that users can only access secured URLs for which they possess specific authorization. | Updated | 3.0 | Rolled into 4.1 |
| **4.3** | Verify that users can only access secured data files for which they possess specific authorization. | Updated | 3.0 | Rolled into 4.1 |
| **4.13** | Verify that limitations on input and access imposed by the business on the application (such as daily transaction limits or sequencing of tasks) cannot be bypassed. | Moved | 3.0 | Moved to V15 Business Logic |
| **4.15** | Verify that all code implementing or using access controls is not affected by any malicious code. | Moved | 2.0 | Moved to V13 Malicious Controls |
| **5.2** | Verify that a positive validation pattern is defined and applied to all input | Deprecated | 2.0 | Removed as too difficult to implement particularly for free form text inputs |
| **5.4** | Verify that a character set, such as UTF-8, is specified for all sources of input | Deprecated | 3.0 | Removed as too difficult to implement in most languages |
| **5.7** | Verify that all input validation failures are logged. | Deprecated | 3.0 | Removed as would create too many useless logs that would be ignored |
| **5.8** | Verify that all input data is canonicalized for all downstream decoders or interpreters prior to validation. | Deprecated | 3.0 | Removed as Type 1 JSP technology specific and not an issue for most modern frameworks |
| **5.9** | Verify that all input validation controls are not affected by any malicious code | Moved | 2.0 | Moved to V13 Malicious controls |
| **5.14** | Verify that the runtime environment is not susceptible to XML Injections or that security controls prevents XML Injections | Merged | 3.0 | Merged with V5.13 |
| **5.15** | -- EMPTY REQUIREMENT -- | Deleted | 3.0 | This requirement never existed |
| **5.19** | Verify that for each type of output encoding/escaping performed by the application, there is a single security control for that type of output for the intended destination | Merged | 3.0 | Genericized to include all security controls and moved to 1.10 |
| **7.1** | Verify that all cryptographic functions used to protect secrets from the application user are implemented server side | Deprecated | 3.0 | Many modern responsive and mobile apps include this by design |
| **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). | Moved | 3.0 | Moved to V2.29 |
| **7.4** | Verify that password hashes are salted when they are created | Moved | 2.0 | Moved to V2.13 |
| **7.5** | Verify that cryptographic module failures are logged | Deprecated | 2.0 | Creating unnecessary logs that are never reviewed is counterproductive |
| **7.10** | Verify that all code supporting or using a cryptographic module is not affected by any malicious code | Moved | 2.0 | Moved to V13 |
| **8.2** | Verify that all error handling is performed on trusted devices |  | 3.0 | Deprecated |
| **8.3** | Verify that all logging controls are implemented on the server. | Moved | 3.0 | Became a more generic architectural control V1.13 |
| **8.9** | Verify that there is a single application-level logging implementation that is used by the software. | Moved | 3.0 | Became a more generic architectural control V1.13 |
| **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. | Deprecated | 3.0 | Removed as not required for secure software |
| **8.12** | Verify that all code implementing or using error handling and logging controls is not affected by any ￼￼￼malicious code. | Moved | 2.0 | Moved to V13 Malicious Controls |
| **8.15** | Verify that logging is performed before executing the transaction. If logging was unsuccessful (e.g. disk full, insufficient permissions) the application fails safe. This is for when integrity and non-repudiation are a must. | Deprecated | 3.0 | Removed as too detailed a control that would only be applicable to small percentage of all apps |
| **10.2** | Verify that failed TLS connections do not fall back to an insecure HTTP connection | Merged | 3.0 | Merged with 10.3 |
| **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 |  |  |  |
| **10.9** | Verify that specific character encodings are defined for all connections (e.g., UTF-8). |  |  |  |
| **V11.1** | Deprecated |  |  |  |
| **V11.4** | Deprecated |  |  |  |
| **V11.5** | Deprecated |  |  |  |
| **V11.6** | Deprecated |  |  |  |
| **V11.7** | Deprecated |  |  |  |
| **V11.8** | Deprecated |  |  |  |
| **V11.4** | Deprecated |  |  |  |
| **V13.1** | Deprecated |  |  |  |
| **V13.2** | Deprecated |  |  |  |
| **V13.3** | Deprecated |  |  |  |
| **V13.4** | Deprecated |  |  |  |
| **V13.5** | Deprecated |  |  |  |
| **V13.6** | Deprecated |  |  |  |
| **V13.7** | Deprecated |  |  |  |
| **V13.8** | Deprecated |  |  |  |
| **V13.9** | Deprecated |  |  |  |
| **15.1-15.7 15.9** | Business Logic Section. | Merged | 3.0 | Most of section 15 has been merged into 15.8 and 15.10. |
| **15.11** | Verify that the application covers off risks associated with Spoofing, Tampering, Repudiation, Information Disclosure, and Elevation of privilege (STRIDE). | Duplicate | 3.0 | Duplicated requirement. Captured by V1.6 |
| **16.4** | 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. | Moved | 3.0 | Moved to V16.2 |
| **17.1** | Verify that the client validates SSL certificates | Deprecated | 3.0 | Duplicated requirement. General requirement already captured by V10. |
| **V17.7** | Deprecated |  |  |  |
| **V17.8** | Deprecated |  |  |  |
| **V17.10** | Deprecated |  |  |  |
| **V17.11** | Deprecated |  |  |  |
| **V17.12** | Deprecated |  |  |  |
| **V17.13** | Deprecated |  |  |  |
| **V17.14** | Deprecated |  |  |  |
| **V17.15** | Deprecated |  |  |  |
| **V17.16** | Deprecated |  |  |  |
| **V17.17** | Deprecated |  |  |  |
| **V17.18** | Deprecated |  |  |  |
| **V17.19** | Deprecated |  |  |  |
| **V17.20** | Deprecated |  |  |  |
| **V17.22** | Deprecated |  |  |  |
| **V17.23** | Deprecated |  |  |  |
| **V17.24** | Deprecated |  |  |  |

# 付録 B: 用語集

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

# 付録 C: 参考情報

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

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

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

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

# 付録 D: 他の標準との対応

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

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

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