OWASP Top 10:2025

OWASP Top 10:2025 リリースへようこそ。

OWASP Top 10 は、開発者と Web アプリケーション セキュリティのための標準的な意識向上のための文書です。本書では、Web アプリケーションにとって最も重大なセキュリティ リスクに関する幅広いコンセンサスを示しています。

本リリースについて

これはOWASP Top 10 の **2025** 年版です。このバージョンには、最新のデータとセキュリティ トレンドに基づいた更新が含まれています。

メイン プロジェクトのページ

メイン プロジェクトのページには、このプロジェクトの古いバージョンとメタデータに関する情報が記載されています。

開始するには

2025 バージョンにおける新規内容について知るには、はじめにから始めます。

ナビゲーション

Top 10:2025 一覧

注: 翻訳版は利用可能になり次第追加されます。

はじめに

OWASP Top 10 の 第 8 版へようこそ！

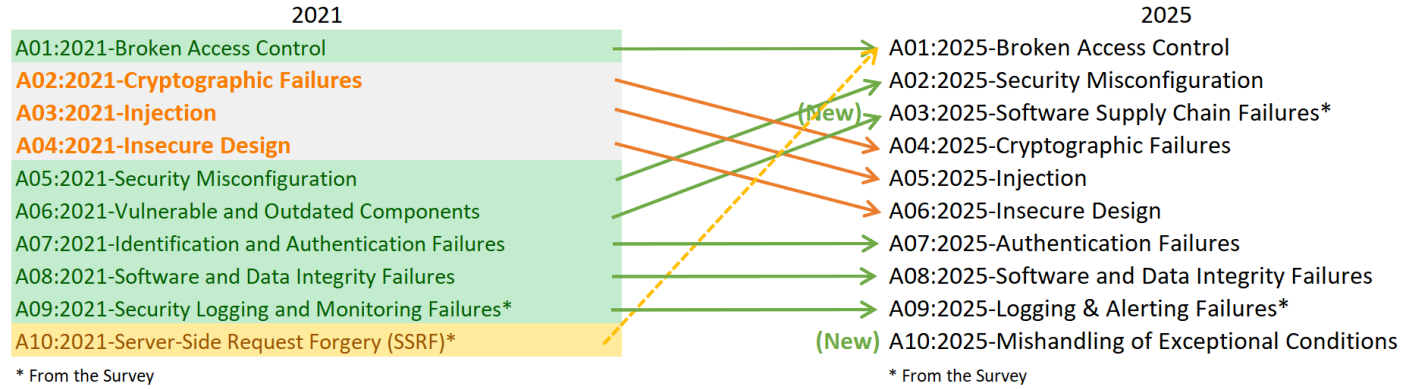
調査にデータと視点をご提供いただいた皆様に心より感謝申し上げます。皆様のご協力がなければ、今回の調査は実現しませんでした。**ありがとうございます！**

OWASP Top 10:2025 の紹介

- A01:2025 - アクセス制御の不備
- A02:2025 - セキュリティ設定のミス
- A03:2025 - ソフトウェア サプライチェーンの失敗
- A04:2025 - 暗号化の失敗
- A05:2025 - インジェクション
- A06:2025 - セキュリティが確保されていない設計
- A07:2025 - 認証の失敗
- A08:2025 - ソフトウェアまたはデータの整合性の不具合
- A09:2025 - ログ記録およびアラートの失敗
- A10:2025 - 例外的な状況への不適切な対処

Top 10 2025 年版における変更点

2025 年版の Top 10 には、2 つの新しいカテゴリが追加され、1 つのカテゴリが統合されました。私たちは、できる限り結果（訳注：原文では "symptons"）ではなく根本原因に焦点を当てるよう努めてきました。ソフトウェア エンジニアリングとソフトウェア セキュリティの複雑さを考えると、ある程度の重複なしに 10 個のカテゴリを作成することは事実上不可能です。



- **A01:2025 - アクセス制御の不備** は、最も深刻なアプリケーション セキュリティ リスクとして引き続き 1 位にランクされています。提供されたデータによると、テスト対象のアプリケーションのうち平均 3.73% が、このカテゴリに含まれる 40 個の共通脆弱性識別子（CWE）のうち 1 つ以上を抱えていました。上の図の破線で示されているように、サーバー サイド リクエスト フォージェリ（SSRF）はこのカテゴリに統合されています。
- **A02:2025 - セキュリティ設定のミス** は、2021 年版では 5 位でしたが 2025 年版では 2 位に昇格しました。今回のデータでは、設定ミスがより多く見られます。テスト対象となったアプリケーションの 3.00% に、このカテゴリの 16 個の CWE のうち 1 つ以上が含まれていました。これは驚くべきことではありません。ソフトウェア エンジニアリングにおいて、アプリケーションの動作が設定に基づいて決定される部分が増え続けているためです。
- **A03:2025 - ソフトウェア サプライチェーンの失敗** は、A06:2021 - 脆弱で古くなったコンポーネント の拡張であり、ソフトウェア依存関係、ビルド システム、および配布インフラストラクチャといったソフトウェア エコシステム全体、あるいはその複数の要素にわたって発生する広範な侵害を網羅するものです。このカテゴリは、コミュニティ調査で最も懸念される事項として圧倒的に多くの票を集めました。このカテゴリには 5 個の CWE が含まれており、収集データにおける出現頻度は低いものの、これはテストにおける課題によるものだと考えており、今後この分野のテストが改善されることを期待しています。このカテゴリはデータにおける出現件数は最も少ないものの、CVE における平均的な悪用可能性スコアと影響度スコアは最も高くなっています。
- **A04:2025 - 暗号化の失敗** は、ランキングで 2 位から 4 位へと 2 ランク順位を下げました。提供されたデータによると、平均してアプリケーションの 3.80% に、このカテゴリに含まれる 32 個の CWE のうち 1 つ以上が存在しています。このカテゴリの脆弱性は、機密データの漏洩やシステム侵害につながることが多いです。
- **A05:2025 - インジェクション** は、ランキングで 3 位から 5 位に 2 つ順位を下げましたが、暗号化の失敗やセキュリティが確保されていない設計といった項目に対する相対的な位置は維持しています。インジェクションは最も多くテストされているカテゴリの 1 つであり、このカテゴリに含まれる 38 個の CWE に関連付けられた CVE の数も最多です。インジェクションには、クロスサイト スクリプティング（高発生頻度／低影響度）から SQL インジェクション（低発生頻度／高影響度）といった幅広い問題が含まれます。
- **A06:2025 - セキュリティが確保されていない設計** は、セキュリティ設定のミスとソフトウェア サプライチェーンの失敗が上位に躍り出たため、ランキングで 4 位から 6 位へと 2 つ順位を落としました。このカテゴリは 2021 年に導入されましたが、脅威モデリングとセキュリティを確保する設計への重点化に関して、業界における顕著な改善が見られました。
- **A07:2025 - 認証の失敗** は、名称がわずかに変更されたものの（以前は「識別と認証の失敗 」でした）、このカテゴリに含まれる 36 個の CWE をより正確に反映するために若干の名称変更（以前は「識別および認証の失敗」）を経た上で、7 位を維持しています。このカテゴリは依然として重要ですが、標準化されたフレームワークが認証に広く使用されるようになったことで、認証の失敗の発生件数に良い影響が出ているようです。
- **A08:2025 - ソフトウェアまたはデータの整合性の不具合** は、引き続き 8 位を維持しています。このカテゴリは、ソフトウェア サプライチェーンの脆弱性よりも低いレベルにおける信頼境界の維持とソフトウェア、コード、データのアーティファクトの整合性の検証の失敗に焦点を当てています。
- **A09:2025 - ログ記録およびアラートの失敗** は、引き続き 9 位を維持しています。このカテゴリは名称が若干変更されました（以前は「セキュリティ ログ記録と監視の失敗」（https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/））。これは、関連するログ イベントに対して適切な措置を講じるために必要なアラート機能の重要性を強調するためです。優れたログ記録があってもアラート機能がなければ、セキュリティ インシデントの特定においてほとんど価値がありません。このカテゴリは常にデータ上では過小評価されがちですが、コミュニティ調査の参加者によって再びリストにランクインしました。
- **A10:2025 - 例外的な状況への不適切な対処** は、2025 年に新設されたカテゴリです。このカテゴリには、不適切なエラー処理、論理エラー、セキュリティ上の欠陥（フェイル オープン）、およびシステムが遭遇する可能性のある異常な状況に起因するその他の関連シナリオに焦点を当てた 24 個の CWE が含まれています。

方法論

今回の Top 10 ランキングは、データに基づいたものですが、データのみに盲目的に依存しているわけではありません。提供されたデータに基づいて 12 個のカテゴリをランク付けし、さらにコミュニティ調査からの回答に基づいて 2 つのカテゴリを上位に昇格させたり強調できるようにしたりしました。これには根本的な理由があります。提供されたデータを検証することは、本質的に過去を振り返ることだからです。アプリケーション セキュリティの研究者は、新しい脆弱性を特定し、新しいテスト方法を開発するために時間を費やしています。これらのテストをツールやプロセスに統合するには、数週間から数年かかる場合があります。弱点を大規模に確実にテストできるようになるまでには、数年が経過している可能性もあります。また、確実にテストすることができず、データにも反映されない重要なリスクも存在します。こうした点を考慮し、バランスを取るために、コミュニティ調査を実施し、最前線で活躍するアプリケーション セキュリティおよび開発の実務者に、テスト データで過小評価されている可能性のある重要なリスクは何かと尋ねています。

カテゴリの構成方法

OWASP Top 10 の以前の版から、いくつかのカテゴリが変更されました。以下に、カテゴリ変更の概要を説明します。

今回の版では、2021 年版のように CWE に制限を設けることなく、データ収集を行いました。具体的には、特定の年（2021 年以降）にテストされたアプリケーションの数と、テストで少なくとも 1 つの CWE が検出されたアプリケーションの数を収集しました。この形式により、各 CWE がアプリケーション全体の中でどの程度普及しているかを追跡できます。ここでは頻度は考慮していません。他の状況では必要となる場合もありますが、頻度を考慮するとアプリケーション全体における実際の普及率が隠れてしまうためです。アプリケーションに CWE が 4 つ存在する場合でも 4000 個存在する場合でも、Top 10 の計算には影響しません。特に、手動テストでは、アプリケーション内で同じ脆弱性が何度繰り返されていても一度しか報告されない傾向があるのに対し、自動テスト フレームワークは脆弱性のすべての事例を個別に報告するためです。分析対象となる CWE の数は、2017年 の約 30 個から 2021 年には約 400 個、そして今回の版では 589 個へと大幅に増加しました。今後、補足として追加のデータ分析を実施する予定です。CWE 数の大幅な増加に伴い、カテゴリの構造にも変更が必要となりました。

OWASP は、数ヶ月かけて CWE をグループ分けし、分類作業を行いましたが、さらに数ヶ月続けることも可能でした。しかし、どこかで区切りをつける必要がありました。CWE には根本原因型と結果（訳注：原文では "symptons"）型の両方があり、根本原因型は「暗号化の失敗」や「設定ミス」など、結果型は「機密データの漏洩」や「サービス拒否攻撃」などです。私たちは、可能な限り根本原因に焦点を当てることにしました。その理由は、これが識別と修復のためのガイダンスを提供する上でより論理的であるためです。結果ではなく根本原因に焦点を当てるという考え方は新しいものではありません。OWASP Top 10 も結果と根本原因が混在しています。CWE も同様に結果と根本原因が混在していますが、私たちはそれをより明確に意識して分類しています。今回の改訂版では、1 つのカテゴリあたり平均 25 個の CWE が含まれており、A02:2025-ソフトウェア サプライチェーンの失敗と A09:2025-ログ記録とアラートの失敗では 5 個、A01:2025-アクセス制御の不備では 40 個となっています。私たちは、1 つのカテゴリに含まれる CWE の数を 40 個に制限することにしました。この更新されたカテゴリ構造は、企業が使用している言語やフレームワークに適したCWEに焦点を当てることができるため、トレーニング面でも追加のメリットがあります。

なぜ MITREの「最も危険なソフトウェア脆弱性 Top 25」のように、10 個の CWE を Top 10としてリストアップしないのか、という質問をいただきました。私たちが複数の CWE をカテゴリに分類して使用している主な理由は 2 つあります。まず、すべての CWE がすべてのプログラミング言語やフレームワークに存在するわけではないためです。これは、ツールやトレーニング/啓発プログラムにおいて問題となります。Top 10 に含まれる脆弱性の中には、適用できないものがある可能性があるからです。2 つ目の理由は、一般的な脆弱性に対して複数の CWE が存在するということです。例えば、一般的なインジェクション、コマンド インジェクション、クロスサイト スクリプティング、ハード コードされたパスワード、入力検証の不備、バッファ オーバーフロー、機密情報の平文保存など、多くの脆弱性に対して複数の CWE が割り当てられています。組織やテスト担当者によって、使用する CWE が異なる場合もあります。複数の CWE を含むカテゴリを使用することで、共通のカテゴリ名の下で発生する可能性のある様々な種類の脆弱性に対する認識を高め、セキュリティ レベルの底上げを図ることができます。この 2025 年版 Top 10 では、10 個のカテゴリに合計 248 個の CWE が含まれています。なお、本リリース時点で、MITRE からダウンロード可能な CWE 辞書には合計 968 個の CWE が登録されています。

カテゴリの選択におけるデータの利用方法

2021 年版と同様に、今回も悪用可能性と（技術的な）影響度に関する CVE データを利用しました。OWASP Dependency Check をダウンロードし、CVSS の悪用可能性スコアと影響度スコアを抽出して、各 CVE に紐付けられた関連する CWE ごとにグループ化しました。すべての CVE には CVSSv2 スコアが付与されていますが、CVSSv2 には CVSSv3 で改善されるべき欠陥があるため、この作業にはかなりの調査と労力が必要でした。一定期間後には、すべての CVE に CVSSv3 スコアも付与されるようになりました。さらに、CVSSv2 と CVSSv3 の間でスコアリングの範囲と計算式が更新されています。

CVSSv2 では、悪用可能性と（技術的な）影響度の両方のスコアは最大で 10.0 でしたが、計算式によって、悪用可能性は 60%、影響度は 40% に調整されていました。CVSSv3 では、理論上の最大値は、悪用可能性が 6.0、影響度が 4.0 に制限されています。重み付けを考慮すると、影響度スコアは CVSSv3 で平均約 1.5 ポイント上昇し、悪用可能性は平均約 0.5 ポイント低下しました。

OWASP Dependency Check から抽出された National Vulnerability Database（NVD）に登録されている CVE と CWE のマッピング データは、約 17 万 5000 件（2021 年の 12 万 5000 件から増加）あります。また、CVE にマッピングされている固有の CWE は 643件（2021 年の 241 件から増加）です。抽出された約 22 万件の CVE のうち、16 万件に CVSSv2 スコア、15 万 6000 件に CVSSv3 スコア、6000 件に CVSSv4 スコアが付与されています。多くの CVE には複数のスコアが付与されているため、合計数は 22 万件を超えています。

2025 年版 Top 10 用に、OWASP は、悪用可能性スコアと影響度スコアの平均値を以下の方法で算出しました。まず、CVSS スコアが付与されているすべての CVE を CWE ごとにグループ化し、CVSSv3 スコアが付与されている CVE の割合と、CVSSv2 スコアが付与されている残りの CVE の割合に基づいて、悪用可能性スコアと影響度スコアの両方に重み付けを行い、全体の平均値を算出しました。これらの平均値をデータセット内の CWE にマッピングし、リスク方程式の後半部分に対して、悪用可能性スコアと（技術的）影響度スコアとして使用しました。

なぜ CVSS v4.0を使用しないのか、という疑問をお持ちの方もいらっしゃるかもしれません。その理由は、スコアリング アルゴリズムが根本的に変更され、CVSSv2 や CVSSv3 のように悪用可能性スコアや影響度スコアを簡単に算出できなくなったためです。今後の Top 10 では、CVSS v4.0の スコアリングを使用する方法を検討する予定ですが、2025 年版については適切な方法を見つけることができませんでした。

コミュニティ調査を利用する理由

データに示されている結果は、主に業界が自動化された方法でテストできる範囲に限定されています。経験豊富なアプリケーション セキュリティの専門家に話を聞けば、データにはまだ反映されていない脆弱性や傾向について教えてくれるでしょう。特定の脆弱性の種類に対するテスト手法を開発するには時間がかかり、さらにそれらのテストを自動化して多数のアプリケーションに対して実行するには、さらに時間がかかります。OWASP が発見するものはすべて過去のものであり、データには含まれていない過去 1 年間のトレンドを見落としている可能性があります。

そのため、データが不完全であることから、10 個のカテゴリのうち 8 つだけを選んでいます。残りの 2 つのカテゴリは、Top 10 のコミュニティ調査に基づいています。これにより、最前線で活躍する専門家が、データには含まれていない（そして今後もデータに表れない可能性のある）最もリスクの高い脆弱性について投票できるようになっています。

データ提供者の皆様への謝辞

以下の組織（および複数の匿名寄付者）のご厚意により、280 万件を超えるアプリケーションに関するデータをご提供いただき、史上最大かつ最も包括的なアプリケーション セキュリティのデータセットが完成しました。皆様のご協力なしには、これは実現不可能でした。

- Accenture (Prague)
- Anonymous (multiple)
- Bugcrowd
- Contrast Security
- CyptoNet Labs
- Intuitor SoftTech Services
- Orca Security
- Probley
- Semgrep
- Sonar
- usd AG
- Veracode

- Wallarm

主著者

- Andrew van der Stock - X: @vanderaj
- Brian Glas - X: @infosecdad
- Neil Smithline - X: @appsecneil
- Tanya Janca - X: @shehackspurple
- Torsten Gigler - Mastodon: @torsten_gigler@infosec.exchange

イシューおよびプル リクエストの記録

修正点や問題点があれば記録してください。

プロジェクトのリンク:

- Homepage
- GitHub repository

OWASP について

Open Worldwide Application Security Project (OWASP) は、信頼できるアプリケーションと API を組織が開発、購入、維持できるようにすることに特化したオープン コミュニティです。

OWASP では、以下のものが無料でオープンに提供されています。

- アプリケーション セキュリティに関するツールおよび標準
- 最先端の研究
- 標準的なセキュリティ管理策およびライブラリ
- 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 レビューに関する書籍一式
- プレゼンテーション資料と 動画
- 多くの一般的なトピックに関する Cheat sheets
- 全世界でオンラインで開催される Chapters meetings
- イベント、トレーニング、会議
- Google Groups

詳細は、https://owasp.org を参照してください。

OWASP のすべてのツール、文書、動画、プレゼンテーション資料、およびチャプターは、アプリケーション セキュリティの向上に関心のある方であれば誰でも無料で利用できます。

OWASP は、アプリケーション セキュリティを人、プロセス、そしてテクノロジーの問題として捉えることを提唱しています。なぜなら、アプリケーショ ンセキュリティへの最も効果的なアプローチは、これらすべての領域の改善を必要とするからです。

OWASP は他とは一線を画す組織です。商業的な圧力から自由な立場にあるため、アプリケーション セキュリティに関する公平で実用的、かつ費用対効果の高い情報を提供することができます。

OWASP は、商用セキュリティ技術の適切な活用を支持していますが、いかなるテクノロジー企業とも提携していません。OWASP は、協力的で透明性が高く、オープンな方法で、様々な資料を作成しています。

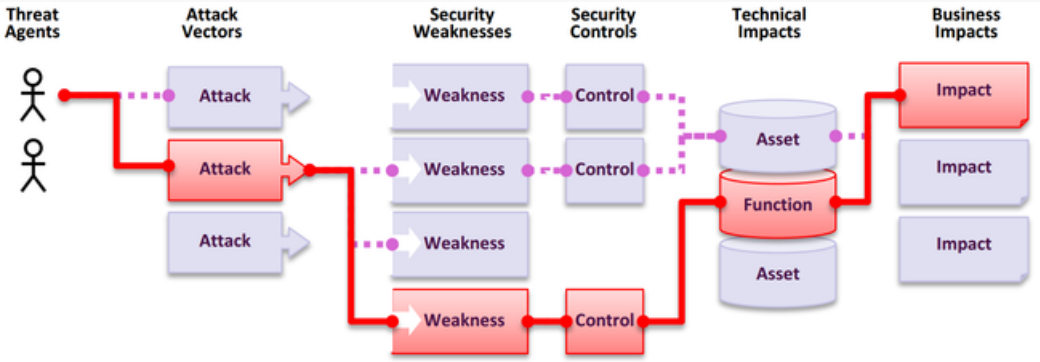
OWASP Foundation は、プロジェクトの長期的な成功を確かなものにする非営利団体です。OWASP 理事会、支部リーダー、プロジェクト リーダー、プロジェクト メンバーなど、OWASP に関わるほぼ全員がボランティアです。私たちは、助成金とインフラを通じて革新的なセキュリティ研究を支援しています。

ぜひご参加ください！

著作権とライセンス

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、アプリケーションを介して様々な経路で侵入し、ビジネスや組織に危害を加える可能性があります。これらの経路はそれぞれ、調査が必要な潜在的なリスクをはらんでいます。



脅威エージェント	攻撃ベクトル	悪用可能性	セキュリティ対策の未実施の可能性	技術的影響	ビジネス影響
環境に依り、状況によって変化	アプリケーションの露出度（環境）に依る	重み付き悪用可能性（平均）	セキュリティ対策の未実施の平均発生率（カバー率による重み付けあり）。	重み付き影響（平均）	ビジネスに依る

OWASP のリスク評価では、悪用可能性、脆弱性に対するセキュリティ対策の未実施率の平均、そしてその技術的影響という普遍的なパラメータを考慮に入れています。

各組織はそれぞれ独自のものであり、その組織における脅威アクター、その目的、そして侵害の影響もそれぞれ異なります。公益団体が公開情報にコンテンツ管理システム（CMS）を使用し、医療システムが機密性の高い医療記録に全く同じ CMS を使用している場合、同じソフトウェアであっても、脅威アクターとビジネスへの影響は大きく異なる可能性があります。アプリケーションの露出度、状況に応じた適切な脅威エージェント（事業内容や場所別の標的型攻撃と非標的型攻撃）、そして個々のビジネスへの影響に基づいて、組織へのリスクを理解することが重要です。

カテゴリーの選択とランク付けにおけるデータの使用方法

2017 年では、発生率に基づいてカテゴリーを選択し、発生可能性を判定しました。その後、数十年にわたる経験に基づき、チームの議論に基づいて、悪用可能性、検出可能性（発生可能性も含む）、技術的影響についてランク付けを行いました。2021 年には、National Vulnerability Database（NVD）の CVSSv2 および CVSSv3 スコアから、悪用可能性と（技術的）影響のデータを使用しました。2025年 も、2021 年に作成した同じ手法を継続しました。

OWASP Dependency Check をダウンロードし、関連する CWE ごとにグループ化された CVSS 悪用可能性スコアおよび影響スコアを抽出しました。すべての CVE に CVSSv2 スコアが割り当てられているため、かなりの調査と労力を要しましたが、CVSSv2 には CVSSv3 で対処すべき欠陥があります。ある時点以降、すべての CVE に CVSSv3 スコアも割り当てられています。さらに、CVSSv2 と CVSSv3 の間では、スコアの範囲と計算式が更新されました。

CVSSv2 では、悪用可能性と（技術的）影響度はどちらも最大 10.0 でしたが、計算式により、悪用可能性は 60%、影響度は 40% まで引き下げられました。CVSSv3 では、理論上の最大値は、悪用可能性が 6.0、影響度が 4.0 に制限されていました。重み付けを考慮すると、CVSSv3 では影響度スコアが平均で約 1.5 ポイント上昇し、2021 年の Top 10 の分析では、悪用可能性スコアは平均で約 0.5 ポイント低下しました。

OWASP Dependency Check から抽出された National Vulnerability Database (NVD) には、CVE が CWE にマッピングされたレコードが約 175,000 件（2021 年の 125,000 件から増加）あります。さらに、CVE にマッピングされた固有の CWE は 643 件（2021 年の 241 件から増加）あります。抽出された約 220,000 件の CVE のうち、160,000 件は CVSS v2 スコア、156,000 件は CVSS v3 スコア、6,000 件は CVSS v4 スコアでした。多くの CVE には複数のスコアがあるため、合計で 220,000 件を超えています。

2025 年の Top 10 では、悪用可能性スコアと影響度スコアの平均を以下の方法で算出しました。CVSS スコアを持つすべての CVE を CWE でグループ化し、悪用可能性スコアと影響度スコアの両方を、CVSSv3 スコアを持つ集団の割合と、CVSSv2 スコアを持つ残りの集団の割合で重み付けして、全体の平均を算出しました。これらの平均値をデータセット内の CWE にマッピングし、リスク方程式のもう一方の半分である悪用可能性スコアおよび（技術的）影響度スコアとして使用しました。

なぜ CVSS v4.0 を使用しないのかと疑問に思うかもしれません。これは、スコアリング アルゴリズムが根本的に変更され、CVSSv2 や CVSSv3 のように悪用可能性スコアや影響度スコアを簡単に提供できなくなったためです。今後のバージョンの Top 10 では CVSS v4.0 スコアを使用する方法を検討しますが、2025年 版では適切な方法を見つけることができませんでした。

発生率については、ある組織が一定期間テストした対象アプリケーション群のうち、各 CWE に対して脆弱なアプリケーションの割合を計算しました。ここで重要なのは、頻度（つまり、アプリケーションで問題が何回発生したか）ではなく、対象アプリケーションの全体のうち、各 CWE が検出された割合です。

カバレッジについては、特定の CWE について、すべての組織でテストされたアプリケーションの割合を調べました。算出されたカバレッジが高いほど、サンプル サイズが対象アプリケーション群をより代表するため、発生率の正確性がより確実になります。

今回のイテレーションで使用した計算式は 2021 年版と同様ですが、次のように重み付けが若干変更されています: (最大発生率% * 1000) + (最大カバレッジ% * 100) + (平均悪用可能性 * 10) + (平均影響度 * 20) + (総発生件数 / 10000) = リスク スコア。

算出されたスコアは、「アクセス制御の不備」カテゴリで 621.60、「メモリ管理エラー」カテゴリで 271.08 でした。

これは完璧なシステムではありませんが、リスク カテゴリーのランク付けには役立ちます。

さらに大きな課題となっているのは、「アプリケーション」の定義です。業界がマイクロ サービスや従来のアプリケーションよりも小規模な実装で構成される異なるアーキテクチャに移行するにつれて、計算はより困難になっています。例えば、組織がコード リポジトリをテストしている場合、何をアプリケーションと見なすのでしょうか？CVSSv4 の成長と同様に、次期 Top 10 では、絶えず変化する業界に対応するために、分析とスコアリングの調整が必要になる可能性があります。

データ要素

Top 10 のカテゴリごとにデータ要素がリストされており、それぞれの意味は以下のとおりです。

対応する CWE 数: Top 10 チームによってカテゴリにマッピングされた CWE の数。

発生率: 発生率は、その組織がその年にテストしたアプリケーションのうち、その CWE に対して脆弱なアプリケーションの割合。

重み付き悪用可能性 CWE にマッピングされた CVE に割り当てられた CVSSv2 および CVSSv3 スコアから算出された悪用可能性サブ スコアを正規化し、10点 満点スケールで評価したもの。

重み付き影響度: CWE にマッピングされた CVE に割り当てられた CVSSv2 および CVSSv3 スコアから算出された影響度サブスコアを正規化し、10 点満点スケールで評価したもの。

（テスト）カバレッジ: 特定の CWE について、すべての組織でテストされたアプリケーションの割合。

総発生件数 特定のカテゴリに CWE がマッピングされているアプリケーションの総数。

総 CVE 数: 特定のカテゴリにマッピングされている CWE にマッピングされている NVD DB 内の CVE の総数。

計算式: (最大発生率% * 1000) + (最大カバレッジ% * 100) + (平均悪用可能性 * 10) + (平均影響度 * 20) + (総発生件数 / 10000) = リスクスコア

現代的なアプリケーション セキュリティ プログラムの確立

OWASP Top 10 のリストは、対象となるトピックにおける最も重要なリスクへの意識向上を目的とした意識啓発のための文書です。これらは完全なリストではなく、あくまで出発点となるものです。以前のバージョンでは、これらのリスクなどを回避する最善の方法として、アプリケーション セキュリティプ ログラムの開始を推奨していました。このセクションでは、現代的なアプリケーション セキュリティ プログラムを開始および構築する方法について説明します。

既にアプリケーション セキュリティ プログラムを持っている場合は、OWASP SAMM (Software Assurance Maturity Model) または DSOMM (DevSecOps Maturity Model) を使用して、成熟度評価を実施することを検討します。これらの成熟度モデルは包括的かつ網羅的であり、プログラムの拡張と成熟化に向けて、どこに重点的に取り組むべきかを判断するのに役立ちます。ただし、OWASP SAMM または DSOMM に記載されているすべての手順を実行する必要はありません。これらのモデルは、ガイドとして、また多くの選択肢を提供することを目的としているためです。これらは、達成不可能な基準を提示したり、予算が足りないプログラムを説明するものではありません。多くのアイデアや選択肢を提供するために、広範囲にわたっています。

プログラムをゼロから始める場合、または OWASP SAMM や DSOMM が現状のチームにとって「過度である」と感じている場合は、以下のアドバイスをレビューします。

1. リスク ベースのポートフォリオ アプローチの確立

- 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ニーズを特定します。これは、保護対象の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基づいて決定する必要があります。
- 組織のリスク許容度を反映した、一貫した発生可能性と影響度を持つ共通のリスク評価モデルを確立します。
- 上記に基づいて、すべてのアプリケーションと API を測定し、優先順位を付けます。結果を構成管理データベース (CMDB) に追加します。
- 必要な範囲と厳密さのレベルを適切に定義するための保証ガイドラインを確立します。

2. 強固な基盤によるイネーブルメント

- すべ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となる、焦点を絞ったポリシーと標準を確立します。
- これらのポリシーと標準を補完する、再利用可能なセキュリティ管理策の共通セットを定義し、その使用に関する設計および開発ガイダンスを提供します。
- 開発における様々な役割やトピックを対象とし、必要かつ適切なアプリケーション セキュリティ トレーニングのカリキュラムを確立します。

3. 既存のプロセスへのセキュリティの統合

- セキュアな実装と検証活動を定義し、既存の開発プロセスおよび運用プロセスに統合します。
- 活動には、脅威モデリング、セキュア設計と設計レビュー、セキュア 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、修復が含まれます。
- 開発チームとプロジェクト チームの成功を支援するため、専門家とサポート サービスを提供します。
- 現在のシステム開発ライフ サイクルと、すべてのソフトウェア セキュリティ活動、ツール、ポリシー、プロセスをレビューし、文書化します。

- 新しいソフトウェアについては、システム開発ライフサイクル (SDLC) の各フェーズに 1 つ以上のセキュリティ活動を追加します。以下では、実行できる多くの提案を示します。すべての新規プロジェクトまたはソフトウェア イニシアチブでこれらの新しい活動を実施することで、新しいソフトウェアが組織にとって許容可能なセキュリティ体制で提供されることを確実にします。
- 最終製品が組織にとって許容可能なリスク レベルを満たすように、活動を選択します。
- 既存のソフトウェア (レガシーと呼ばれることもあります) については、正式な維持計画が必要です。セキュアなアプリケーションを維持する方法については、以下の「運用と変更管理」セクションを確認します。

4. アプリケーション セキュリティ教育

- 開発者に知っておいてほしいことをすべて教えるために、セキュリティ チャンピオン プログラム、または開発者向けの一般的なセキュリティ教育プログラム（セキュリティ推進プログラム（訳注: 原文は advocay program）やセキュリティ意識向上プログラムと呼ばれることもあります）の開始を検討します。これにより、開発者は最新の情報を入手し、セキュアに作業を行う方法を理解できるようになり、現場のセキュリティ文化がより前向きになります。また、チーム間の信頼関係が強化され、円滑な業務関係が築かれることも少なくありません。OWASP は、OWASP Security Champions Guide を通じて、この点を支援しており、このガイドは段階的に拡充されています。
- OWASP Education Project は、Web アプリケーション セキュリティに関する開発者の教育に役立つトレーニング資料を提供しています。脆弱性に関する実践的な学習には、OWASP Juice Shop Project または OWASP WebGoat で試すことができます。最新情報を入手するには、OWASP AppSec Conference、OWASP AppSec Conference, OWASP Conference Training、またはローカルの OWASP Chapter ミーティングに参加します。

5. 管理の可視性の提供

- 指標を用いて管理します。指標と収集した分析データに基づいて、改善と予算配分の決定を推進します。指標には、セキュリティ対策および活動の遵守状況、導入された脆弱性、軽減された脆弱性、アプリケーション カバレッジ、種類別の不具合密度、事例数などが含まれます。
- 実装および検証活動から得られたデータを分析し、根本原因と脆弱性のパターンを特定することで、企業全体にわたる戦略的かつ体系的な改善を推進します。失敗から学び改善を促進するための積極的なインセンティブを提供します。

反復可能なセキュリティ プロセスと標準セキュリティ管理策の確立と使用

要件およびリソース管理フェーズ

- アプリケーションのビジネス要件を収集し、ビジネス部門と交渉します。これには、すべてのデータ資産の機密性、真正性、完全性、可用性に関する保護要件、および想定されるビジネス ロジックが含まれます。
- 機能的および非機能的なセキュリティ要件を含む技術要件をまとめます。OWASP は、アプリケーションのセキュリティ要件を設定するためのガイドとして、OWASP Application Security Verification Standard (ASVS) の使用を推奨しています。
- セキュリティ活動を含む、設計、構築、テスト、運用のあらゆる側面をカバーする予算を計画し、交渉します。
- プロジェクト スケジュールにセキュリティ活動を追加します。
- プロジェクトのキックオフ時にセキュリティ担当者として参画し、担当者が誰と話をすればよいかを把握できるようにします。

提案依頼書 (RFP) と契約

- 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（SDLC、ベスト プラクティスなど）を含む要件について、内部または外部の開発者と交渉します。
- 計画・設計フェーズを含む、すべての技術要件の達成状況を評価します。
- 設計、セキュリティ、サービス レベル契約 (SLA) を含む、すべての技術要件について交渉します。
- OWASP Secure Software Contract Annex[(1)] などのテンプレートとチェック リストを採用します。
  ****1: 注* この附属書は米国の契約法に関するものです。サンプルの附属書を使用する前に、必ず資格のある法律専門家に相談する必要があります。*

計画・設計フェーズ

- 開発者や内部関係者（セキュリティ専門家など）と、計画と設計について協議します。
- 保護ニーズと想定される脅威レベルに適したセキュリティ アーキテクチャ、コントロール、対策、設計レビューを定義します。これにはセキュリティ専門家のサポートが必要です。
- アプリケーションや API に後からセキュリティを組み込むよりも、最初からセキュリティを設計に組み込む方がはるかに費用対効果が高いです。OWASP は、OWASP Cheat Sheet: Threat Modeling と OWASP Proactive Controls を、最初からセキュリティを組み込んだ設計方法のガイダンスとして推奨しています。
- 脅威モデリングを実施します。実施においては、OWASP Cheat Sheet: Threat Modeling を参照します。
- ソフトウェア アーキテクトにセキュアな設計の考え方とパターンを教え、可能な場合は設計に取り入れるよう依頼します。
- 開発者と共にデータ フローを検証します。
- 他のユーザー ストーリーに加えて、セキュリティに関するユーザー ストーリーも追加します。

セキュアな開発ライフ サイクル

- 組織がアプリケーションや API を構築する際のプロセスを改善するために、OWASP は OWASP Software Assurance Maturity Model (SAMM) を推奨しています。このモデルは、組織が直面する特定のリスクに合わせたソフトウェア セキュリティ戦略を策定・実装するのに役立ちます。
- ソフトウェア開発者に、セキュア コーディングのトレーニング、およびより堅牢でセキュアなアプリケーションの作成に役立つと思われるその他のトレーニングを提供します。

- コード レビューについては、OWASP Cheat Sheet: Secure Code Review を参照します。
- 開発者にセキュリティ ツールを提供し、その使い方を指導します。特に、静的解析、ソフトウェア構成解析、シークレット情報、および Infrastructure-as-Code (IaC) 用のスキャナーについて指導します。
- 可能であれば、開発者向けのガードレール（よりセキュアな選択肢へと導くための技術的安全策）を作成します。
- 強力で使いやすいセキュリティ管理策を構築するのは困難です。可能な限りセキュアなデフォルト設定を提供し、「舗装された道」（何かをするにあたり最も簡単で最もセキュアな方法、つまり明らかに好ましい方法）を可能な限り用意します。OWASP Cheat Sheets は開発者にとって良い出発点となり、多くの最新フレームワークには、認証、検証、CSRF 防止などのための標準的で効果的なセキュリティ管理策が組み込まれています。
- 開発者にセキュリティ関連の IDE プラグインを提供し、使用を促します。
- シークレット情報管理ツール、ライセンス、使用方法に関する文書を提供します。
- 開発者が使用できるプライベート AI を提供します。理想的には、有用なセキュリティ文書が満載の RAG サーバー、より良い結果を得るためにチームが作成したプロンプト、そして組織で選択したセキュリティ ツールを呼び出す MCP サーバーでセットアップします。開発者に AI を安全に使用する方法を教えます。好むと好まざるとにかかわらず、開発者は AI を使うことになるからです。

継続的なアプリケーション セキュリティ テストの確立

- 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 テストを調整します。
- 技術面とビジネス面の両方の観点から、「使用」と「乱用」のテストケースを作成します。
- 内部プロセス、保護ニーズ、アプリケーションが想定する脅威レベルに応じて、セキュリティ テストを管理します。
- セキュリティ テスト ツール（ファジング ツール、DAST など）、安全なテスト環境、およびそれらの使用方法に関するトレーニングを提供します。あるいは、テストを代行するか、テスターを雇用します。
- 高いレベルの保証が必要な場合は、正式なペネトレーション テストに加え、ストレス テストとパフォーマンス テストの実施も検討します。
- 開発者と協力し、バグ レポートに基づいて修正が必要な箇所を決定できるよう支援します。また、管理者が決定のための時間を確保できるようにします。

展開（ロールアウト）

- アプリケーションの運用を開始し、必要に応じて以前使用していたアプリケーションから移行します。
- 変更管理データベース (CMDB) とセキュリティ アーキテクチャを含むすべての文書を完成させます。

運用と変更管理

- 運用には、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（パッチ管理など）を含める必要があります。
- ユーザーのセキュリティ意識を高め、ユーザビリティとセキュリティの両立に関する矛盾を管理します。
- 変更を計画し、管理します（例: アプリケーションの新バージョンへの移行。OS、ミドルウェア、ライブラリなどのコンポーネントへの移行）、
- すべてのアプリケーションがインベントリに登録され、重要な詳細がすべて文書化されていることを確実にします。CMDB、セキュリティ アーキテクチャ、管理策、対策、作業手順書やプロジェクト文書など、すべての文書を更新します。
- すべてのアプリケーションに対して、ログ記録、監視、アラート発報を実行します。不足している場合は追加します。
- 効果的かつ効率的な更新実施とパッチ適用のためのプロセスを作成します。
- 定期的なスキャン スケジュールを作成します（動的スキャン、静的スキャン、シークレット情報スキャン、IaC スキャン、ソフトウェア構成分析など）。
- セキュリティ バグの修正に関する SLA を作成します。
- 従業員（そしてできれば顧客も）がバグを報告できる手段を提供します。
- ソフトウェア攻撃の様相を理解し、監視ツールを備えた、トレーニングを受けたインシデント対応チームを編成します。
- 自動攻撃を阻止するためのブロッキング ツールまたはシールド ツールを実行します。
- 年 1 回（またはより頻繁に）、構成を強化します。
- （アプリケーションに必要なセキュリティ レベルに応じて）少なくとも年 1 回、ペネトレーション テストを実施します。
- ソフトウェア サプライチェーンの強化と保護のためのプロセスとツールを確立します。
- 最も重要なアプリケーションと、それらの維持に使用するツールを含む、事業継続性と災害復旧計画を策定・更新します。

システムの廃止

- 必要なデータはすべてアーカイブする必要があります。その他のデータはすべてセキュアに消去する必要があります。
- 未使用のアカウント、ロール、権限の削除など、アプリケーションをセキュアに廃止します。
- CMDB でアプリケーションの状態を廃止に設定します。

OWASP Top 10 の標準としての使用

OWASP Top 10 は、主に意識向上を目的とした文書です。しかしながら、2003 年の発表以来、多くの組織がこれを事実上の業界標準として利用し続けています。OWASP Top 10 をコーディング標準やテスト標準として使用する場合は、これが最低限の標準であり、出発点に過ぎないことを理解しておく必要があります。

OWASP Top 10 を標準として使用する際の難しさの一つは、文書化されているのがアプリケーション セキュリティのリスクであり、必ずしも簡単にテストできる問題ではないことです。例えば、A06:2025-セキュリティが確保されていない設計 は、ほとんどのテストの範囲を超えています。また、ログ記録と監視が適切にかつ運用中に効果的に実装されているかどうかをテストすることも重要ですが、これはインタビューや効果的なインシデント対応のサンプル調査によってのみ実施可能です。静的コード解析ツールはログ記録の欠如を検出できますが、ビジネス ロジックやアクセス制御が重大なセキュリティ侵害を記録しているかどうかを判断できない可能性があります。ペネトレーション テスターは、テスト環境でインシデント対応が発動されたことしか判断できない可能性があります。これは、テスト環境は、本番環境と同じように監視されることはほとんどないためです。

OWASP Top 10 の使用が適切な場合の推奨事項を以下に示します。

利用事例	OWASP Top 10 2025	OWASP Application Security Verification Standard
意識向上	適している	
トレーニング	入門レベル	包括的
設計とアーキテクチャ	稀に適する場合あり	適している
コーディング標準	最低限のみ	適している
セキュアなコード レビュー	最低限のみ	適している
ピア レビューのチェックリスト	最低限のみ	適している
単体テスト	稀に適する場合あり	適している
統合テスト	稀に適する場合あり	適している
ペネトレーション テスト	最低限のみ	適している
ツール支援	最低限のみ	適している
セキュアなサプライチェーン	稀に適する場合あり	適している

アプリケーション セキュリティ標準の導入を検討されている方は、OWASP Application Security Verification Standard (ASVS) の使用を推奨します。ASVS は検証とテストが可能なように設計されており、セキュアな開発ライフ サイクルのあらゆる段階で使用できます。

ツール ベンダーにとって、ASVS は唯一の選択肢です。A06:2025-セキュリティが確保されていない設計 に記載されているように、OWASP Top 10 のリスクのいくつかは、その性質上、ツールでは OWASP Top 10 を包括的に検出、テスト、または防御することはできません。OWASP は、OWASP Top 10 を完全に網羅しているという主張を推奨しません。それは、単に真実ではないからです。

A01:2025 アクセス制御の不備

背景

Top 10 で 1 位を維持し、アプリケーションの 100% で何らかのアクセス制御の不備がテストされました。注目すべき CWE には、*CWE-200:* 権限のないアクターによる機密情報の漏洩、*CWE-201:* 送信データを介した機密情報の意図せぬ公開、*CWE-918:* サーバー サイド リクエスト フォージェリ *(SSRF)*、*CWE-352:* クロスサイト リクエスト フォージェリ *(CSRF)* が含まれます。このカテゴリは、提供されたデータの中で最も多くの発生件数を記録し、関連する CVE の数も 2 番目に多くなっています。

スコア表

対応する CWE 数	インシデント割合（最大）	インシデント割合（平均）	カバレッジ（最大）	カバレッジ（平均）	重み付き悪用可能性（最大）	重み付き悪用可能性（平均）	総発生件数	総 CVE 数
40	20.15%	3.74%	100.00%	42.93%	7.04	3.84	1,839,701	32,654

解説

アクセス制御は、ユーザーが意図した権限の範囲外で操作できないようにポリシーを執行します。アクセス制御の不備は、通常、許可されていない情報の開示、全データの変更または破壊、あるいはユーザーの権限を超えたビジネス機能の実行につながります。一般的なアクセス制御の脆弱性には、以下のようなものがあります。

- 最小権限の原則（一般的には「デフォルトで拒否」とも呼ばれます）の違反。アクセスは特定の機能、ロール、またはユーザーにのみ付与されるべきであるにもかかわらず、誰でもアクセス可能です。
- URL（パラメータの改ざんや強制ブラウジング）、アプリケーション内部の状態、または HTML ページの変更、あるいは API リクエストを変更する攻撃ツールの使用により、アクセス制御チェックをバイパスします。
- 一意の識別子を与えることで、他のユーザーのアカウントの閲覧または編集を許可します（オブジェクトの安全でない直接参照）。
- POST、PUT、および DELETE のアクセス制御が欠落している、アクセス可能な API。
- 権限の昇格。ログインしていない状態でユーザーとして操作したり、ユーザーとしてログインしている状態で管理者として操作したりできます。

- メタデータ操作（JSON Web Token (JWT) アクセス制御トークンのリプレイまたは改ざん、権限昇格を目的とした Cookie または隠しフィールドの操作、JWT 無効化の悪用など）。
- CORS の設定ミスにより、許可されていないまたは信頼できないオリジンからの API アクセスが許可されます。
- 認証されていないユーザーとして認証済みのページを、または標準ユーザーとして権限のあるページを強制的に閲覧（URL を推測）します。

防止方法

アクセス制御は、攻撃者がアクセス制御チェックやメタデータを変更できない、サーバー側の信頼できるコードまたはサーバーレス API に実装されている場合にのみ有効です。

- パブリック リソースを除き、デフォルトで拒否します。
- アクセス制御機構は一度実装し、アプリケーション全体で再利用します。これには、クロス オリジン リソース共有 (CORS) の使用を最小限に抑えることも含まれます。
- モデル レベルのアクセス制御では、ユーザーが任意のレコードを作成、読み取り、更新、または削除できるようにするのではなく、レコードの所有権を強制する必要があります。
- 固有のアプリケーション ビジネス制限要件は、ドメイン モデルによって強制される必要があります。
- Web サーバーのディレクトリ一覧表示を無効にし、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 Web ルート内に存在しないことを確実にします。
- アクセス制御の失敗（例: 繰り返し失敗する場合）をログに記録し、必要に応じて管理者に警告します。
- 自動攻撃ツールによる被害を最小限に抑えるため、API およびコントローラーへのアクセスにレート制限を実装します。
- ステートフル セッション識別子は、ログアウト後にサーバー上で無効化する必要があります。ステートレス JWT トークンは、攻撃者の攻撃機会を最小限に抑えるため、有効期間を短くする必要があります。より長期間有効な JWT の場合、アクセスを取り消すには OAuth 標準に従うことを強く推奨します。
- 簡潔で宣言的なアクセス制御を提供する、確立されたツールキットまたはパターンを使用します。

開発者と QA スタッフは、単体テストと統合テストに機能的なアクセス制御を含める必要があります。

攻撃シナリオの例

シナリオ #1: アプリケーションは、アカウント情報にアクセスする SQL 呼び出しで未検証のデータを使用します。

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

攻撃者は、ブラウザの「acct」パラメータを変更するだけで、任意のアカウント番号を送信できます。正しく検証されていない場合、攻撃者は任意のユーザーのアカウントにアクセスできます。

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

シナリオ #2: 攻撃者は、ブラウザで特定の URL を強制的にアクセスするだけです。管理ページにアクセスするには管理者権限が必要です。

```
https://example.com/app/getappInfo
https://example.com/app/admin_getappInfo
```

認証されていないユーザーがどちらかのページにアクセスできる場合、それは欠陥です。管理者以外のユーザーが管理ページにアクセスできる場合も、それは欠陥です。

シナリオ #3: アプリケーションはすべてのアクセス制御をフロント エンドに集約しています。攻撃者は、ブラウザで JavaScript コードが実行されているため `https://example.com/app/admin_getappInfo` にアクセスできませんが、コマンド ラインから以下のコマンドを実行するだけでアクセス可能です。

```
$ curl https://example.com/app/admin_getappInfo
```

参考情報

- OWASP Proactive Controls: C1: Implement Access Control
- OWASP Application Security Verification Standard: V8 Authorization
- OWASP Testing Guide: Authorization Testing
- OWASP Cheat Sheet: Authorization
- PortSwigger: Exploiting CORS misconfiguration
- OAuth: Revoking Access

対応する CWE の一覧

- CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

- CWE-23 Relative Path Traversal
- CWE-36 Absolute Path Traversal
- CWE-59 Improper Link Resolution Before File Access ('Link Following')
- CWE-61 UNIX Symbolic Link (Symlink) Following
- CWE-65 Windows Hard Link
- CWE-200 Exposure of Sensitive Information to an Unauthorized Actor
- CWE-201 Exposure of Sensitive Information Through Sent Data
- CWE-219 Storage of File with Sensitive Data Under Web Root
- CWE-276 Incorrect Default Permissions
- CWE-281 Improper Preservation of Permissions
- CWE-282 Improper Ownership Management
- CWE-283 Unverified Ownership
- CWE-284 Improper Access Control
- CWE-285 Improper Authorization
- CWE-352 Cross-Site Request Forgery (CSRF)
- CWE-359 Exposure of Private Personal Information to an Unauthorized Actor
- CWE-377 Insecure Temporary File
- CWE-379 Creation of Temporary File in Directory with Insecure Permissions
- CWE-402 Transmission of Private Resources into a New Sphere ('Resource Leak')
- CWE-424 Improper Protection of Alternate Path
- CWE-425 Direct Request ('Forced Browsing')
- CWE-441 Unintended Proxy or Intermediary ('Confused Deputy')
- CWE-497 Exposure of Sensitive System Information to an Unauthorized Control Sphere
- CWE-538 Insertion of Sensitive Information into Externally-Accessible File or Directory
- CWE-540 Inclusion of Sensitive Information in Source Code
- CWE-548 Exposure of Information Through Directory Listing
- CWE-552 Files or Directories Accessible to External Parties
- CWE-566 Authorization Bypass Through User-Controlled SQL Primary Key
- CWE-601 URL Redirection to Untrusted Site ('Open Redirect')
- CWE-615 Inclusion of Sensitive Information in Source Code Comments
- CWE-639 Authorization Bypass Through User-Controlled Key
- CWE-668 Exposure of Resource to Wrong Sphere
- CWE-732 Incorrect Permission Assignment for Critical Resource
- CWE-749 Exposed Dangerous Method or Function
- CWE-862 Missing Authorization
- CWE-863 Incorrect Authorization
- CWE-918 Server-Side Request Forgery (SSRF)
- CWE-922 Insecure Storage of Sensitive Information
- CWE-1275 Sensitive Cookie with Improper SameSite Attribute

A02:2025 セキュリティ設定のミス

背景

前回の 5 位からランク アップし、テスト対象アプリケーションの 100% に何らかの設定ミスが見つかりました。平均発生率は 3.00% で、このリスク カテゴリにおける共通脆弱性リスト（CWE）の出現件数は 71 万 9 千件を超えています。高度に構成可能なソフトウェアへの移行が進む中、このカテゴリのランクが上がるのは当然のことです。注目すべき CWE としては、*CWE-16* 設定 と *CWE-611 XML* 外部実体参照（*XXE*）の不適切な制限 が挙げられます。

スコア表

対応する CWE 数	インシデント割合（最大）	インシデント割合（平均）	カバレッジ（最大）	カバレッジ（平均）	重み付き悪用可能性（最大）	重み付き悪用可能性（平均）	総発生件数	総 CVE 数
16	27.70%	3.00%	100.00%	52.35%	7.96	3.97	719,084	1,375

解説

セキュリティ設定のミスとは、システム、アプリケーション、またはクラウド サービスがセキュリティの観点から不適切に設定され、脆弱性が生じることです。

アプリケーションが脆弱になる可能性があるのは、以下のような場合です。

- アプリケーション スタックのどこかに適切なセキュリティ強化が施されていません。またはクラウド サービスに対する権限が適切に構成されていません。
- 不要な機能（不要なポート・サービス・ページ・アカウント、テスト用フレームワーク・権限など）が有効化またはインストールされています。
- デフォルトのアカウントとそのパスワードが有効化され、変更されていません。
- 過剰なエラー メッセージを遮断するための集中管理設定が不足しています。エラー処理によって、スタック トレースやその他の過剰な情報を含むエラー メッセージがユーザーに公開されます。
- アップグレードされたシステムで、最新のセキュリティ機能が無効化されているか、セキュアに構成されていません。
- 後方互換性を過度に優先し、セキュアでない構成になっています。
- アプリケーション サーバー、アプリケーション フレームワーク（Struts、Spring、ASP.NETなど）、ライブラリ、データベースなどのセキュリティ設定がセキュアな値に設定されていません。
- サーバーがセキュリティ ヘッダーまたはセキュリティ ディレクティブを送信しないか、それらがセキュアな値に設定されていません。

協調的かつ反復可能なアプリケーション セキュリティ設定の強化プロセスがなければ、システムはより高いリスクにさらされます。

防止方法

以下の項目を含む、セキュアなインストール プロセスを実装する必要があります。

- 適切にロック ダウンされた別の環境を迅速かつ容易にデプロイできる繰り返し可能な強化プロセス。開発環境、QA 環境、本番環境はすべて同一構成とし、各環境で異なる認証情報を使用する必要があります。このプロセスは自動化することで、新しいセキュアな環境の構築に必要な労力を最小限に抑えることができます。
- 不要な機能・コンポーネント・文書・サンプルを含まない最小限のプラットフォーム。使用しない機能やフレームワークは削除するか、インストールしてはなりません。
- パッチ管理プロセスの一環として、すべてのセキュリティ ノート、アップデート、パッチに適切な構成をレビューし、更新するタスク（A03:2025-ソフトウェア サプライチェーンの失敗 を参照）。クラウド ストレージの権限（S3 バケットの権限など）をレビューします。
- セグメント化され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、コンテナ化、またはクラウド セキュリティ グループ（ACL）によって、コンポーネントまたはテナント間の効果的かつセキュアな分離を実現します。
- セキュリティ ヘッダーなどのセキュリティ ディレクティブをクライアントに送信します。
- すべての環境における構成と設定の有効性を検証する自動プロセス。
- 過剰なエラーメッセージをバックアップとして傍受するための中央構成を事前対処的に追加します。
- これらの検証が自動化されていない場合は、少なくとも年に 1 回は手動で検証する必要があります。
- コード、構成ファイル、パイプラインに静的な鍵やシークレット情報を埋め込む代わりに、基盤となるプラットフォームによって提供される ID フェデレーション、有効期間の短い資格情報、またはロール ベースのアクセス機構を使用します。

攻撃シナリオの例

シナリオ #1: アプリケーション サーバーには、本番サーバーから削除されていないサンプル アプリケーションが付属しています。これらのサンプル アプリケーションには、攻撃者がサーバーへの侵入に利用する既知のセキュリティ上の欠陥があります。これらのアプリケーションの 1 つが管理コンソールで、デフォルトのアカウントが変更されていないと仮定します。この場合、攻撃者はデフォルトのパスワードでログインし、サーバーを乗っ取ることができます。

シナリオ #2: サーバー上でディレクトリ一覧表示が無効化されていません。攻撃者は、ディレクトリ一覧表示が簡単にできることを発見します。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、逆コンパイルしてリバース エンジニアリングを行い、コードを確認します。そして、アプリケーションに重大なアクセス制御の欠陥があることを発見します。

シナリオ #3: アプリケーション サーバーの設定により、スタック トレースなどの詳細なエラー メッセージをユーザーに返すことが可能になっています。これにより、脆弱性が知られているコンポーネントのバージョンなど、機密情報や潜在的な欠陥が明らかになる可能性があります。

シナリオ #4: クラウド サービス プロバイダー (CSP) は、デフォルトで共有権限をインターネットに公開しています。これにより、クラウド ストレージに保存されている機密データにアクセスできるようになります。

参考情報

- OWASP Testing Guide: Configuration Management
- OWASP Testing Guide: Testing for Error Codes
- Application Security Verification Standard V13 Configuration
- NIST Guide to General Server Hardening
- CIS Security Configuration Guides/Benchmarks
- Amazon S3 Bucket Discovery and Enumeration
- ScienceDirect: Security Misconfiguration

対応する CWE の一覧

- CWE-5 J2EE Misconfiguration: Data Transmission Without Encryption
- CWE-11 ASP.NET Misconfiguration: Creating Debug Binary
- CWE-13 ASP.NET Misconfiguration: Password in Configuration File
- CWE-15 External Control of System or Configuration Setting
- CWE-16 Configuration
- CWE-260 Password in Configuration File
- CWE-315 Cleartext Storage of Sensitive Information in a Cookie
- CWE-489 Active Debug Code
- CWE-526 Exposure of Sensitive Information Through Environmental Variables
- CWE-547 Use of Hard-coded, Security-relevant Constants
- CWE-611 Improper Restriction of XML External Entity Reference
- CWE-614 Sensitive Cookie in HTTPS Session Without 'Secure' Attribute
- CWE-776 Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')
- CWE-942 Permissive Cross-domain Policy with Untrusted Domains
- CWE-1004 Sensitive Cookie Without 'HttpOnly' Flag
- CWE-1174 ASP.NET Misconfiguration: Improper Model Validation

A03:2025 ソフトウェア サプライチェーンの失敗

背景

これは Top 10 のコミュニティ調査で第 1 位にランクインし、回答者のちょうど 50% が 1 位に挙げました。2013 年の Top 10 に「A9 – 既知の脆弱性を持つコンポーネントの使用」として初めて登場して以来、このリスクは既知の脆弱性に関連するものだけでなく、サプライチェーン全体におけるあらゆる問題を含むように範囲が拡大しました。このように範囲が拡大したにもかかわらず、サプライチェーンの問題は依然として特定が困難であり、関連する CWE を持つ共通脆弱性識別子（CVE）はわずか 11 件しかありません。しかし、提供されたデータに基づいてテストおよび報告された結果では、このカテゴリの平均発生率は 5.19% と最も高くなっています。関連する CWE は、*CWE-477:* 非推奨機能の使用、*CWE-1104:* メンテナンスされていないサードパーティ コンポーネントの使用、*CWE-1329:* 更新不可能なコンポーネントへの依存、および *CWE-1395:* 脆弱なサードパーティコンポーネントへの依存です。

スコア表

対応する CWE 数	インシデント割合（最大）	インシデント割合（平均）	カバレッジ（最大）	カバレッジ（平均）	重み付き悪用可能性（最大）	重み付き悪用可能性（平均）	総発生件数	総 CVE 数
5	8.81%	5.19%	65.42%	28.93%	8.17	5.23	215,248	11

解説

ソフトウェア サプライチェーンの失敗とは、ソフトウェアの構築、配布、または更新プロセスにおける不具合やその他の侵害のことです。これらは多くの場合、システムが依存しているサード パーティ製のコード、ツール、またはその他の依存関係における脆弱性や悪意のある変更によって引き起こされます。

以下のような場合は、脆弱な状態にある可能性が高いです。

- 使用しているすべてのコンポーネント（クライアント側とサーバー側の両方）のバージョンを注意深く追跡していない場合。これには、直接使用しているコンポーネントだけでなく、ネストされた（推移的な）依存関係にあるコンポーネントも含まれます。

- ソフトウェアに脆弱性がある、サポートが終了している、または古いバージョンである場合。これには、OS、Web/アプリケーション サーバー、データベース管理システム（DBMS）、アプリケーション、API、およびすべてのコンポーネント、ランタイム環境、ライブラリが含まれます。
- 脆弱性スキャンを定期的に実行せず、使用しているコンポーネントに関連するセキュリティ情報を購読していない場合。
- 変更管理プロセスがない、またはサプライチェーン内の変更を追跡していない場合。これには、IDE、IDE 拡張機能とアップデート、組織のコード リポジトリ、サンドボックス、イメージおよびライブラリのリポジトリへの変更、アーティファクトの作成および保存方法などが含まれます。サプライチェーンのすべての部分は、特に変更点について文書化する必要があります。
- アクセス制御と最小権限の原則の適用に特に重点を置いて、サプライチェーンのすべての部分を堅牢化していない場合。
- サプライチェーン システムでは職務分離がされていない場合。単一の人物が、他の人間の監視なしにコードを記述し、本番環境にデプロイできるべきではありません。
- 技術スタックのあらゆる部分にわたる信頼できないソースからのコンポーネントが、実稼働環境で使用されているか、実稼働環境に影響を及ぼす可能性がある場合。
- 基盤となるプラットフォーム、フレームワーク、および依存関係を、リスク ベースでタイムリーに修正またはアップグレードしていない場合。これは、パッチ適用が変更管理の下で月次または四半期ごとのタスクになっている環境でよく発生し、組織は脆弱性を修正するまで数日から数ヶ月間、不必要なリスクにさらされることになります。
- ソフトウェア開発者が、更新、アップグレード、またはパッチ適用されたライブラリの互換性をテストしていない場合。
- システムのすべての部分の構成をセキュアに設定していない場合（A02:2025-セキュリティ設定ミス を参照）。
- CI/CD パイプライン（特に複雑な場合）のセキュリティが CI/CD がビルドしデプロイするシステムに比べて弱い場合。

防止方法

パッチ管理プロセスを導入して、以下を実施する必要があります。

- ソフトウェア全体のソフトウェア部品表（SBOM）を把握し、SBOM 辞書を一元的に管理します。
- 自身の依存関係だけでなく、それらの（推移的な）依存関係なども追跡します。
- 使用されていない依存関係、不要な機能、コンポーネント、ファイル、ドキュメントを削除して、攻撃対象領域を削減します。
- バージョン管理ツール、OWASP Dependency Check、retire.js などのツールを使用して、クライアント側とサーバー側のコンポーネント（フレームワーク、ライブラリなど）とその依存関係のバージョンを継続的にインベントリ化します。
- 使用しているコンポーネントの脆弱性について、Common Vulnerability and Exposures（CVE）や National Vulnerability Database（NVD）などの情報源を継続的に監視します。ソフトウェア構成分析、ソフトウェア サプライチェーン、またはセキュリティに特化した SBOM ツールを使用してプロセスを自動化します。使用しているコンポーネントに関連するセキュリティ脆弱性に関するメール アラートを購読します。
- コンポーネントは、セキュアなリンクを介して公式の（信頼できる）ソースからのみ入手します。改ざんされた悪意のあるコンポーネントが含まれる可能性を減らすために、署名付きパッケージを優先します（A08:2025-ソフトウェアとデータの整合性の不具合を参照）。
- 使用する依存関係のバージョンを意図的に選択し、必要な場合にのみアップグレードします。
- メンテナンスされていないライブラリやコンポーネント、または古いバージョンに対するセキュリティ パッチが提供されていないライブラリやコンポーネントを監視します。パッチ適用が不可能な場合は、検出された問題に対する監視、検出、または保護のために仮想パッチの導入を検討します。
- CI/CD、IDE、およびその他の開発ツールを定期的に更新します。
- すべてのシステムへのアップデートの同時デプロイは避けます。信頼できるベンダーが侵害された場合に備えて、段階的なロール アウトやカナリア デプロイメントを実施し、リスクを軽減します。

変更管理プロセスや追跡システムを導入して、以下に対する変更を追跡する必要があります。

- CI/CD の設定（すべてのビルド ツールとパイプライン）
- コード リポジトリ
- サンドボックス環境
- 開発者向け IDE
- SBOM ツールおよび生成されたアーティファクト
- ログ記録システムとログ
- SaaS などのサード パーティ統合
- アーティファクト リポジトリ
- コンテナ レジストリ

以下のシステムを堅牢化します。これには、MFA の有効化と IAM のロック ダウンが含まれます。

- コード リポジトリ（シークレットのチェック イン禁止、ブランチ保護、バックアップなどを含む）
- 開発者ワークステーション（定期的なパッチ適用、MFA、監視など）
- ビルド サーバーおよび CI/CD（職務分掌、アクセス制御、署名付きビルド、環境スコープの機密情報、改ざん防止ログなど）
- アーティファクト（出所、署名、タイム スタンプによる整合性の確保、環境ごとに再ビルドするのではなくアーティファクトを活用、ビルドの不変性の確保）
- コードとしてのインフラストラクチャ（プル リクエストやバージョン管理の使用を含め、すべてのコードと同様に管理）

すべての組織は、アプリケーションまたはポートフォリオのライフサイクル全体にわたって、監視、トリアージ、およびアップデートや構成変更の適用に関する継続的な計画を確実にする必要があります。

攻撃シナリオの例

シナリオ #1: 信頼できるベンダーがマルウェアに感染し、システム アップデート時に自社のコンピュータ システムも感染してしまうケース。おそらく最も有名な例は以下のとおりです。

- 2019 年に発生した SolarWinds 社のシステム侵害事件。これにより、約 18,000 もの組織が被害を受けました。 https://www.npr.org/2021/04/16/985439655/a-worst-nightmare-cyberattack-the-untold-story-of-the-solarwinds-hack

シナリオ #2: 信頼できるベンダーが侵害され、特定の条件下でのみ悪意のある動作をするようになる。

- 2025 年に発生した Bybit の 15 億ドル相当の盗難事件は、標的のウォレットが使用されている場合にのみ実行されるウォレット ソフトウェアへのサプライチェーン攻撃によって引き起こされました。https://thehackernews.com/2025/02/bybit-hack-traced-to-safewallet-supply.html

シナリオ #3: 2025 年に発生した Shai-Hulud サプライチェーン攻撃 は、自己増殖型の npm ワームが初めて成功した事例でした。攻撃者は人気パッケージの悪意あるバージョンを仕込み、インストール後のスクリプトを用いて機密データを収集し、GitHub の公開リポジトリに流出させました。このマルウェアは被害者の環境内で npm トークンを検出し、アクセス可能なパッケージの悪意あるバージョンを自動的にプッシュします。ワームは npm によって阻止されるまでに 500 以上のパッケージ バージョンに感染しました。このサプライチェーン攻撃は高度で、急速に拡散し、甚大な被害をもたらしました。開発者のマシンを標的とすることで、開発者自身がサプライチェーン攻撃の主要な標的となっていることを実証しました。

シナリオ #4: 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、いずれかのコンポーネントに欠陥があると深刻な影響が生じる可能性があります。このような欠陥は、偶発的なもの（例：コーディング エラー）である場合もあれば、意図的なもの（例：コンポーネントに仕込まれたバックドア）である場合もあります。発見された悪用可能なコンポーネントの脆弱性の例を、以下にいくつか挙げます。

- Struts 2のリモート コード実行の脆弱性である CVE-2017-5638 は、サーバー上で任意のコードを実行できるため、重大な情報漏洩の原因となりました。
- Apache Log4j のリモート コード実行ゼロデイ脆弱性である CVE-2021-44228 (「Log4Shell」) は、ランサムウェア、暗号通貨マイニング、その他の攻撃キャンペーンの原因となりました。

参考情報

- OWASP Application Security Verification Standard: V15 Secure Coding and Architecture
- OWASP Cheat Sheet Series: Dependency Graph SBOM
- OWASP Cheat Sheet Series: Vulnerable Dependency Management
- OWASP Dependency-Track
- OWASP CycloneDX
- 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
- 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
- GitHub Advisory Database
- Ruby Libraries Security Advisory Database and Tools
- SAFECode Software Integrity Controls (PDF)
- Glassworm supply chain attack
- PhantomRaven supply chain attack campaign

対応する CWE の一覧

- CWE-447 Use of Obsolete Function
- CWE-1035 2017 Top 10 A9: Using Components with Known Vulnerabilities
- CWE-1104 Use of Unmaintained Third Party Components
- CWE-1329 Reliance on Component That is Not Updateable
- CWE-1395 Dependency on Vulnerable Third-Party Component

A04:2025 暗号化の失敗

背景

2 つ順位を下げて 4 位となったこの脆弱性は、暗号化の欠如、暗号化の強度不足、暗号鍵の漏洩、および関連するエラーに関連する不具合に焦点を当てています。このリスクにおける最も一般的な共通脆弱性リスト（CWE）のうち、脆弱な疑似乱数生成器の使用に関連するものは 3 つあります。*CWE-327* 破損または危険な暗号化アルゴリズムの使用、*CWE-331:* 不十分なエントロピー、*CWE-1241:* 乱数生成器における予測可能なアルゴリズムの使用、および *CWE-338* 暗号的に脆弱な疑似乱数生成器（*PRNG*）の使用です。

スコア表

対応する CWE 数	インシデント割合（最大）	インシデント割合（平均）	カバレッジ（最大）	カバレッジ（平均）	重み付き悪用可能性（最大）	重み付き悪用可能性（平均）	総発生件数	総 CVE 数
32	13.77%	3.80%	100.00%	47.74%	7.23	3.90	1,665,348	2,185

解説

一般的に、転送中のすべてのデータはトランスポート層（OSI 層 4）で暗号化する必要があります。CPU パフォーマンスや秘密鍵／証明書管理といった従来の課題は、暗号化を高速化するように設計された命令（例：AES サポート）を備えた CPU によって解決され、秘密鍵と証明書の管理は LetsEncrypt.org などのサービスによって簡素化されています。大手クラウド ベンダーは、それぞれのプラットフォーム向けに、より緊密に統合された証明書管理サービスを提供しています。

トランスポート層のセキュリティ確保に加えて、転送中（アプリケーション層、OSI 層 7）に追加の暗号化が必要なデータは何か？だけでなく、保存時に暗号化が必要なデータは何か？を判断することが重要です。例えば、パスワード、クレジット カード番号、健康記録、個人情報、企業秘密などは、EU の一般データ保護規則（GDPR）などのプライバシー法や PCI データ セキュリティ基準（PCI DSS）などの規制の対象となる場合は、特に、特別な保護が必要です。これらのデータはすべて、以下の点に注意する必要があります。

- デフォルトまたは古いコードで、古いあるいは脆弱な暗号アルゴリズムやプロトコルが使用されていますか？
- デフォルトの暗号鍵が使用されていますか？弱い暗号鍵が生成されていますか？鍵が再利用されていますか？あるいは、適切な鍵管理と変更が行われていませんか？
- 暗号鍵はソース コード リポジトリにチェックインされていますか？
- 暗号化が強制されていませんか？例えば、HTTP ヘッダー（ブラウザ）のセキュリティ ディレクティブやセキュリティ ヘッダーが欠落していませんか？
- 受信したサーバー証明書と信頼チェーンは適切に検証されていますか？
- 初期化ベクトルを無視したり再利用していますか？あるいは、暗号動作モードに対して十分にセキュアな方法で生成されていませんか？ECB などの安全でない動作モードが使用されていますか？認証付き暗号化の方が適切な場合に暗号化が使用されていますか？
- パスワード ベースの鍵導出関数がないにもかかわらず、パスワードが暗号鍵として使用されていますか？
- 暗号要件を満たすように設計されていないランダム性が暗号化目的で使用されていますか？正しい関数が選択されている場合でも、開発者がシードを設定する必要がありますか？必要でない場合、開発者は、組み込みの強力なシード機能を、十分なエントロピー/予測不可能性に欠けるシードで上書きしていませんか？
- MD5 や SHA1 などの非推奨のハッシュ関数が使用されていますか？あるいは、暗号ハッシュ関数が必要なのに非暗号ハッシュ関数が使用されていますか？
- 暗号エラー メッセージやサイド チャネル情報は、例えばパディング オラクル攻撃などの形で悪用される可能性がありますか？
- 暗号アルゴリズムはダウングレードまたは迂回される可能性がありますか？

参考情報「ASVS: 暗号化 (V11)、セキュアな通信 (V12)、およびデータ保護 (V14)」を参照してください。

防止方法

少なくとも以下を実施し、参考情報を参照します。

- アプリケーションによって処理、保存、または伝送されるデータを分類し、ラベル付けします。プライバシー法、規制要件、またはビジネス ニーズに応じて、機密性の高いデータを特定します。
- 最も機密性の高い鍵は、ハードウェアまたはクラウド ベースの HSM に保存します。
- 可能な限り、信頼性の高い暗号化アルゴリズムの実装を使用します。
- 機密データを不必要に保存してはなりません。できるだけ早く破棄するか、PCI DSS に準拠したトークン化、あるいはトランケーションを使用してください。保持されていないデータは盗難されることはありません。
- 保存中のすべての機密データは必ず暗号化します。
- 最新かつ強力な標準アルゴリズム、プロトコル、および鍵が確実に導入され、適切な鍵管理が採用されていることを確実にします。
- 伝送中のすべてのデータは、前方秘匿性 (FS) 暗号、サーバーによる暗号の優先順位付け、およびセキュアなパラメータを使用した TLS などのセキュアなプロトコルを使用して暗号化します。HTTP Strict Transport Security (HSTS) などのディレクティブを使用して暗号化を強制します。
- 機密データを含む応答のキャッシュを無効にします。これには、CDN、Web サーバー、およびあらゆるアプリケーション キャッシュ (例: Redis) におけるキャッシュが含まれます。

- データ分類に応じて必要なセキュリテ管理策を適用します。
- FTP や STARTTLS などの暗号化されていないプロトコルは使用してはなりません。機密データの送信には SMTP の使用は避けます。
- パスワードは、Argon2、yescrypt、scrypt、PBKDF2-HMAC-SHA-256 など、ワーク ファクター (遅延係数) を持つ強力な適応型ソルト付きハッシュ関数を使用して保存します。bcrypt を使用しているレガシー システムについては、OWASP OWASP Cheat Sheet: Password Storage で詳細なアドバイスを参照できます。
- 初期化ベクトルは、動作モードに適したものを選択する必要があります。これには、CSPRNG (暗号的にセキュアな擬似乱数生成器) の使用が含まれます。ナンスを必要とするモードでは、初期化ベクトル (IV) に CSPRNG は必要ありません。いかなる場合でも、固定キーに対して IV を 2 回使用してはなりません。
- 単なる暗号化ではなく、常に認証付き暗号化を使用します。
- 鍵は暗号学的にランダムに生成し、バイト配列としてメモリに保存する必要があります。パスワードを使用する場合は、適切なパスワード ベースの鍵導出関数を使用して鍵に変換する必要があります。
- 暗号学的ランダム性が適切な場合に使用され、予測可能な方法や低いエントロピーでシードされていないことを確実にします。ほとんどの最新の API では、開発者が CSPRNG のシードをセキュアに使用する必要がありません。
- MD5、SHA1、暗号ブロック連鎖モード（CBC）、PKCS #1 v1.5などといった非推奨の暗号化関数、ブロック構築方法、パディング スキームは使用してはなりません。
- 設定と構成がセキュリティ要件を満たしていることを確実にするために、セキュリティ専門家、この目的のために設計されたツール、またはその両方によるレビューを受けます。
- 高リスク システムが 2030 年末までに安全になるように、今すぐ耐量子暗号 (PQC) の準備をする必要があります（参考文献 (ENISA) を参照）。

攻撃シナリオの例

シナリオ #1: サイトがすべてのページで TLS を使用または強制していないか、弱い暗号化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（例: セキュアでない無線ネットワーク）を監視し、接続を HTTPS から HTTP にダウングレードし、リクエストを傍受してユーザーのセッション Cookie を盗みます。その後、攻撃者はこの Cookie をリプレイし、ユーザーの（認証済み）セッションを乗っ取り、ユーザーの個人データにアクセスまたは変更します。上記の他に、送金の受取者など、伝送されるすべてのデータを改ざんすることも可能です。

シナリオ #2: パスワード データベースでは、ソルトなしまたは単純なハッシュを使用してすべてのパスワードが保存されています。ファイル アップロードの脆弱性により、攻撃者はパスワード データベースを取得できます。ソルトなしのハッシュはすべて、事前に計算されたハッシュのレインボー テーブルによって意図せず公開される可能性があります。単純または高速なハッシュ関数によって生成されたハッシュは、ソルト付きであっても GPU によって解読される可能性があります。

参考情報

- OWASP Proactive Controls: C2: Use Cryptography to Protect Data
- OWASP Application Security Verification Standard (ASVS): V11, 12, 14
- OWASP Cheat Sheet: Transport Layer Protection
- OWASP Cheat Sheet: User Privacy Protection
- OWASP Cheat Sheet: Password Storage
- OWASP Cheat Sheet: Cryptographic Storage
- OWASP Cheat Sheet: HSTS
- OWASP Testing Guide: Testing for weak cryptography
- ENISA: A Coordinated Implementation Roadmap for the Transition to Post-Quantum Cryptography
- NIST Releases First 3 Finalized Post-Quantum Encryption Standards

対応する CWE の一覧

- CWE-261 Weak Encoding for Password
- CWE-296 Improper Following of a Certificate's Chain of Trust
- CWE-319 Cleartext Transmission of Sensitive Information
- CWE-320 Key Management Errors (Prohibited)
- CWE-321 Use of Hard-coded Cryptographic Key
- CWE-322 Key Exchange without Entity Authentication
- CWE-323 Reusing a Nonce, Key Pair in Encryption
- CWE-324 Use of a Key Past its Expiration Date
- CWE-325 Missing Required Cryptographic Step
- CWE-326 Inadequate Encryption Strength
- CWE-327 Use of a Broken or Risky Cryptographic Algorithm
- CWE-328 Reversible One-Way Hash

- CWE-329 Not Using a Random IV with CBC Mode
- CWE-330 Use of Insufficiently Random Values
- CWE-331 Insufficient Entropy
- CWE-332 Insufficient Entropy in PRNG
- CWE-334 Small Space of Random Values
- CWE-335 Incorrect Usage of Seeds in Pseudo-Random Number Generator(PRNG)
- CWE-336 Same Seed in Pseudo-Random Number Generator (PRNG)
- CWE-337 Predictable Seed in Pseudo-Random Number Generator (PRNG)
- CWE-338 Use of Cryptographically Weak Pseudo-Random Number Generator(PRNG)
- CWE-340 Generation of Predictable Numbers or Identifiers
- CWE-342 Predictable Exact Value from Previous Values
- CWE-347 Improper Verification of Cryptographic Signature
- CWE-523 Unprotected Transport of Credentials
- CWE-757 Selection of Less-Secure Algorithm During Negotiation('Algorithm Downgrade')
- CWE-759 Use of a One-Way Hash without a Salt
- CWE-760 Use of a One-Way Hash with a Predictable Salt
- CWE-780 Use of RSA Algorithm without OAEP
- CWE-916 Use of Password Hash With Insufficient Computational Effort
- CWE-1240 Use of a Cryptographic Primitive with a Risky Implementation
- CWE-1241 Use of Predictable Algorithm in Random Number Generator

A05:2025 インジェクション

背景

インジェクションはランキングで 3 位から 5 位に 2 つ順位を落としましたが、A04:2025-暗号化の失敗 および A06:2025-セキュリティが確保されていない設計 との比較では順位を維持しています。インジェクションは最も多くのテストが実施されているカテゴリの 1 つであり、アプリケーションの 100% が何らかのインジェクションのテストを受けています。このカテゴリには 37 個の CWE があり、全カテゴリ中で最も多くの CVE 数を記録しています。インジェクションには、3 万件を超える CVE を持つクロス サイト スクリプティング（高頻度/低影響）と、1 万 4 千件を超える CVE を持つ SQL インジェクション（低頻度/高影響）が含まれます。*CWE-79 Web* ページ生成における入力の不適切な中和」（「クロス サイト スクリプティング」） の CVE が大量に報告されたことが、このカテゴリの平均的な加重影響度を低下させています。

スコア表

対応する CWE 数	インシデント割合（最大）	インシデント割合（平均）	カバレッジ（最大）	カバレッジ（平均）	重み付き悪用可能性（最大）	重み付き悪用可能性（平均）	総発生件数	総 CVE 数
37	13.77%	3.08%	100.00%	42.93%	7.15	4.32	1,404,249	62,445

解説

インジェクション脆弱性とは、攻撃者がプログラムの入力フィールドに悪意のあるコードやコマンド（SQL やシェル コードなど）を挿入し、あたかもシステムの一部であるかのようにシステムを騙して実行させることを可能にするシステム欠陥です。これは、非常に深刻な結果をもたらす可能性があります。

アプリケーションが攻撃に対して脆弱な状況とは、以下のような状況を指します。

- ユーザーが入力したデータは、アプリケーションによって検証、フィルタリング、または無害化されません。
- 動的クエリまたはコンテキスト対応エスケープのない非パラメータ化呼び出しが、インタープリタで直接使用されます。
- 無害化されていないデータは、追加の機密レコードを抽出するために、オブジェクト リレーショナル マッピング (ORM) 検索パラメータ内で使用されます。。
- 潜在的に悪意のあるデータが直接使用または連結されます。SQL またはコマンドには、動的クエリ、コマンド、またはストアド プロシージャの構造と悪意のあるデータが含まれています。

一般的なインジェクションには、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、式言語 (EL) またはオブジェクト グラフ ナビゲーション ライブラリ (OGNL) インジェクションなどがあります。この概念はすべてのインタープリタで同一です。すべてのパラメータ、ヘッダー、URL、Cookie、JSON、SOAP、および XML データ入力のソース コード レビューと自動テスト（ファジングを含む）を組み合わせることで、最も効果的に検出できます。静的（SAST）、動的（DAST）、および対話

型（IAST）アプリケーション セキュリティ テスト ツールを CI/CD パイプラインに追加することでも、本番環境へのデプロイ前にインジェクション脆弱性を特定するのに役立ちます。

LLM では、関連する種類のインジェクション脆弱性が一般的になっています。これらについては、OWASP LLM Top 10、特に LLM01:2025 Prompt Injection で個別に説明されています。

防止方法

インジェクションを防ぐ最善の方法は、データをコマンドやクエリから分離することです。

- 推奨される選択肢は、安全な API を使用することです。これにより、インタープリタの使用を完全に回避したり、パラメータ化されたインターフェースを提供したり、オブジェクト リレーショナル マッピング ツール（ORM）に移行したりします。**注:** パラメータ化されている場合でも、PL/SQL または T-SQL がクエリとデータを連結したり、EXECUTE IMMEDIATE または exec() を使用して悪意のあるデータを実行したりすると、ストアド プロシージャによって SQL インジェクションが発生する可能性があります。

データとコマンドを分離できない場合は、以下の手法を用いて脅威を軽減できます。

- サーバー側での入力検証を実施します。テキスト エリアやモバイル アプリケーションの API など、多くのアプリケーションでは特殊文字が必要となるため、これは完全な防御策ではありません。
- 残りの動的クエリについては、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します。**注:** テーブル名、列名などの SQL 構造はエスケープできないため、ユーザーが指定した構造名は危険です。これはレポート作成ソフトウェアでよく見られる問題です。

警告: これらの手法では、複雑な文字列を解析してエスケープする必要があるため、エラーが発生しやすくなり、基盤となるシステムに小さな変更が加えられた場合に堅牢性が損なわれます。

攻撃シナリオの例

シナリオ #1: アプリケーションは、以下の脆弱な SQL 呼び出しの構築に信頼できないデータを使用します。

```
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目的に信頼すると、依然として脆弱なクエリが発生する可能性があります (例: ハイバネート クエリ言語 (HQL))。

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

どちらの場合も、攻撃者は、ブラウザの `id` パラメータの値を `' UNION SLEEP(10);--` に変更して送信します。例えば、

```
http://example.com/app/accountView?id=' UNION SELECT SLEEP(10);--
```

と変更すると、両方のクエリの意味が変わり、accounts テーブルのすべてのレコードが返されるようになります。より危険な攻撃では、データが変更または削除されたり、ストアド プロシージャが呼び出されたりする可能性があります。

シナリオ 3: アプリケーションがユーザー入力を OS コマンドに直接渡します。

```
String cmd = "nslookup " + request.getParameter("domain");
Runtime.getRuntime().exec(cmd);
```

攻撃者は、`example.com; cat /etc/passwd` と入力して、サーバー上で任意のコマンドを実行します。

参考情報

- OWASP Proactive Controls: Secure Database Access
- OWASP ASVS: V5 Input Validation and Encoding
- OWASP Testing Guide: SQL Injection, Command Injection, and ORM Injection
- OWASP Cheat Sheet: Injection Prevention
- OWASP Cheat Sheet: SQL Injection Prevention
- OWASP Cheat Sheet: Injection Prevention in Java
- OWASP Cheat Sheet: Query Parameterization
- OWASP Automated Threats to Web Applications – OAT-014
- PortSwigger: Server-side template injection
- Awesome Fuzzing: a list of fuzzing resources

対応する CWE の一覧

- CWE-20 Improper Input Validation
- CWE-74 Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')
- CWE-76 Improper Neutralization of Equivalent Special Elements
- CWE-77 Improper Neutralization of Special Elements used in a Command ('Command Injection')
- CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
- CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
- CWE-80 Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)
- CWE-83 Improper Neutralization of Script in Attributes in a Web Page
- CWE-86 Improper Neutralization of Invalid Characters in Identifiers in Web Pages
- CWE-88 Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')
- CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
- CWE-90 Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')
- CWE-91 XML Injection (aka Blind XPath Injection)
- CWE-93 Improper Neutralization of CRLF Sequences ('CRLF Injection')
- CWE-94 Improper Control of Generation of Code ('Code Injection')
- CWE-95 Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')
- CWE-96 Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')
- CWE-97 Improper Neutralization of Server-Side Includes (SSI) Within a Web Page
- CWE-98 Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')
- CWE-99 Improper Control of Resource Identifiers ('Resource Injection')
- CWE-103 Struts: Incomplete validate() Method Definition
- CWE-104 Struts: Form Bean Does Not Extend Validation Class
- CWE-112 Missing XML Validation
- CWE-113 Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')
- CWE-114 Process Control
- CWE-115 Misinterpretation of Output
- CWE-116 Improper Encoding or Escaping of Output
- CWE-129 Improper Validation of Array Index
- CWE-159 Improper Handling of Invalid Use of Special Elements
- CWE-470 Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')
- CWE-493 Critical Public Variable Without Final Modifier
- CWE-500 Public Static Field Not Marked Final
- CWE-564 SQL Injection: Hibernate
- CWE-610 Externally Controlled Reference to a Resource in Another Sphere
- CWE-643 Improper Neutralization of Data within XPath Expressions ('XPath Injection')
- CWE-644 Improper Neutralization of HTTP Headers for Scripting Syntax
- CWE-917 Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')

A06:2025 セキュリティが確保されていない設計

背景

セキュリティが確保されていない設計は、**A02:2025-セキュリティ設定のミス** と **A03:2025-ソフトウェア サプライチェーンの失敗** に追い抜かれたため、ランキングで 4 位から 6 位に 2 つ順位を落としました。このカテゴリは 2021 年に導入され、脅威モデリングとセキュアな設計の重視に関して、業界では目立った改善が見られました。このカテゴリは、設計とアーキテクチャの欠陥に関連するリスクに焦点を当てており、脅威モデリング、セキュアな設計パターン、およびリファレンス アーキテクチャのより多くの使用を求めています。これには、アプリケーションのビジネス ロジックの欠陥、例えば、アプリケーション内の不要または予期しない状態変更の定義の欠如が含まれます。コミュニティとして、コーディング領域における「シフト レフト」を超えて、要件定義やアプリケーション設計といった、セキュア・バイ・デザインの原則に不可欠な事前コーディング活動へと進む必要があります（例: **現代的なアプリケーション セキュリティ プログラムの確立: 計画・設計フェーズ** を参照）。注目すべき共通脆弱性リスト（CWE）には、*CWE-256:* 保護されていない資格情報の保管、*CWE-269:* 不適切な権限管理、*CWE-434:* 危険な種類のファイルの無制限のアップロード、*CWE-501:* 信頼境界違反、および *CWE-522:* 十分に保護されていない資格情報 が含まれます。

スコア表

対応する CWE 数	インシデント割合（最大）	インシデント割合（平均）	カバレッジ（最大）	カバレッジ（平均）	重み付き悪用可能性（最大）	重み付き悪用可能性（平均）	総発生件数	総 CVE 数
39	22.18%	1.86%	88.76%	35.18%	6.96	4.05	729,882	7,647

解説

セキュリティが確保されていない設計は、様々な弱点を表す広範なカテゴリーであり、「管理策設計の欠如、または効果のない管理策設計」と表現されます。セキュリティが確保されていない設計は、他の Top 10 リスク カテゴリすべてに当てはまるわけではありません。セキュリティが確保されていない設計とセキュリティが確保されていない実装には違いがあることに注意してください。設計上の欠陥と実装上の欠陥を区別するのには理由があります。それらは根本原因が異なり、開発プロセスの異なる時期に発生し、修正方法も異なるからです。セキュリティが確保された設計であっても、実装上の欠陥が悪用される可能性のある脆弱性につながる可能性があります。セキュリティが確保されていない設計は、特定の攻撃に対する防御のために必要なセキュリティ管理策が策定されていないため、完璧な実装によって修正することはできません。セキュリティが確保されていない設計につながる要因の 1 つは、開発中のソフトウェアまたはシステムに内在するビジネス リスク プロファイリングの欠如であり、その結果、必要なセキュリティ設計レベルを判断できないことです。

セキュリティが確保された設計を実現するための 3 つの重要な要素は以下のとおりです。

- 要件の収集とリソース管理
- セキュリティが確保された設計の作成
- セキュアな開発ライフサイクルの確立

要件とリソース管理

アプリケーションのビジネス要件を収集し、ビジネス部門と交渉します。これには、すべてのデータ資産と想定されるビジネス ロジックの機密性、完全性、可用性、真正性に関する保護要件が含まれます。アプリケーションの公開範囲と、（アクセス制御に必要な範囲を超えて）テナントの分離が必要かどうかを考慮します。機能的および非機能的なセキュリティ要件を含む技術要件をまとめます。セキュリティ活動を含む、すべての設計、構築、テスト、運用をカバーする予算を計画し、交渉します。

セキュリティが確保された設計

セキュリティが確保された設計とは、脅威を継続的に評価し、既知の攻撃方法を防ぐためにコードが堅牢に設計・テストされていることを保証する文化と方法論です。脅威モデリングは、精緻化セッション（または類似の活動）に統合し、データフロー、アクセス制御、その他のセキュリティ管理策における変更点を探る必要があります。ユーザー ストーリーの開発では、正しいフローと障害状態を特定し、責任者と影響を受ける関係者がそれらを十分に理解し、合意していることを確認します。想定されるフローと失敗フローの前提条件と条件を分析し、それらが正確かつ望ましい状態であることを確実にします。前提条件を検証し、適切な動作に必要な条件を適用する方法を決定します。結果がユーザー ストーリーに文書化されていることを確実にします。失敗から学び、改善を促進するための積極的なインセンティブを提供します。セキュリティが確保された設計は、ソフトウェアに追加できるアドオンでもツールでもありません。

セキュアな開発ライフサイクル

セキュアなソフトウェアには、セキュアな開発ライフサイクル、セキュアな設計パターン、確立された方法論、セキュアなコンポーネント ライブラリ、適切なツール、脅威モデリング、そしてプロセス改善に活用されるインシデント事後分析が必要です。ソフトウェア プロジェクトの開始時、プロジェクト全体、そして継続的なソフトウェア維持において、セキュリティ専門家に相談します。セキュアなソフトウェア開発の取り組みを構築するには、OWASP Software Assurance Maturity Model (SAMM) の活用を検討します。

開発者の自己責任は、しばしば過小評価されます。意識、責任、そして積極的なリスク軽減の文化を育みます。セキュリティに関する定期的な意見交換（脅威モデリング セッションなど）は、すべての重要な設計決定にセキュリティを組み込むという考え方を醸成するのに役立ちます。

防止方法

- アプリケーション セキュリティ専門家と連携し、セキュリティとプライバシー関連の管理策の評価と設計を支援するセキュアな開発ライフサイクルを確立・運用します。
- セキュアな設計パターンまたは既成コンポーネントのライブラリを確立・運用します。
- 認証、アクセス制御、ビジネス ロジック、主要なフローなど、アプリケーションの重要な部分に脅威モデリングを適用します。
- セキュリティ マインドセットを育む教育ツールとして脅威モデリングを利用します・
- セキュリティ言語と管理策をユーザー ストーリーに統合します。
- アプリケーションの各層（フロント エンドからバック エンドまで）に妥当性チェックを統合します。
- すべての重要なフローが脅威モデルに耐性があることを検証するための単体テストと統合テストを作成します。アプリケーションの各層の利用事例と誤用ケースをまとめます。
- 露出状況と保護のニーズに応じて、システム層とネットワーク層を分離します。
- すべての層を通して、テナントを設計によって堅牢に分離します。

攻撃シナリオの例

シナリオ #1: 資格情報の復旧ワークフローには「質問と回答」が含まれる場合がありますが、これは NIST 800-63b、OWASP ASVS、OWASP Top 10 で禁止されています。質問と回答は、複数の人が回答を知ることができるため、本人確認の証拠として信頼できません。このような機能は削除し、よりセキュアな設計に置き換える必要があります。

シナリオ #2: ある映画館チェーンでは、団体予約割引を設けており、最大 15 名まで予約金を要求しています。攻撃者は、このフローを脅威モデル化し、アプリケーションのビジネス ロジックに攻撃ベクトルを見つけられるかどうかをテストできます。例えば、数回のリクエストで 600 席の座席とすべての映画館を一括予約し、莫大な収益損失を引き起こすといった攻撃です。

シナリオ #3: ある小売チェーンの e コマース ウェブ サイトには、高級ビデオカードを購入しオークション サイトで転売する転売業者が運営するボットに対する保護対策が施されていません。これは、ビデオカード メーカーと小売チェーンのオーナーにとって悪評を招き、どんなに高くてもこれらのカードを入手できない愛好家との確執を招いてしまいます。慎重なアンチ ボット設計と、入手可能になってから数秒以内に購入を行うといったドメイン ロジック ルールを整備することで、不正な購入を識別し、そのような取引を拒否できる可能性があります。

参考情報

- OWASP Cheat Sheet: Secure Design Principles
- OWASP SAMM: Design | Secure Architecture
- OWASP SAMM: Design | Threat Assessment
- NIST – Guidelines on Minimum Standards for Developer Verification of Software
- The Threat Modeling Manifesto
- Awesome Threat Modeling

対応する CWE の一覧

- CWE-73 External Control of File Name or Path
- CWE-183 Permissive List of Allowed Inputs
- CWE-256 Unprotected Storage of Credentials
- CWE-266 Incorrect Privilege Assignment
- CWE-269 Improper Privilege Management
- CWE-286 Incorrect User Management
- CWE-311 Missing Encryption of Sensitive Data
- CWE-312 Cleartext Storage of Sensitive Information
- CWE-313 Cleartext Storage in a File or on Disk
- CWE-316 Cleartext Storage of Sensitive Information in Memory
- CWE-362 Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')
- CWE-382 J2EE Bad Practices: Use of System.exit()
- CWE-419 Unprotected Primary Channel
- CWE-434 Unrestricted Upload of File with Dangerous Type
- CWE-436 Interpretation Conflict
- CWE-444 Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')
- CWE-451 User Interface (UI) Misrepresentation of Critical Information
- CWE-454 External Initialization of Trusted Variables or Data Stores
- CWE-472 External Control of Assumed-Immutable Web Parameter
- CWE-501 Trust Boundary Violation
- CWE-522 Insufficiently Protected Credentials
- CWE-525 Use of Web Browser Cache Containing Sensitive Information
- CWE-539 Use of Persistent Cookies Containing Sensitive Information
- CWE-598 Use of GET Request Method With Sensitive Query Strings
- CWE-602 Client-Side Enforcement of Server-Side Security
- CWE-628 Function Call with Incorrectly Specified Arguments
- CWE-642 External Control of Critical State Data
- CWE-646 Reliance on File Name or Extension of Externally-Supplied File
- CWE-653 Insufficient Compartmentalization
- CWE-656 Reliance on Security Through Obscurity

- CWE-657 Violation of Secure Design Principles
- CWE-676 Use of Potentially Dangerous Function
- CWE-693 Protection Mechanism Failure
- CWE-799 Improper Control of Interaction Frequency
- CWE-807 Reliance on Untrusted Inputs in a Security Decision
- CWE-841 Improper Enforcement of Behavioral Workflow
- CWE-1021 Improper Restriction of Rendered UI Layers or Frames
- CWE-1022 Use of Web Link to Untrusted Target with window.opener Access
- CWE-1125 Excessive Attack Surface

A07:2025 認証の失敗

背景

認証の失敗は、このカテゴリに含まれる 36 個の CWE をより正確に反映するためにわずかに名称を変更したものの、7 位を維持しました。標準化されたフレームワークの恩恵を受けているにもかかわらず、このカテゴリは 2021 年から 7 位を維持しています。注目すべき CWE には、*CWE-259* ハード コードされたパスワードの使用、*CWE-297:* ホスト不一致による証明書の不適切な検証、*CWE-287:* 不適切な認証、*CWE-384:* セッション固定、*CWE-798* ハード コードされた資格情報の使用 が含まれます。

スコア表

対応する CWE 数	インシデント割合（最大）	インシデント割合（平均）	カバレッジ（最大）	カバレッジ（平均）	重み付き悪用可能性（最大）	重み付き悪用可能性（平均）	総発生件数	総 CVE 数
36	15.80%	2.92%	100.00%	37.14%	7.69	4.44	1,120,673	7,147

解説

攻撃者がシステムを騙して無効なユーザーや不正なユーザーを正当なユーザーとして認識させてしまう場合、この脆弱性が存在することになります。アプリケーションが以下の条件に該当する場合、認証に脆弱性が存在する可能性があります。

- 攻撃者が漏洩した有効なユーザー名とパスワードのリストを悪用し、クレデンシャル スタッフィングなどの自動攻撃を許します。近年、この種類の攻撃はハイブリッド パスワード攻撃、クレデンシャル スタッフィング（パスワード スプレー攻撃とも呼ばれます）へと拡張され、攻撃者は漏洩した認証情報のバリエーションや増分（例えば、Password1!、Password2!、Password3! など）を使用してアクセス権限を奪取しようとします。
- 総当たり攻撃やその他の自動化されたスクリプト攻撃を許しますが、これらはすぐにはブロックされません。
- デフォルトのパスワード、脆弱なパスワード、またはよく知られているパスワード（例えば、「Password1」や、「admin」というユーザー名に「admin」というパスワード）を許します。
- すでに侵害されたことがわかっている資格情報を使用して、ユーザーが新規アカウントを作成できます。
- 安全を確保できない「知識ベースの回答」などの、脆弱または効果のない認証情報の復旧およびパスワード失念時のプロセスの使用を許可します。
- 平文、または弱い暗号化やハッシュ化されたパスワード データ ストアを使用しています（A04:2025-Cryptographic Failures を参照）。
- 多要素認証が欠落しているか、効果がありません。
- 多要素認証が利用できない場合、脆弱または効果のないフォール バックの使用を許可しています。
- セッション識別子が、URL、隠しフィールド、またはクライアントがアクセスできるその他の安全でない場所に公開されています。
- ログイン成功後に同じセッション識別子が再利用されています。
- ログアウト中または非アクティブ期間中に、ユーザー セッションまたは認証トークン（主にシングル サインオン (SSO) トークン）が正しく無効化されていません。
- 提供された資格情報の範囲と対象ユーザーを正しくアサートしません。

防止方法

- 可能な場合は、多要素認証を導入しその使用を強制することで、自動クレデンシャル スタッフィング、総当たり攻撃、盗難された認証情報の再利用攻撃を防止します。
- 可能な場合は、パスワード マネージャーの使用を推奨し、ユーザーがより適切な選択を行えるようにします。
- 特に管理者ユーザーについては、デフォルトの認証情報で出荷またはデプロイしてはなりません。
- 新規または変更したパスワードを、最悪のパスワード上位 10,000 件のリストと比較するなど、脆弱なパスワードに対するチェックを実装します。
- 新規アカウントの作成時およびパスワードの変更時には、既知の侵害された認証情報のリストと照合します（例: haveibeenpwned.com を使用）。

- パスワードの長さ、複雑さ、および変更に関するポリシーを、記憶シークレットまたはその他の最新の証拠に基づくパスワード ポリシーに関する National Institute of Standards and Technology (NIST) 800-63b's guidelines in section 5.1.1 に準拠させます。
- 侵害の疑いがない限り、パスワードの変更を強制してはなりません。侵害の疑いがある場合は、直ちにパスワードのリセットを強制します。
- すべての結果に同じメッセージ（「ユーザー名またはパスワードが無効です」）を使用することで、登録、■情報の復旧、および API パスウェイが、アカウント列挙攻撃に対して堅牢化されていることを確実にします。
- ログイン失敗回数を制限するか、遅延を増やします。ただし、サービス拒否（DoS）シナリオを作成しないよう注意します。クレデンシャル スタッフィング、総当たり攻撃、その他の攻撃が検知または疑われる場合は、すべての失敗を記録し、管理者に警告します。
- ログイン後に高エントロピーの新しいランダム セッション ID を生成するような、サーバー側のセキュアな組み込みセッション マネージャーを使用します。セッション ID は URL に含めず、セキュアな Cookie に安全に保存し、ログアウト、アイドル タイムアウト、および絶対タイム アウト後に無効にする必要があります。
- 認証、ID、およびセッション管理には、既成の信頼性の高いシステムを使用するのが理想的です。可能な限り、堅牢化され十分にテストされたシステムを購入して活用することで、このリスクを移転します。
- 提供された資格情報の使用目的を検証します（例: JWTの場合、`aud`、`iss` クレーム、スコープを確認します）。

攻撃シナリオの例

シナリオ #1: 既知のユーザー名とパスワードの組み合わせリストを利用するクレデンシャル スタッフィングは、今や非常に一般的な攻撃です。最近では、攻撃者が人間の一般的な行動に基づいてパスワードを「増分」したり、調整したりすることが判明しています。例えば、「Winter2025」を「Winter2026」に、「ILoveMyDog6」を「ILoveMyDog7」や「ILoveMyDog5」に変更するなどです。このようなパスワード試行の調整は、ハイブリッド クレデンシャル スタッフィング攻撃またはパスワード スプレー攻撃と呼ばれ、従来の攻撃よりもさらに効果的です。アプリケーションが自動化された脅威（総当たり、スクリプト、ボット）やクレデンシャル スタッフィングに対する防御を実装していない場合、そのアプリケーションはパスワード オラクルとして利用され、認証情報の有効性を判断し、不正アクセスを許す可能性があります。

シナリオ #2: 認証攻撃の成功の多くは、パスワードを唯一の認証要素として使い続けることに起因しています。かつてはベスト プラクティスと考えられていたパスワードの変更と複雑さの要件は、ユーザーにパスワードの再利用と脆弱なパスワードの使用を促します。組織は、NIST 800-63 に従ってこれらの慣行を中止し、すべての重要なシステムで多要素認証の使用を強制することが推奨されます。

シナリオ #3: アプリケーションのセッション タイムアウトが正しく実装されていません。あるユーザーは、公共のコンピュータを使用してアプリケーションにアクセスし、「ログアウト」を選択する代わりにブラウザ タブを閉じて立ち去ってしまいます。また、シングル サインオン (SSO) セッションをシングル ログアウト (SLO) で終了できない場合も、この問題の好例です。つまり、1 回のログインで、例えばメール リーダー、文書管理システム、チャット システムなどにログインできますが、ログアウトは現在のシステムからのみ行われます。被害者がログアウトに成功したと思っても、一部のアプリケーションへの認証が残っている状態で攻撃者が同じブラウザを使用すると、被害者のアカウントにアクセスできます。オフィスや企業では、機密性の高いアプリケーションが正しく終了されておらず、同僚がロック解除されたコンピュータに（一時的に）アクセスできる場合にも、同様の問題が発生する可能性があります。

参考情報

- OWASP Authentication Cheat Sheet
- OWASP Secure Coding Practices

対応する CWE の一覧

- CWE-258 Empty Password in Configuration File
- CWE-259 Use of Hard-coded Password
- CWE-287 Improper Authentication
- CWE-288 Authentication Bypass Using an Alternate Path or Channel
- CWE-289 Authentication Bypass by Alternate Name
- CWE-290 Authentication Bypass by Spoofing
- CWE-291 Reliance on IP Address for Authentication
- CWE-293 Using Referer Field for Authentication
- CWE-294 Authentication Bypass by Capture-replay
- CWE-295 Improper Certificate Validation
- CWE-297 Improper Validation of Certificate with Host Mismatch
- CWE-298 Improper Validation of Certificate with Host Mismatch
- CWE-299 Improper Validation of Certificate with Host Mismatch
- CWE-300 Channel Accessible by Non-Endpoint
- CWE-302 Authentication Bypass by Assumed-Immutable Data
- CWE-303 Incorrect Implementation of Authentication Algorithm
- CWE-304 Missing Critical Step in Authentication
- CWE-305 Authentication Bypass by Primary Weakness

- CWE-306 Missing Authentication for Critical Function
- CWE-307 Improper Restriction of Excessive Authentication Attempts
- CWE-308 Use of Single-factor Authentication
- CWE-309 Use of Password System for Primary Authentication
- CWE-346 Origin Validation Error
- CWE-350 Reliance on Reverse DNS Resolution for a Security-Critical Action
- CWE-384 Session Fixation
- CWE-521 Weak Password Requirements
- CWE-613 Insufficient Session Expiration
- CWE-620 Unverified Password Change
- CWE-640 Weak Password Recovery Mechanism for Forgotten Password
- CWE-798 Use of Hard-coded Credentials
- CWE-940 Improper Verification of Source of a Communication Channel
- CWE-941 Incorrectly Specified Destination in a Communication Channel
- CWE-1390 Weak Authentication
- CWE-1391 Use of Weak Credentials
- CWE-1392 Use of Default Credentials
- CWE-1393 Use of Default Password

A08:2025 ソフトウェアまたはデータの整合性の不具合

背景

ソフトウェアまたはデータの整合性の不具合は、引き続き 8 位にランクインしています。「ソフトウェア および データの整合性の不具合」という名称から、やや分かりやすく名称が変更されています。このカテゴリは、A03:2025-ソフトウェア サプライチェーン の失敗よりも低いレベルで、信頼境界の維持とソフトウェア、コード、およびデータ アーティファクトの整合性検証の失敗に焦点を当てています。このカテゴリは、整合性を検証することなく、ソフトウェアの更新や重要なデータに関する憶測を行うことに焦点を当てています。注目すべき共通脆弱性リスト（CWE）には、*CWE-829:* 信頼されていない制御範囲からの機能の取り込み、*CWE-915:* 動的に決定されたオブジェクト属性の不適切な制御による変更、*CWE-502:* 信頼されていないデータの逆シリアル化 などがあります。

スコア表

対応する CWE 数	インシデント割合（最大）	インシデント割合（平均）	カバレッジ（最大）	カバレッジ（平均）	重み付き悪用可能性（最大）	重み付き悪用可能性（平均）	総発生件数	総 CVE 数
14	8.98%	2.75%	78.52%	45.49%	7.11	4.79	501,327	3,331

解説

ソフトウェアまたはデータの整合性の不具合は、無効または信頼できないコードやデータが信頼できる有効なものとして扱われることに対する保護対策を講じていないコードやインフラストラクチャに関連しています。例えば、アプリケーションが、信頼できないソース、リポジトリ、コンテンツ配信ネットワーク（CDN）からのプラグイン、ライブラリ、またはモジュールに依存している場合などが挙げられます。ソフトウェアの整合性チェックを利用・提供しないセキュアでない CI/CDパイプラインは、不正アクセス、セキュアでないコードや悪意のあるコード、あるいはシステム侵害の危険性をはらんでいます。別の例として、信頼できない場所からコードやアーティファクトをプルしたり、使用前に検証（署名の確認など）を行わない CI/CD が挙げられます。さらに、多くのアプリケーションには自動更新機能が搭載されており、十分な整合性検証を行わずに更新がダウンロードされ、以前は信頼されていたアプリケーションに適用されます。攻撃者は、独自の更新をアップロードし、それをすべてのインストール環境で配布・実行させる可能性があります。また、オブジェクトやデータが攻撃者が参照・変更できる構造にエンコードまたはシリアル化されている場合セキュアでない逆シリアル化に対して脆弱です。

防止方法

- デジタル署名または同様の仕組みを使用して、ソフトウェアまたはデータが期待されるソースからのものであり、変更されていないことを検証します。
- npm や Maven などのライブラリと依存関係が、信頼できるリポジトリのみを使用していることを確実にします。リスク プロファイルが高い場合は、内部で検証済みで信頼性のあるリポジトリをホストすることを検討します。
- 悪意のあるコードや構成がソフトウェア パイプラインに持ち込まれる可能性を最小限に抑えるため、コードと構成の変更に対するレビュー プロセスが確立されていることを確実にします。
- CI/CD パイプラインに適切な分離、構成、アクセス制御が備わっていることを確実にし、ビルドおよびデプロイ プロセスを流れるコードの整合性を確保します。

- 署名なしまたは暗号化されていないシリアル化データを信頼できないクライアントから受信していないことを確実にし、シリアル化データの改ざんやリプレイを検出するための何らかの整合性チェックやデジタル署名が行われずに使用されないようにします。

攻撃シナリオの例

シナリオ #1 信頼できないソースの Web 機能のの取り込み: ある企業は、サポート機能を提供するために外部サービス プロバイダーを利用しています。便宜上、`myCompany.SupportProvider.com` から `support.myCompany.com` への DNS マッピングが設定されています。つまり、`myCompany.com` ドメインに設定されたすべての Cookie（認証 Cookie を含む）がサポート プロバイダーに送信されることになります。サポート プロバイダーのインフラストラクチャにアクセスできるユーザーは誰でも、`support.myCompany.com` にアクセスしたすべてのユーザーの Cookie を盗み、セッション ハイジャック攻撃を実行することができます。

シナリオ #2 署名なしの更新: 多くの家庭用ルーター、セットトップ ボックス、デバイスのファームウェアなどは、署名されたファームウェアによるアップデートの検証を行っていません。署名のないファームウェアは攻撃者の標的として増加しており、今後さらに悪化すると予想されています。多くの場合、将来のバージョンで修正され以前のバージョンが期限切れになるまで待つ以外に対策手段がないため、これは大きな懸念事項です。

シナリオ #3 信頼できないソースからのパッケージの使用: 開発者は、探しているパッケージの更新バージョンを見つけるのに苦労し、通常の信頼できるパッケージ マネージャーではなく、オンラインのウェブサイトからダウンロードします。パッケージは署名されていないため、整合性を確認する機会がありません。パッケージには悪意のあるコードが含まれています。

シナリオ #4 セキュアでない逆シリアル化: React アプリケーションは、Spring Boot マイクロサービス群を呼び出します。関数型プログラマーである開発者は、コードの不変性を確保しようと試みました。そこで開発者が思いついた解決策は、ユーザー状態をシリアル化し、リクエストごとにそれをやり取りすることでした。攻撃者は「rO0」という Java オブジェクト シグネチャ（base64形式）に気づき、Java Deserialization Scanner を使用して、アプリケーション サーバー上でリモート コード実行権限を取得します。

参考情報

- OWASP Cheat Sheet: Software Supply Chain Security
- OWASP Cheat Sheet: Infrastructure as Code
- OWASP Cheat Sheet: Deserialization
- SAFECode Software Integrity Controls
- A 'Worst Nightmare' Cyberattack: The Untold Story Of The SolarWinds Hack
- CodeCov Bash Uploader Compromise
- Securing DevOps by Julien Vehent
- Insecure Deserialization by Tenendo

対応する CWE の一覧

- CWE-345 Insufficient Verification of Data Authenticity
- CWE-353 Missing Support for Integrity Check
- CWE-426 Untrusted Search Path
- CWE-427 Uncontrolled Search Path Element
- CWE-494 Download of Code Without Integrity Check
- CWE-502 Deserialization of Untrusted Data
- CWE-506 Embedded Malicious Code
- CWE-509 Replicating Malicious Code (Virus or Worm)
- CWE-565 Reliance on Cookies without Validation and Integrity Checking
- CWE-784 Reliance on Cookies without Validation and Integrity Checking in a Security Decision
- CWE-829 Inclusion of Functionality from Untrusted Control Sphere
- CWE-830 Inclusion of Web Functionality from an Untrusted Source
- CWE-915 Improperly Controlled Modification of Dynamically-Determined Object Attributes
- CWE-926 Improper Export of Android Application Components

A09:2025 ログ記録およびアラートの失敗

背景

ログ記録とアラートの失敗は、9 位を維持しました。このカテゴリは、関連するログ イベントへの対応を促すために必要なアラート機能を強調する目的で、若干の名称変更を行いました。このカテゴリは、データでは常に過小評価されており、コミュニティ アンケート参加者からの投票により 3 度目の選出となりました。このカテゴリはテストが非常に難

しく、CVE/CVSS データへの反映も最小限（わずか 723 件）ですが、可視性、インシデント アラート、フォレンジックにおいて非常に大きな影響力を持つ可能性があります。このカテゴリには、*CWE-117:* ログ ファイルへの出力エンコーディングの適切な処理、*CWE-532:* ログ ファイルへの機密データの挿入、*CWE-778:* 不十分なログ記録 に関する問題が含まれます。

スコア表

対応する CWE 数	インシデント割合（最大）	インシデント割合（平均）	カバレッジ（最大）	カバレッジ（平均）	重み付き悪用可能性（最大）	重み付き悪用可能性（平均）	総発生件数	総 CVE 数
5	11.33%	3.91%	85.96%	46.48%	7.19	2.65	260,288	723

解説

ログ記録と監視がなければ、攻撃や侵害を検知できず、アラートがなければセキュリティ インシデント発生時に迅速かつ効果的に対応することは非常に困難です。積極的な対応を開始するためのログ記録、継続的な監視、検知、アラートが不十分な状況は、以下のような例がありますが、常に発生します。

- ログイン、ログインの失敗、高価値のトランザクションなどの監査対象イベントがログに記録されていない、または一貫性のないログ記録が行われています（例えば、成功したログインのみが記録され、失敗したログインは記録されない）。
- 警告やエラーが発生しても、ログ メッセージがまったく生成されない、不十分なログメッセージしか生成されない、あるいは不明瞭なログ メッセージが生成されます。
- ログの整合性が改ざんから適切に保護されていません。
- アプリケーションや API のログは、不審なアクティビティを監視していません。
- ログはローカルにのみ保存され、適切にバックアップされていません。
- 適切なアラートしきい値と対応エスカレーション プロセスが整備されていないか、有効ではありません。アラートは妥当な時間内に受信またはレビューされていません。
- ペネトレーション テストおよび動的アプリケーション セキュリティ テスト (DAST) ツール（Burp や ZAP など）によるスキャンでは、アラートがトリガーされません。
- アプリケーションは、アクティブな攻撃をリアルタイムまたはほぼリアルタイムで検出、エスカレーション、またはアラートすることができません。
- ログ記録およびアラート イベントをユーザーまたは攻撃者に公開したり（A01:2025-アクセス制御の不備 を参照）、ログに記録すべきでない機密情報（PII や PHI など）をログに記録したりすると、機密情報の漏洩に対して脆弱になります。
- ログ データが正しくエンコードされていない場合、ログ記録システムまたは監視システムへのインジェクションや攻撃に対して脆弱になります。
- アプリケーションがエラーやその他の例外的な状況を認識していない、または適切に処理していないため、システムがエラーを認識できず、問題が発生したことをログに記録できません。
- アラートを発行するための適切な「ユース ケース」が不足しているか、古くなっているため、特別な状況を認識できません。
- 誤検知アラートが多すぎると、重要なアラートと重要でないアラートを区別できなくなり、アラートの認識が遅れたり、まったく認識されなかったりします（SOC チームの物理的な負荷が高まります）。
- ユース ケースのプレイブックが不完全、古い、または不足しているため、検出されたアラートを正しく処理できません。

防止方法

開発者は、アプリケーションのリスクに応じて、以下の管理策の一部またはすべてを実装する必要があります。

- ログイン、アクセス制御、サーバー側入力検証の失敗をすべて、十分なユーザー コンテキストでログに記録し、疑わしいアカウントや悪意のあるアカウントを特定できるようにし、後で行われるフォレンジック分析に十分な時間保持できるようにします。
- セキュリティ管理策を含むアプリケーションのすべての部分が、成功か失敗かに関わらずログに記録されるようにします。
- ログ管理ソリューションが容易に使用できる形式でログが生成されるようにします。
- ログ データが正しくエンコードされ、ログ システムや監視システムへのインジェクションや攻撃を防止できるようにします。
- すべてのトランザクションに、改ざんや削除を防ぐための整合性管理策（追記専用のデータベース テーブルなど）を備えた監査証跡があることを確実にします。
- エラーが発生したすべてのトランザクションがロール バックされ、最初からやり直されるようにします。常にフェイル クローズします。
- アプリケーションまたはそのユーザーが疑わしい動作をした場合は、アラートを発行します。開発者がこれに対処するためのコードを記述したり、システムを購入したりできるように、このトピックに関するガイダンスを作成します。
- DevSecOps およびセキュリティ チームは、セキュリティ オペレーション センター (SOC) チームによって疑わしいアクティビティが検出され、迅速に対応できるように、プレイブックを含む効果的な監視およびアラートのユース ケースを確立する必要があります。
- 攻撃者を罠にかける「ハニー トークン」を、アプリケーション（データベース、データなど）に、実在のユーザー ID や技術的ユーザー ID として追加します。ハニー トークンは通常の業務では使用されないため、アクセスがあった場合、ほぼ誤検知なしでアラートを生成できるログ データが生成されます。
- アラートの誤検知率を低減するために、振舞い分析と AI サポートをオプションで追加することも可能です。
- 米国国立標準技術研究所（NIST）800-61r2 以降に準拠したインシデント対応・復旧計画を策定または導入します。ソフトウェア開発者にアプリケーション攻撃やインシデントの発生状況を指導し、報告できるようにします。

OWASP ModSecurity Core Rule Set などの商用およびオープン ソースのアプリケーション保護製品や、Elasticsearch、Logstash、Kibana（ELK）スタックなどのオープン ソースのログ相関ソフトウェアには、カスタム ダッシュボードとアラート機能が搭載されており、これらの問題への対処に役立ちます。ほぼリアルタイムで攻撃に対応したりブロックしたりするのに役立つ商用の可観測性ツールもあります。

攻撃シナリオの例

シナリオ #1: 小児医療保険会社のウェブサイト運営者は、監視とログ記録の不足により、侵害を検知できませんでした。外部関係者から、攻撃者が 350 万人以上の小児の数千件の機密性の高い医療記録にアクセスし、改ざんしたとの報告を受けました。事後調査の結果、ウェブサイト開発者が重大な脆弱性に対処していなかったことが判明しました。システムのログ記録や監視が不十分であったため、データ侵害は 2013 年から 7 年以上にわたって進行していた可能性があります。

シナリオ #2: インドの大手航空会社で、パスポートやクレジット カード情報を含む、10 年以上にわたる数百万人の乗客の個人データが漏洩しました。データ漏洩はサードパーティのクラウド ホスティング プロバイダーで発生し、プロバイダーはしばらくして航空会社に漏洩を通知しました。

シナリオ #3: 欧州の大手航空会社が GDPR 報告義務違反に見舞われました。この違反は、攻撃者が決済アプリケーションのセキュリティ脆弱性を悪用したことが原因と報じられており、攻撃者は 40 万件以上の顧客決済記録を盗み出しました。この結果、航空会社は GDPR 規制当局から 2,000 万ポンドの罰金を科されました。

参考情報

- OWASP Proactive Controls: C9: Implement Logging and Monitoring
- OWASP Application Security Verification Standard: V16 Security Logging and Error Handling
- OWASP Cheat Sheet: Application Logging Vocabulary
- OWASP Cheat Sheet: Logging
- Data Integrity: Recovering from Ransomware and Other Destructive Events
- Data Integrity: Identifying and Protecting Assets Against Ransomware and Other Destructive Events
- Data Integrity: Detecting and Responding to Ransomware and Other Destructive Events

対応する CWE の一覧

- CWE-117 Improper Output Neutralization for Logs
- CWE-221 Information Loss of Omission
- CWE-223 Omission of Security-relevant Information
- CWE-532 Insertion of Sensitive Information into Log File
- CWE-778 Insufficient Logging

A10:2025 例外的な状況への不適切な対処

背景

例外的な状況への不適切な対処は、2025 年に新設されたカテゴリです。このカテゴリには 24 件の CWE が含まれており、不適切なエラー処理、論理エラー、フェイル オープン、その他システムが遭遇する可能性のある異常な状況に起因する関連シナリオに焦点を当てています。このカテゴリには、以前はコード品質の低さと関連付けられていた CWE も含まれています。しかし、以前の分類はあまりにも包括的すぎたため、より具体的なこのカテゴリの方が、より適切なガイダンスを提供できると考えています。

このカテゴリに含まれる主な CWE は以下のとおりです: *CWE-209* 機密情報を含むエラー メッセージの生成、*CWE-234* パラメータ不足の処理漏れ、*CWE-274* 不十分な権限の不適切な処理、*CWE-476 NULL* ポインタの参照解除、および *CWE-636* 安全でないフェイル オーバー（「フェイル オープン」）。

スコア表

対応する CWE 数	インシデント割合（最大）	インシデント割合（平均）	カバレッジ（最大）	カバレッジ（平均）	重み付き悪用可能性（最大）	重み付き悪用可能性（平均）	総発生件数	総 CVE 数
24	20.67%	2.95%	100.00%	37.95%	7.11	3.81	769,581	3,416

解説

ソフトウェアにおける例外的な状況への不適切な対処とは、プログラムが異常かつ予測不可能な状況を防止、検出、および適切に対応できない場合に発生し、クラッシュ、予期せぬ動作、そして時には脆弱性につながります。これは、以下の 3 つの不備のうち 1 つ以上が関係している可能性があります。それは、アプリケーションが異常な状況の発生を防止しない、発生している状況を認識しない、状況発生後の対応が不十分であるか全く対応しない、といったケースです。

例外的な状況は、入力検証の欠落、不備、または不完全さ、あるいはエラーが発生した関数ではなく、より上位のレベルでエラー処理が行われること、メモリ、権限、ネットワークの問題などの予期せぬ環境状態、一貫性のない例外処理、または全く処理されない例外などによって引き起こされる可能性があります。これらにより、システムは未知の予測不可能な状態に陥ります。アプリケーションが次に実行すべき命令を判断できない場合、例外処理が適切に行われていないことになります。発見が困難なエラーや例外は、アプリケーション全体のセキュリティを長期間にわたって脅かす可能性があります。

例外的な状況への不適切な対処によって、ロジック バグ、オーバー フロー、競合状態、不正なトランザクション、メモリ、状態、リソース、タイミング、認証、認可に関する問題など、さまざまなセキュリティ脆弱性が発生する可能性があります。これらの脆弱性は、システムまたはデータの機密性、可用性、完全性に悪影響を与える可能性があります。攻撃者は、アプリケーションの不完全なエラー処理を悪用して、これらの脆弱性を突きます。

防止方法

例外的な状況に適切に対処するためには、そのような状況（最悪の事態）を想定して計画を立てる必要があります。発生しうるあらゆるシステム エラーは、発生箇所で直接「捕捉」し、適切に処理しなければなりません（つまり、問題を解決し、システムが正常な状態に復旧するように、何らかの有意義な対応を行う必要があります）。処理の一環として、エラーをスローする（ユーザーに分かりやすい形で通知する）、イベントをログに記録する、必要に応じてアラートを発行する、といった対応を含める必要があります。また、万が一見落としがあった場合に備えて、グローバルな例外ハンドラも用意しておく必要があります。理想的には、繰り返し発生するエラーや、継続的な攻撃を示唆するパターンを監視し、何らかの対応、防御、またはブロックを行うための監視ツールやオブザーバビリティ機能も備えているべきです。これにより、エラー処理の弱点を狙うスクリプトやボットをブロックし、対処することができます。

例外的な状況を捕捉して適切に処理することで、プログラムの基盤となるインフラストラクチャが予測不能な状況に対処する必要がなくなるようにすることを確実にします。何らかのトランザクションの途中でエラーが発生した場合は、トランザクションのすべての部分をロール バックして最初からやり直すことが非常に重要です（これは「クローズド フェイル」とも呼ばれます）。トランザクションの途中で復旧を試みると、回復不能なエラーが発生することがよくあります。

可能な限り、レート制限、リソース クォータ、スロットリングなどの制限を適用し、例外的な状況がそもそも発生しないようにします。情報技術において、あらゆるものが無制限であるべきではありません。無制限な状態は、アプリケーションの回復力の低下、サービス拒否攻撃、ブルートフォース攻撃の成功、そして高額なクラウド料金につながるからです。一定の頻度を超えて同一のエラーが繰り返し発生する場合、それらのエラーは、発生頻度と時間枠を示す統計情報としてのみ出力することを検討してください。この情報は、自動ログ記録および監視システムに影響を与えないように、元のメッセージに追加する必要があります（A09:2025 ログ記録とアラートの失敗 TJを参照）。

さらに、厳格な入力検証（受け入れざるを得ない潜在的に危険な文字についてはサニタイズまたはエスケープ処理を行う）、そして一元化されたエラー処理、ログ記録、監視、アラート機能、およびグローバル例外ハンドラを実装する必要があります。アプリケーション内で例外的な状況を処理する機能が複数存在すべきではなく、すべて一箇所で、常に同じ方法で処理される必要があります。また、このセクションで述べたすべての事項についてプロジェクトのセキュリティ要件を策定し、プロジェクトの設計段階で脅威モデリングやセキュア設計レビューを実施し、コードレビューまたは静的解析を行うとともに、最終システムに対して負荷テスト、パフォーマンス テスト、および侵入テストを実施する必要があります。

可能であれば、組織全体で例外的な状況への対処方法を統一する必要があります。そうすることで、この重要なセキュリティ対策におけるコードのエラーをレビューおよび監査しやすくなります。

攻撃シナリオの例

シナリオ #1: アプリケーションがファイルのアップロード時に例外を捕捉したもののその後リソースを適切に解放しなかった場合、例外条件の不適切な処理によるリソース枯渇（サービス拒否）が発生する可能性があります。新たな例外が発生するたびに、リソースがロックされたり、使用できなくなったりする状態が続き、最終的にすべてのリソースが使い果たされます。

シナリオ #2: 不適切なエラー処理やデータベース エラーによってシステム エラーの詳細がユーザーに表示されてしまうことで、機密データが漏洩する可能性があります。攻撃者は、この機密性の高いシステム情報を利用して、より高度な SQL インジェクション攻撃を仕掛けるために、意図的にエラーを発生させ続けます。ユーザーに表示されるエラー メッセージに含まれる機密データは、攻撃者にとって偵察情報となります。

シナリオ #3: 金融取引における状態の破損は、攻撃者がネットワーク障害などを利用して複数ステップのトランザクションを中断させることで発生する可能性があります。例えば、トランザクションの順序が「ユーザーの口座からの引き落とし」「送金先口座への入金」「トランザクションのログ記録」だったとします。システムが途中でエラーが発生した場合にトランザクション全体を適切にロール バックしない（フェイル クローズしない）場合、攻撃者はユーザーの口座から資金を不正に引き出したり、競合状態を利用して送金先口座に複数回送金したりする可能性があります。

参考情報

OWASP MASVS-RESILIENCE

- OWASP Cheat Sheet: Logging
- OWASP Cheat Sheet: Error Handling
- OWASP Application Security Verification Standard (ASVS): V16.5 Error Handling
- OWASP Testing Guide: 4.8.1 Testing for Error Handling
- Best practices for exceptions (Microsoft, .Net)

- Clean Code and the Art of Exception Handling (Toptal)
- General error handling rules (Google for Developers)
- Example of real-world mishandling of an exceptional condition

対応する CWE の一覧

- CWE-209 Generation of Error Message Containing Sensitive Information
- CWE-215 Insertion of Sensitive Information Into Debugging Code
- CWE-234 Failure to Handle Missing Parameter
- CWE-235 Improper Handling of Extra Parameters
- CWE-248 Uncaught Exception
- CWE-252 Unchecked Return Value
- CWE-274 Improper Handling of Insufficient Privileges
- CWE-280 Improper Handling of Insufficient Permissions or Privileges
- CWE-369 Divide By Zero
- CWE-390 Detection of Error Condition Without Action
- CWE-391 Unchecked Error Condition
- CWE-394 Unexpected Status Code or Return Value
- CWE-396 Declaration of Catch for Generic Exception
- CWE-397 Declaration of Throws for Generic Exception
- CWE-460 Improper Cleanup on Thrown Exception
- CWE-476 NULL Pointer Dereference
- CWE-478 Missing Default Case in Multiple Condition Expression
- CWE-484 Omitted Break Statement in Switch
- CWE-550 Server-generated Error Message Containing Sensitive Information
- CWE-636 Not Failing Securely ('Failing Open')
- CWE-703 Improper Check or Handling of Exceptional Conditions
- CWE-754 Improper Check for Unusual or Exceptional Conditions
- CWE-755 Improper Handling of Exceptional Conditions
- CWE-756 Missing Custom Error Page

次のステップ

OWASP Top 10 は、設計上、最も重大なリスク 10 件に限定されています。すべての OWASP Top 10 には、掲載に向けて綿密に検討されたものの、最終的に採用されなかった「選ばれる一歩手前の」リスクが含まれています。他のリスクの方がより蔓延しており、影響も大きいからです。

成熟したアプリケーション セキュリティ プログラムの構築を目指す組織、セキュリティ コンサルタント、あるいは製品の適用範囲を拡大したいと考えているツール ベンダーにとって、以下の 2 つの問題は、特定して対策を講じる価値が十分にあります。

X01:2025 アプリケーションの回復力の欠如

背景

これは 2021 年のサービス拒否攻撃を名称変更したものです。これは根本原因ではなく結果を説明していたため、名称が変更されました。このカテゴリは、レジリエンス（回復力）の問題に関連する脆弱性を説明する CWE に焦点を当てています。このカテゴリのスコアは、A10:2025-例外的な状況への不適切な対処 と非常に近いものでした。関連する CWE には、*CWE-400* 制御不能なリソース消費、*CWE-409* 高度に圧縮されたデータの不適切な処理（データ増幅）、*CWE-674* 制御不能な再帰、*CWE-835* 到達不可能な終了条件を持つループ（「無限ループ」） などがあります。

スコア表

対応する CWE 数	インシデント割合（最大）	インシデント割合（平均）	カバレッジ（最大）	カバレッジ（平均）	重み付き悪用可能性（最大）	重み付き悪用可能性（平均）	総発生件数	総 CVE 数
16	20.05%	4.55%	86.01%	41.47%	7.92	3.49	865,066	4,423

解説

このカテゴリは、アプリケーションがストレス、障害、そして障害から回復できないエッジ ケースへの対応方法におけるシステム的な弱点を代表しています。アプリケーションが予期せぬ状況、リソース制約、その他の有害事象を適切に処理、耐久、回復できない場合、（最も一般的には）可用性の問題だけでなく、データ破損、機密データの漏洩、連鎖障害、セキュリティ管理策のバイパスなどにも容易につながる可能性があります。

さらに、X02:2025 メモリ管理の失敗 も、アプリケーションまたはシステム全体の障害につながる可能性があります。

防止方法

この種類の脆弱性を防止するには、システムの障害発生と復旧を想定した設計が必要です。

- 制限、クォータ、フェイル オーバー機能を追加し、特にリソースを最も消費する動作に注意します。
- リソースを大量に消費するページを特定し、事前に計画を立てます。特に、不要な「ガジェット」や、大量のリソース（CPU、メモリなど）を必要とする機能を、未知のユーザーや信頼できないユーザーに公開しないようにすることで、攻撃対象領域を削減します。
- 許可リストとサイズ制限を使用して厳格な入力検証を行い、徹底的にテストします。
- レスポンスのサイズを制限し、生のレスポンスをクライアントに返さないようにします（サーバー側で処理します）。
- デフォルトで safe/closed に設定し（決して開かない）、デフォルトで拒否し、エラーが発生した場合はロール バックします。
- リクエスト スレッドで同期呼び出しをブロックしないようにします（非同期/非ブロッキングを使用する、タイムアウトを設定する、同時実行数を制限するなど）。
- エラー処理機能を慎重にテストします。
- サーキット ブレーカー、バルク ヘッド、リトライ ロジック、グレースフル デグラデーションなどの回復力パターンを実装します。
- パフォーマンス テストと負荷テストを実施します。リスク選好がある場合は、カオス エンジニアリングを追加します。
- 合理的かつ費用対効果の高い範囲で、冗長性を実装および設します。
- 監視、可観測性、アラート機能を実装します。
- RFC 2267 に従って、無効な送信元アドレスをフィルタリングします。
- フィンガー プリント、IP アドレス、または振舞いに基づいて、既知のボット ネットを動的にブロックします。
- プルーフ・オブ・ワーク：攻撃者側でリソースを消費する操作を開始します。この操作は、通常のユーザーには大きな影響を与えませんが、大量のリクエストを送信しようとするボットには影響を与えます。システムの全体的な負荷が上昇した場合、特に信頼性の低いシステムやボットのように見えるシステムに対しては、プルーフ・オブ・ワークの難易度を上げます。
- 非アクティブ状態と最終タイム アウトに基づいて、サーバー側のセッション時間を制限します。
- セッションにバインドされた情報の保存を制限します。

攻撃シナリオの例

シナリオ #1: 攻撃者は意図的にアプリケーション リソースを消費し、システム内で障害を引き起こし、サービス拒否を引き起こします。具体的には、メモリ枯渇、ディスク容量の枯渇、CPU の飽和、あるいは無限接続の確立などが挙げられます。

シナリオ #2: 入力ファジングにより、細工された応答が生成され、アプリケーションのビジネス ロジックが破壊されます。

シナリオ #3: 攻撃者はアプリケーションの依存関係に着目し、API やその他の外部サービスを停止させることで、アプリケーションの動作を継続不能にします。

参考情報

- OWASP Cheat Sheet: Denial of Service
- OWASP MASVS-RESILIENCE
- ASP.NET Core Best Practices (Microsoft)
- Resilience in Microservices: Bulkhead vs Circuit Breaker (Parser)
- Bulkhead Pattern (Geeks for Geeks)
- NIST Cybersecurity Framework (CSF)
- Avoid Blocking Calls: Go Async in Java (Devlane)

対応する CWE の一覧

- CWE-73 External Control of File Name or Path
- CWE-183 Permissive List of Allowed Inputs
- CWE-256 Plaintext Storage of a Password
- CWE-266 Incorrect Privilege Assignment
- CWE-269 Improper Privilege Management
- CWE-286 Incorrect User Management
- CWE-311 Missing Encryption of Sensitive Data
- CWE-312 Cleartext Storage of Sensitive Information

- CWE-313 Cleartext Storage in a File or on Disk
- CWE-316 Cleartext Storage of Sensitive Information in Memory
- CWE-362 Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')
- CWE-382 J2EE Bad Practices: Use of System.exit()
- CWE-419 Unprotected Primary Channel
- CWE-434 Unrestricted Upload of File with Dangerous Type
- CWE-436 Interpretation Conflict
- CWE-444 Inconsistent Interpretation of HTTP Requests ('HTTP Request/Response Smuggling')
- CWE-451 User Interface (UI) Misrepresentation of Critical Information
- CWE-454 External Initialization of Trusted Variables or Data Stores
- CWE-472 External Control of Assumed-Immutable Web Parameter
- CWE-501 Trust Boundary Violation
- CWE-522 Insufficiently Protected Credentials
- CWE-525 Use of Web Browser Cache Containing Sensitive Information
- CWE-539 Use of Persistent Cookies Containing Sensitive Information
- CWE-598 Use of GET Request Method With Sensitive Query Strings
- CWE-602 Client-Side Enforcement of Server-Side Security
- CWE-628 Function Call with Incorrectly Specified Arguments
- CWE-642 External Control of Critical State Data
- CWE-646 Reliance on File Name or Extension of Externally-Supplied File
- CWE-653 Improper Isolation or Compartmentalization
- CWE-656 Reliance on Security Through Obscurity
- CWE-657 Violation of Secure Design Principles
- CWE-676 Use of Potentially Dangerous Function
- CWE-693 Protection Mechanism Failure
- CWE-799 Improper Control of Interaction Frequency
- CWE-807 Reliance on Untrusted Inputs in a Security Decision
- CWE-841 Improper Enforcement of Behavioral Workflow
- CWE-1021 Improper Restriction of Rendered UI Layers or Frames
- CWE-1022 Use of Web Link to Untrusted Target with window.opener Access
- CWE-1125 Excessive Attack Surface

X02:2025 メモリ管理の失敗

背景

Java、C#、JavaScript/TypeScript (node.js)、Go、そして「安全な」Rust といった言語はメモリ セーフです。メモリ管理の問題は、C や C++ といったメモリ セーフではない言語で発生する傾向があります。このカテゴリは、関連する CVE が 3 番目に多いにもかかわらず、コミュニティ調査では最も低いスコアとなり、データでも低い数値となっています。これは、従来のデスクトップ アプリケーションよりも Web アプリケーションが主流になっているためだと考えられます。メモリ管理の脆弱性は、CVSS スコアが最も高いことがよくあります。

スコア表

対応する CWE 数	インシデント割合（最大）	インシデント割合（平均）	カバレッジ（最大）	カバレッジ（平均）	重み付き悪用可能性（最大）	重み付き悪用可能性（平均）	総発生件数	総 CVE 数
24	2.96%	1.13%	55.62%	28.45%	6.75	4.82	220,414	30,978

解説

アプリケーションが自らメモリ管理せざるを得ない場合、ミスが発生しやすくなります。メモリ セーフ言語の使用は増加傾向にありますが、世界中で運用されているレガシー システム、メモリ セーフではない言語の使用を必要とする新規の低レベル システム、そしてメイン フレーム、IoT デバイス、ファームウェアなど、独自でメモリを管理せざるを得ない可能性のあるシステムと連携する Web アプリケーションは依然として多く存在します。代表的な CWE としては、*CWE-120* 入力サイズ チェックなしのバッファ コピー（「クラシック バッファオ ーバーフロー」） と *CWE-121* スタック ベース バッファ オーバーフロー が挙げられます。

メモリ管理の失敗は、以下のような場合に発生する可能性があります。

- 変数に十分なメモリを割り当てていません。
- 入力を検証せず、ヒープ、スタック、バッファのオーバーフローを引き起こしています。
- 変数の型が保持できるよりも大きなデータ値を格納しています。
- 未割り当てのメモリ空間またはアドレス空間を使用しようとしています。
- オフ バイ ワン エラー（0 ではなく 1 からカウントする）を引き起こしています。
- 解放済みのオブジェクトにアクセスしようとしています。
- 初期化されていない変数を使用しています。
- メモリ リークが発生しているか、アプリケーションが失敗するまで利用可能なメモリをすべて使い切っています。

メモリ管理の失敗は、アプリケーション、さらにはシステム全体の障害につながる可能性があります。X01:2025 アプリケーションの回復力の欠如 も参照してください。

防止方法

メモリ管理の失敗を防ぐ最善の方法は、メモリ セーフな言語を使用することです。例としては、Rust、Java、Go、C#、Python、Swift、Kotlin、JavaScript などがあります。新しいアプリケーションを開発する際には、メモリ セーフな言語への移行は学習に見合う価値があることを組織に強く納得させましょう。完全なリファクタリングを行う場合は、可能かつ実現可能な場合は、メモリ セーフな言語での書き換えを強く推奨してください。

メモリセーフな言語を使用できない場合は、以下の手順を実行します。

- メモリ管理エラーの悪用を困難にする次のサーバー機能を有効にします: アドレス空間配置のランダム化 (ASLR)、データ実行保護 (DEP)、構造化例外処理上書き保護 (SEHOP)。
- アプリケーションのメモリ リークを監視します。
- システムへのすべての入力を慎重に検証し、想定を満たさない入力はすべて拒否します。
- 使用している言語を調査し、安全でない関数と安全性の高い関数のリストを作成し、そのリストをチーム全体と共有します。可能であれば、セキュア コーディングのガイドラインまたは標準に追加します。例えば、C 言語では、strcpy() よりも strncpy()、strcat() よりも strncat() を優先します。
- 使用している言語またはフレームワークでメモリ安全性ライブラリが提供されている場合は、それらを使用します。例えば、Safestringlib や SafeStr などです。
- 可能な限り、生の配列やポインタではなく、マネージド バッファとマネージド 文字列を使用します。
- メモリの問題や選択した言語に焦点を当てたセキュア コーディングのトレーニングを受講します。トレーナーに、メモリ管理の失敗が懸念されることを伝えます。
- コード レビューや静的解析を実施します。
- StackShield、StackGuard、Libsafe などのメモリ管理を支援するコンパイラ ツールを使用します。
- システムへのすべての入力に対してファジングを実施します。
- ペネトレーション テストを実施する場合は、メモリ管理の失敗が懸念事項であり、テスト中に特に注意するようテスターに伝えます。
- すべてのコンパイラ エラーと警告を修正します。プログラムがコンパイルされるからといって、警告を無視してはなりません。
- 基盤となるインフラストラクチャが定期的にパッチ適用、スキャン、堅牢化されていることを確実にします。
- 基盤となるインフラストラクチャを、特に潜在的なメモリ脆弱性やその他の障害がないか監視します。
- アドレス スタックをオーバーフロー攻撃から保護するために、カナリア の使用を検討します。

攻撃シナリオの例

シナリオ #1: バッファ オーバーフローは最もよく知られているメモリ脆弱性です。これは、攻撃者がフィールドに許容量を超える情報を送信し、基になる変数用に作成されたバッファをオーバーフローさせる状況です。攻撃が成功すると、オーバーフローした文字列がスタック ポインタを上書きし、攻撃者がプログラムに悪意のある命令を挿入できるようになります。

シナリオ #2: ユーズ アフター フリー（UAF）は頻繁に発生するため、ブラウザのバグ報奨金プログラムでもよく取り上げられます。DOM 要素を操作する JavaScript を処理する Web ブラウザを想像してみてください。攻撃者は、オブジェクト（DOM 要素など）を作成し、その参照を取得する JavaScript ペイロードを作成します。巧妙な操作によって、ブラウザはオブジェクトのメモリを解放しますが、そのオブジェクトへのダングリング ポインタは保持されます。ブラウザがメモリが解放されたことに気付く前に、攻撃者は同じメモリ空間を占有する新しいオブジェクトを割り当てます。ブラウザが元のポインタを使用しようとすると、そのポインタは攻撃者が制御するデータを指すようになります。このポインタが仮想関数テーブルへのポインタだった場合、攻撃者はコード実行を自身のペイロードにリダイレクトできます。

シナリオ #3: ユーザー入力を受け付け、適切に検証またはサニタイズせずに、ログ出力関数に直接渡すネットワーク サービスがあるとします。ユーザーからの入力は、syslog("%s", user_input) ではなくフォーマットを指定しない syslog(user_input) としてログ出力関数に渡されます。攻撃者は、スタック メモリの読み取り（機密データの漏洩）のための %x や、メモリ アドレスへの書き込みのための %n などのフォーマット指定子を含む悪意のあるペイロードを送信します。複数のフォーマット指定子を連鎖させることで、スタックをマッピングし、重要なアドレスを特定して上書きすることができます。これは、フォーマット文字列の脆弱性（制御されていない文字列フォーマット）です。

注: 最新のブラウザは、ブラウザ サンドボックス、ASLR、DEP/NX、RELRO、PIE など、多段階の防御策を用いてこのような攻撃を防御しています。ブラウザに対するメモリ管理エラー攻撃は、簡単に実行できません。

参考情報

- OWASP community pages: Memory leak, Doubly freeing memory, & Buffer Overflow
- Awesome Fuzzing: a list of fuzzing resources
- Project Zero Blog

- Microsoft MSRC Blog

対応する CWE の一覧

- CWE-14 Compiler Removal of Code to Clear Buffers
- CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer
- CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
- CWE-121 Stack-based Buffer Overflow
- CWE-122 Heap-based Buffer Overflow
- CWE-124 Buffer Underwrite ('Buffer Underflow')
- CWE-125 Out-of-bounds Read
- CWE-126 Buffer Over-read
- CWE-190 Integer Overflow or Wraparound
- 191 Integer Underflow (Wrap or Wraparound)
- CWE-196 Unsigned to Signed Conversion Error
- CWE-367 Time-of-check Time-of-use (TOCTOU) Race Condition
- CWE-415 Double Free
- CWE-416 Use After Free
- CWE-457 Use of Uninitialized Variable
- CWE-459 Incomplete Cleanup
- CWE-467 Use of sizeof() on a Pointer Type
- CWE-787 Out-of-bounds Write
- CWE-788 Access of Memory Location After End of Buffer
- CWE-824 Access of Uninitialized Pointer

X03:2025 AI が生成するコードにおける不適切な信頼（「バイブ コーディング」）

背景

現在、世界中でAIが話題となり活用されており、ソフトウェア開発者も例外ではありません。AI が生成するコードに関連する CVE や CWE は現時点では存在しませんが、AI が生成するコードには人間が書いたコードよりも多くの脆弱性が含まれることが多いことは周知の事実であり、文書化されています。

解説

ソフトウェア開発の慣行は変化しつつあり、AI の支援を受けて記述されたコードだけでなく、人間の監督をほぼ一切行わずに記述・コミットされたコード（いわゆるバイブ コーディング）も含まれるようになっています。ブログやウェブサイトからコード スニペットを安易にコピーすることは決して良い考えではありませんでしたが、今回のケースでは問題がさらに深刻化しています。良質で安全なコード スニペットは、かつても今も稀であり、システムの制約により AI によって統計的に無視される可能性があります。

防止方法

コードを書くすべての人に、AI を使用する際に以下のの点を考慮することを強く推奨します。

- 提出するすべてのコードは、たとえ AI によって書かれたものやオンライン フォーラムからコピーしたものであっても、完全に読み理解できる必要があります。コミットするすべてのコードに対して、あなたは責任を負います。
- AI 支援コードはすべて、脆弱性がないか徹底的にレビューする必要があります。理想的には、自身の目で確認するだけでなく、この目的のために開発されたセキュリティ ツール（静的解析など）も活用します。OWASP Cheat Sheet Series: Secure Code Review に記載されている従来のコード レビュー手法の活用も検討します。
- 理想的には、自身の独自のコードを記述し、AI に改善を提案させ、AI のコードを確認し、結果に満足するまで AI に修正をさせます。
- 組織のセキュリティ コーディング ガイドライン、標準、ポリシーなど、独自に収集およびレビューした安全なコード サンプルと文書を RAG (Retrieval Augmented Generation) サーバーで使用することを検討し、RAG サーバーでポリシーや標準を適用するようにします。
- 選択した AI で使用するために、プライバシーとセキュリティのガードレールを実装するツールの購入を検討します。
- プライベート AI の購入を検討します。理想的には、AI が組織のデータ、クエリ、コード、またはその他の機密情報に基づいてトレーニングされないことを規定する契約 (プライバシー契約を含む) を締結します。
- IDE と AI の間にモデル コンテキスト プロトコル (MCP) サーバーを実装することを検討し、選択したセキュリティ ツールの使用を強制するように設定します。
- SDLC の一環としてポリシーとプロセスを実装し、開発者 (およびすべての従業員) に組織内で AI をどのように使用すべきか、また使用すべきでないかを通知します。
- IT セキュリティのベスト プラクティスを考慮した、効果的で適切なプロンプトのリストを作成します。理想的には、社内のセキュア コーディング ガイドラインも考慮する必要があります。開発者は、このプロンプトをプログラムの出発点として利用できます。
- AI は、システム開発ライフサイクルの各フェーズにおいて、効果的かつ安全な活用方法の両面で重要な要素となる可能性があるため、賢く活用します。

- 実際、複雑な機能、ビジネス クリティカルなプログラム、または長期間使用されるプログラムには、バイブ コーディングを使用することは推奨され**ません**。
- シャドウ AI の使用に対する技術的なチェックと安全対策を実装します。
- 開発者に対して、ポリシー、安全な AI の使用、ソフトウェア開発で AI を使用するためのベスト プラクティスについてトレーニングを行います。

参考情報

- OWASP Cheat Sheet: Secure Code Review

対応する CWE の一覧

-なし-