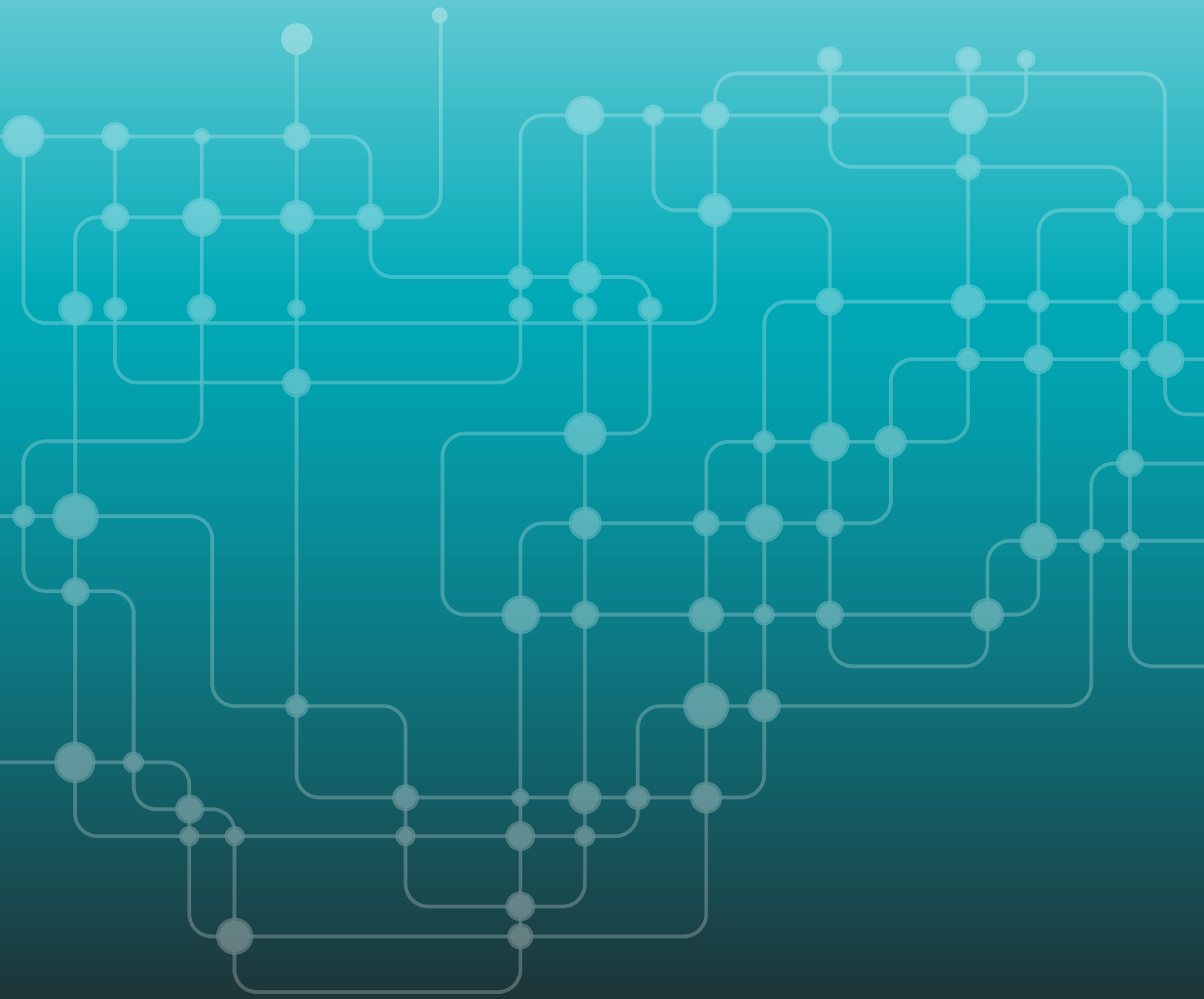
BLACKDUCK®

# 2025 オープンソース・セキュリティ & リスク分析レポート



# 目次

# 2025 OSSRA レポートへようこそ

オープンソース・ソフトウェア（OSS）は、膨大な数のビルド済みコンポーネントをリポジトリで提供することでコスト削減、柔軟性、スケーラビリティなど多くの利点を実現し、アプリケーション開発に革命をもたらしました。しかし、こうしたメリットにはリスクがつきものであり、オープンソースを使用しているすべての組織がこれらのリスクを認識し、いつでも対処できるように準備しておく必要があります。

2025「オープンソース・セキュリティ & リスク分析」（OSSRA）レポートは、セキュリティ上の脆弱性、ライセンスの問題、コンポーネントのメンテナンス、業界トレンドなど、Black Duck® 監査のデータから得られた重要な知見をまとめたものです。今回の分析結果が示すように、オープンソースはほとんどすべてのソフトウェアで使用されており、これらを適切に特定・管理しなければ大きなリスクを抱えることになります。

> 「虞を以て不虞を待つ者は勝つ」
>
> – 孫子

## このレポートの想定読者

このレポートには、さまざまな読者にとって有益な知見が含まれていますが、特に**ソフトウェア・サプライチェーンのセキュリティ対策**に携わっている方、ならびに**ソフトウェア開発、セキュリティおよびリスク管理、合併・買収（M&A）**に直接関与されている方に一読をお勧めします。

開発者の方は、クロスサイト・スクリプティング（XSS）やサービス拒否（DoS）など、今回の監査でオープンソース・ソフトウェアにどのようなタイプの脆弱性が多く見つかったかを知ることができます。例えば、このレポートでは入力バリデーションや サニタイズなどの手法の重要性を強調しており、これらを実践することでよりセキュアなアプリケーション開発が可能になります。

さらに、このレポートでは特に多くの脆弱性が検出されたオープンソース・コンポーネントも特定しています。この情報は、開発者がオープンソース・ライブラリおよびフレームワークを選ぶ際の判断材料になります。例えば、jQuery、jackson-databind、Spring Framework などのコンポーネントには多くの脆弱性が含まれていることが今回のデータで明らかになっており、これらのコンポーネントを使用している開発チームは、定期的な管理とパッチ適用が不可欠であることに注意する必要があります。

2025 OSSRA レポートでは、旧バージョンのコンポーネントの使用に伴うリスク、およびすべての組織が迅速なアップデートのプロセスを確立することの必要性を強調しています。例えば、今回監査したコードベースの 90% で、旧バージョンになってから 4 年以上経過したオープンソース・コンポーネント が見つかっています。旧バージョンのコンポーネントは、セキュリティ・リスクの増大、アタック・サーフェス の拡大、およびコンプライアンスと互換性の問題を引き起こします。旧バージョンのオープンソースがこれほど多く見つかったということは、開発者が改良されたソフトウェアの恩恵を受けていないこと、そして既にメンテナンスが行われなくなったコードに依存し続けていることを示唆しています。

セキュリティに関心のある読者は、このレポートのデータを自社の脆弱性管理プロセスの改善に役立てることができます。例えば、このレポートでは今回の監査で最も多く見つかった CVE（共通脆弱性識別子）と、それらに関連付けられる CWE（共通脆弱性タイプ一覧）を特定しています。

リスク管理の専門家は、OSSRA のデータに基づいてオープンソース・ソフトウェアの導入とリスク軽減に関する戦略的意思決定を行うことができます。脆弱性が含まれる割合など、さまざまな指標を業種間で比較することにより、リスク・マネージャーは自社がどの部分で平均を上回っているのか、あるいはどの部分で改善が必要なのかをピンポイントで特定できます。

OSSRA のデータは、主に M&A の対象コードの分析から得られたものであり、合併・買収取引に関与している専門家は、競合が起こりやすいオープンソース・ライセンス、買収対象企業のセキュリティ態勢、買収対象の知的財産の価値に影響を与えかねない潜在的な運用面の課題など、実際の合併・買収取引で問題になることの多い事項について重要な知見を得ることができます。

## このレポートが伝える重要なメッセージ

**ソフトウェアには、みなさんが考えている以上に多くのオープンソースが含まれています。**監査したコードベースの 97% にオープンソースが含まれており、アプリケーション 1 つあたりに平均で 911 個の OSS コンポーネントが含まれていました。業種別に見ると、オープンソースを含むコードベースの割合はコンピュータ・ハードウェア / 半導体、エドテック、インターネット / モバイル・アプリが 100% と最も多く、最も少ない製造 / 産業 / ロボット工学でも 79% でした。

**オープンソース・コードベースは大規模化と複雑化が進んでいます。**アプリケーション 1 つあたりに含まれるオープンソース・ファイルの数の平均は、過去わずか 4 年間で 3 倍に増加していることが今回の調査で明らかになっています。その理由の 1 つに、「推移的依存関係」（他のソフトウェア・コンポーネントの動作に必要なオープンソース・ライブラリ）の使用があります。オープンソースが他のオープンソースを使用することは珍しくありません。今回の監査では、スキャンで検出されたオープンソース・コンポーネントの 64% が推移的依存関係でした。これらの依存関係は、自動化したツールを使用しなければ特定や追跡はほぼ不可能です。現在使用しているサードパーティ・コードを目録化せずに推移的依存関係のすべてのインスタンスを見つけるのは、干し草の山の中から 1 本の針を見つけるようなものです。

**これらのオープンソースはどこから来ているのか。**今回の監査では、オープンソースの大半がパッケージ管理ツールのリポジトリからダウンロードされていることが明らかになっています。特に、今回のスキャンで見つかった約 100 万個の OSS コンポーネントのうち、実に 28 万個以上がこうしたリポジトリの 1 つである npm（JavaScript パッケージの大規模な公開データベース）からダウンロードされていました。

**オープンソースを「フリー」と考えるかどうかは別として、その使用にはコストが伴います。**現在使用しているアプリケーションに高リスクまたは重大なリスクのあるオープンソース脆弱性が含まれている可能性は 80% 以上あり、それらの半分近くが推移的依存関係によってもたらされています。

**推移的依存関係は、セキュリティ上の課題だけでなくライセンスとメンテナンスの問題も引き起こします。**今回の監査では、半分以上のコードベースにライセンスの競合が見つかっており、その多くが、推移的依存関係のライセンスとそれ以外のコンポーネントのライセンスの両立性の問題に起因していました。今回の監査で見つかったコンポーネント・ライセンス間の競合の約 30% は、推移的依存関係によって引き起こされています。

**静的アプリケーション・セキュリティ・テスト（SAST）および動的アプリケーション・セキュリティ・テスト（DAST）はコーディング・エラーの特定に役立ちます。**これらのテスト手法により、不適切な入力バリデーションや機微な情報の曝露などのエラー、そして重要なデータを平文のままインターネット上に送信すること、古い暗号手法や弱い暗号手法を使用すること、パスワードなどの秘密情報を適切に保護していないことなどのミスを見つけることができます。

**Web アプリケーションおよびサービスを使用している場合は、これらをソフトウェア・コンポジション解析（SCA）および DAST ツールを使用して評価することが重要です。**開発およびセキュリティ・チームは、DAST、SAST、SCA を組み合わせた多面的なセキュリティ・アプローチを導入し、現代のソフトウェアで求められる網羅的なセキュリティ・カバレッジを達成する必要があります。このような全方位のアプローチを適用すれば、深刻な脆弱性にさらされる可能性を著しく軽減できることが今回の調査で明らかになっています。

# このレポートのデータおよび Black Duck 監査について

このレポートは、Black Duck 監査チームが 2024 年に 16 の業種から提出を受けた 965 の商用コードベースに対して 1,658 件の分析を実施し、その結果を匿名化して評価したデータを使用しています（以下の注を参照）。

ブラック・ダックは、さまざまなニーズや目的に合わせたオープンソース監査など、幅広いサービスを提供しています。オープンソース監査は、自動化したツール、充実したデータベース、および専門家による分析を組み合わせることにより、組織における OSS の利用を徹底的に評価します。これらの監査で重要な役割を果たしている Black Duck KnowledgeBase ™には、これまで 20 年間にわたって数百ものオープンソース・コンポーネントに関するデータ（ライセンス、脆弱性、潜在的リスクに関する情報を含む）が登録されています。Black Duck KnowledgeBase に登録されている 780 万以上のオープンソース・コンポーネントに関するデータは、31,000 ものフォージおよびリポジトリからブラック・ダックのサイバーセキュリティ・リサーチ・センター（CyRC）が収集・整理したものです。

通常、Black Duck オープンソース監査は以下の手順で実施します。

- **コードベースの提出：**ブラック・ダックが顧客から監査対象のコードベースの提供を受け、これらへのアクセスを許可してもらいます。これには、ソースコード、バイナリ、およびその他の成果物が含まれます。
- **自動解析：**Black Duck SCA など複数の自動化されたツールを使用してコードベースをスキャンし、高度な文字列検索機能によってすべてのオープンソース・コンポーネントとその依存関係（推移的依存関係を含む）を特定します。
- **専門家のレビュー：**ブラック・ダックのオープンソース専門家チームが自動解析の結果をレビューしてその妥当性、完全性、正確性を確認します。
- **レポートの作成：**ブラック・ダックはさまざまな種類のレポートを作成し、すべてのオープンソース・コンポーネントとそのライセンス、既知の脆弱性、潜在的な運用リスクを詳細に記録したソフトウェア部品表（SBOM）を提供します。これらのレポートには、SBOM に記録された問題の詳細説明も含まれます。
- **修正ガイダンス：**脆弱なコンポーネントのアップデート、ライセンス競合の解決、運用リスクの軽減など、特定した問題の修正方法に関するガイダンスを提供します。

**注：**Black Duck 監査チームが監査データを評価および提示する方法は、2024 年にいくつかの点で改良されました。最も大きな変更は、顧客から提出された 1 つのコードベースを、今回から「プロジェクト」と呼ばれる複数の単位に分割して分析するようになった点です。この新しい手法により、コードベースを細分化して分析できるようになり、より詳細なレポート、コンポーネントの特定および依存関係追跡の精度向上など、顧客にもいくつかのメリットが生まれています。これらの変更は、監査データの提示方法にも影響しています。例えば、今回の OSSRA でも便宜上「コードベース」の表記を残していますが、実際には 2024 年にブラック・ダックに提出された 965 のコードベースを 1,658 のプロジェクトに細分化して分析を実施しています。

# 数字で見る調査結果

**1,658** Black Duck 監査でスキャンしたプロジェクトの数

**97%** オープンソースを含むコードベースの割合

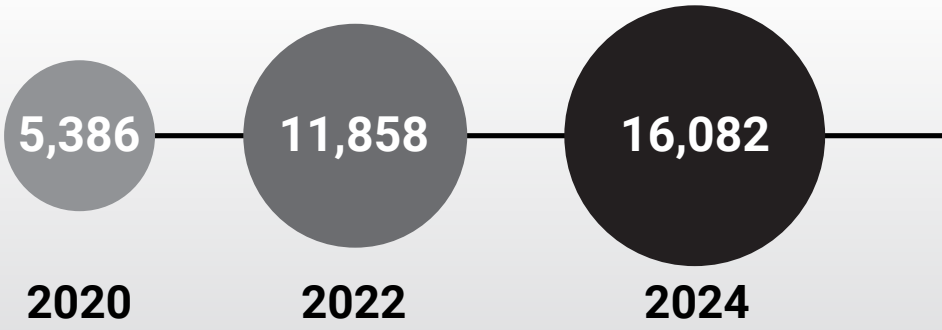
**70%** スキャンしたコードに占めるオープンソース由来コードの割合

**911**
アプリケーション
1 つあたりに含まれる
OSS コンポーネントの数の平均

**64%**

OSS コンポーネントのうち、
過渡的依存関係が
占める割合

アプリケーション
1 つあたりに含まれる
オープンソース・ファイルの
数の平均は過去 4 年間で
3 倍に増加




今回の監査で見つかったオープンソースのうち、npm リポジトリからダウンロードされたものが 280,000 以上ありました。**スキャンで検出された OSS パッケージのほとんどは JavaScript で作成されていました。**

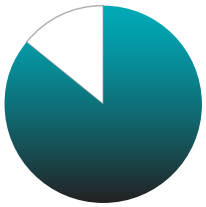
| ダウンロード元リポジトリ | 言語 | 2024 年の監査で見つかったコンポーネントの数 |
|---|---|---|
| npm | JavaScript | 282,521 |
| yarn (JavaScript) | JavaScript | 162,327 |
| pnpm (JavaScript) | JavaScript | 24,069 |

**しかしそれで全部ではありません。**例えば、C/C++ のメモリ安全性の問題を避けるため、Rust パッケージ・リポジトリを使用する開発者が急速に増えています。

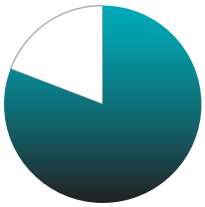
| ダウンロード元リポジトリ | 言語 | 2024 年の監査で見つかったコンポーネントの数 |
|---|---|---|
| Cargo | Rust | 33,327 |
| Nuget | C#、Visual Basic、F#、WiX、C++、Q# | 29,818 |
| go_mod | Go | 24,069 |
| Maven | Java | 14,097 |
| packagist | PHP | 6,112 |
| Gradle | Java、C、JavaScript | 4,615 |

# 数字で見る調査結果

## 脆弱性とセキュリティ



**86%**
リスク診断を受けたコードベースのうち、脆弱性を含んでいたものの割合



**81%**
リスク診断を受けたコードベースのうち、高リスク脆弱性を含んでいたものの割合

**8/10**
高リスク脆弱性 Top 10 のうち 8 つが jQuery に関連

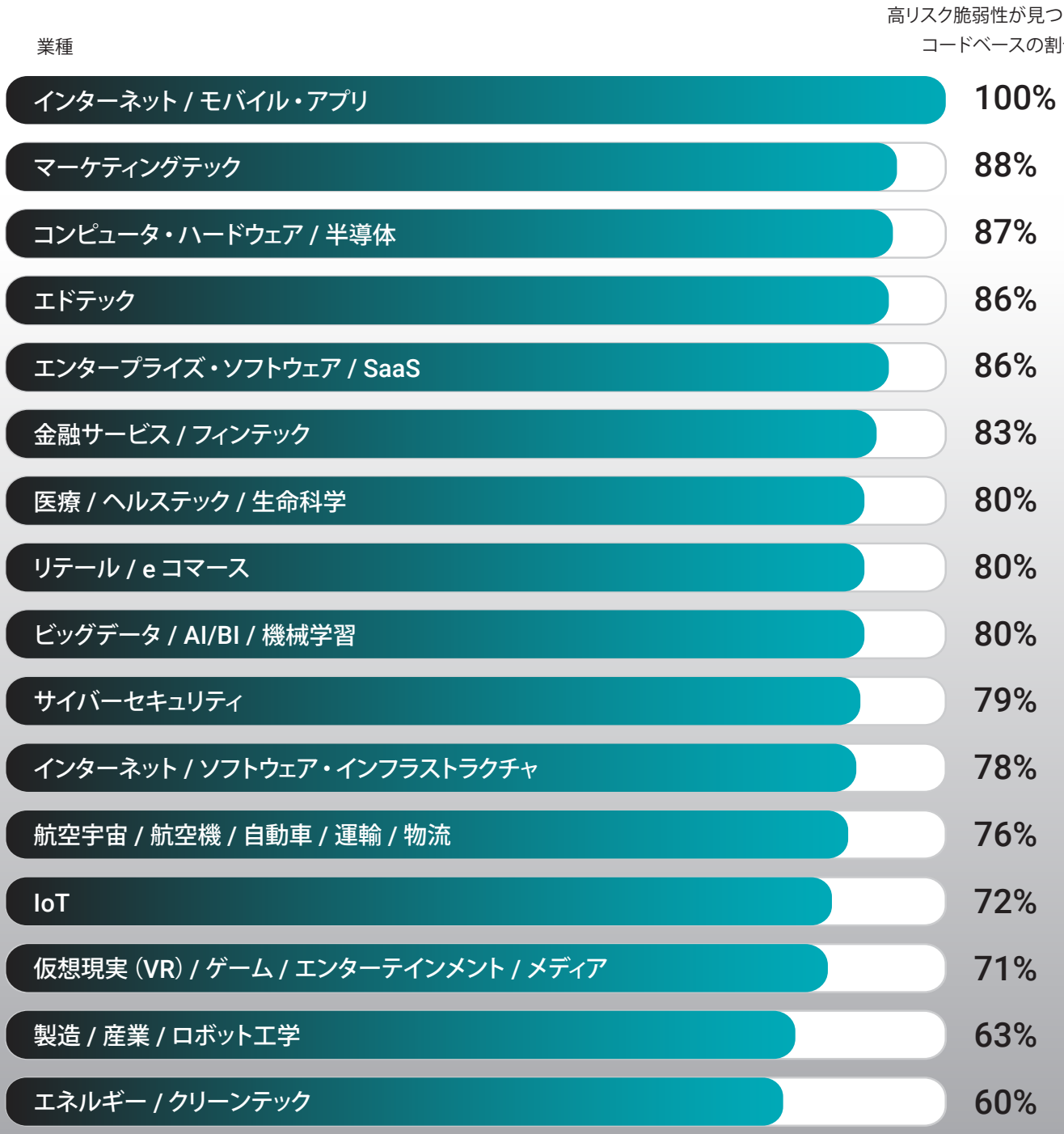



図 1：高リスク脆弱性が見つかったコードベースの業種別割合

# 数字で見る調査結果

## ライセンス

**56%** ライセンスの競合が見つかったコードベースの割合

**33%** ライセンスのない、またはカスタム・ライセンス（通常はソフトウェアの使用方法に関して開発者がコメントとして記述）のOSSコンポーネントを使用しているコードベースの割合

## メンテナンスおよび運用面のリスク

**91%** 旧バージョンのOSSコンポーネントを使用しているコードベースの割合

**90%** 最新バージョンよりも11バージョン以上前のコンポーネントを使用しているコードベースの割合

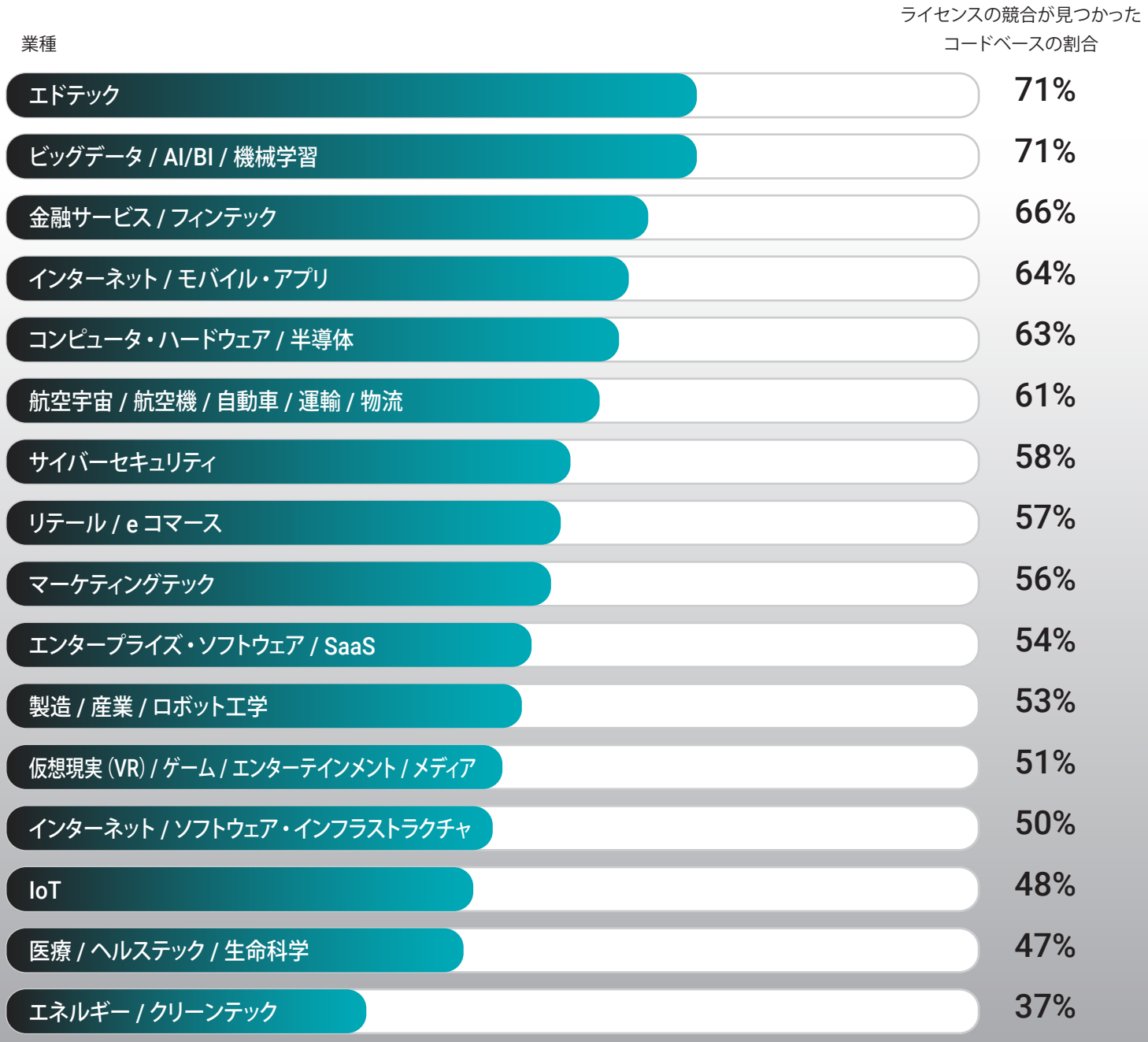



図 2：ライセンスの競合が見つかったコードベースの業種別割合

# オープンソースのリスクと脆弱性

Black Duck 監査では、オープンソース・ライセンスのコンプライアンス診断を必ず実施していますが、脆弱性および運用リスクの診断は任意（オプトアウト方式）で実施しています。2024 年に Black Duck 監査チームが脆弱性および運用リスクの診断を実施したコードベースの数は 901 でした。このセクションと「リスクに影響するメンテナンスおよび運用面の要因」のセクションに示したデータは、これら任意の診断結果に基づいています。

## ソフトウェア・セキュリティの第一歩はコードの可視化から

**86%**

**少なくとも1つの脆弱性が見つかったコードベースの割合**

**81%**

**高リスクまたは重大なリスクのある脆弱性が見つかったコードベースの割合**

**1つのコードベースで最も多く見つかった脆弱性の数（重複を除く）**

**3,548**

**コードベース1つあたりに見つかった脆弱性の数（重複を除く）の平均**

**154**

| コンポーネント | このコンポーネントを含むコードベースの割合 |
|---|---|
| jQuery | 32% |
| jQuery UI | 16% |
| Bootstrap (Twitter) | 15% |
| Spring Framework | 12% |
| Lodash | 12% |
| Netty Project | 11% |
| jackson-databind | 9% |
| Apache Tomcat | 8% |
| Python プログラミング言語 | 5% |
| TensorFlow | 1% |

図 3：高リスクまたは重大なリスクのある脆弱性を含むコンポーネント Top 10

# リスク管理を理解し、コードを可視化する

効果的なオープンソース・リスク管理とは、すべての脆弱性を見つけて修正することではありません。仮にそのようなことをしても、果てしない徒労に終わるだけです。そうではなく、リスク管理の本質とは、コードのリスクに関して正確な意思決定を下すために必要な知識を得ることにあります。例えば、ある脆弱性が特定された場合、その重大度、悪用可能性、そして自社システムに対する潜在的な影響を評価するといったことです。同様に、すべてのライセンス競合やコード品質上の問題がすべての組織にとって優先度の高い問題となるわけではありません。最も重大な問題から重点的に対処することが、オープンソースの効率的なリスク管理の基本です。

しかし、重大な問題に対処するには、まずその問題を認識する必要があります。修正作業に優先順位を付けるには、コードを深いレベルで理解することが重要です。そこで、すべてのソフトウェア・コンポーネントとその依存関係を完全に記録した SBOM を導入する組織が増えています。

> 「確信がなければ、さらなる探究が必要だ」
>
> – エリック・サイデル（ポーカー・プレイヤー）

# SCA と SBOM でソフトウェアのセキュリティと透明性を高める

SBOM とは、ソフトウェアの開発に使用したすべてのコンポーネントの詳細とサプライチェーンの関係を記録した正式な文書です。また、SBOM はオープンソース・ライブラリ、サードパーティ・モジュール、フレームワーク、およびそれらに関連するメタデータ（ライセンスやバージョンなど）といった、ソフトウェア・アプリケーションのすべての構成要素の目録としての役割も果たします。SBOM によってソフトウェアの構成要素の透明性が確保され、自社の組織で何が稼働しているかを理解できるようになり、最終的にセキュリティ・チームはリスクを理解し、依存関係を追跡し、ソフトウェアを監査できるようになります。

Linux Foundation が実施した調査によると、SBOM を生成している組織は、アプリケーションを構成するコンポーネント間の依存関係の理解、コンポーネントに存在する脆弱性の監視、そして OSS ライセンスのコンプライアンスの管理ができるようになることが分かっています。また、Gartner® によるレポートで強調されているように、SBOM を導入することで、ソフトウェア・サプライチェーンにおける自社開発コードとオープンソース・コードの可視性、透明性、セキュリティ、完全性が向上します。現在では、ソフトウェア・デリバリ・パイプラインの最も下流に位置する顧客の多くが、ベンダー契約において SBOM の提供を要件の 1 つに含めています。

一言でいえば、SBOM は組織がソフトウェア・アプリケーションのセキュリティ、コンプライアンス、および全体的な健全性を確保する上で不可欠な要素です。以下に、SBOM の利点をまとめます。

- **リスク管理：**ソフトウェア・サプライチェーンに存在するリスクを特定および管理できます。
- **脆弱性管理：**既知の脆弱性を短時間で特定して軽減できます。
- **ライセンス・コンプライアンス：**オープンソースおよびサードパーティのライセンスへのコンプライアンスを確保できます。
- **ソフトウェアの品質：**旧バージョンのコンポーネントやサポート活動の終了しているコンポーネントを特定できます。
- **合併・買収（M&A）：**M&A の際に、ソフトウェア・コンポーネントに関連する法的リスクや知的財産（IP）のリスクを評価できます。

- **セキュア・ソフトウェア開発：**米国サイバーセキュリティ・社会基盤安全保障庁（CISA）、SLSA（Supply Chain Levels for Software Artifacts）フレームワーク、NIST セキュア・ソフトウェア開発フレームワーク（SSDF）によって推奨されているセキュア・ソフトウェア開発プラクティスを強化できます。
- **ソフトウェアの効果的な開発、デプロイ、メンテナンス：**効率的なソフトウェア開発プロセスとライフサイクル管理が促進されます。
- **一貫性と可読性のある依存関係プロファイル：**アプリケーションの依存関係が標準に準拠した理解しやすい形で表現されます。
- **依存関係の標準化された目録と自動化：**依存関係を一貫性のある方法で目録にすることで、自動化が容易になります。

**SBOM を導入することで、ソフトウェア・サプライチェーンにおける自社開発コードとオープンソース・コードの可視性、透明性、セキュリティ、完全性が向上します。**

## SCA ツールはどのように SBOM を生成するのか

SCA ツールは、以下の機能を実行して SBOM を生成します。

- **コードのスキャン：**ソフトウェア・プロジェクトのソースコードまたはバイナリ・ファイルをスキャンしてすべてのコンポーネントと依存関係を特定します。これらのスキャナーは、以下のようにさまざまなスキャン手法を利用します。
  - **マニフェスト・スキャン：**マニフェスト・ファイル（package.json や Cargo.toml など）をチェックして、リスト化されている依存関係を調べます。
  - **バイナリ・スキャン：**コンパイルされたバイナリをチェックして、特定のライブラリにトレースバック可能なサードパーティ・コードの有無を調べます。
  - **ハイブリッド・スキャン：**マニフェスト・スキャンとバイナリ・スキャンを組み合わせて依存関係の取りこぼしをなくします。
  - **スニペット・スキャン：**ファイルの一部、または小さなコード・ブロック（スニペット）を解析し、これらを既知のオープンソース・コンポーネント全体のデータベースと照合します。
- **依存関係の解析：**直接依存関係や推移的依存関係も含め、コンポーネント間の関係を解析します。
- **脆弱性とライセンスの特定：**特定した各コンポーネントを脆弱性データベースやライセンス・リポジトリと照合して、潜在的なセキュリティ・リスクおよびライセンス・コンプライアンスの問題を特定します。
- **SBOM の生成：**これらの解析から得られた情報に基づき、SPDX や CycloneDX などの標準フォーマットの詳細な SBOM を作成します。
- **継続的監視：**SCA ツールの多くには、SBOM を常に最新の状態に維持するための継続的な監視機能があります。オープンソース・コンポーネントに新しい脆弱性が見つかった場合や、アップデートが利用可能になった場合、SCA ツールはそれに合わせて SBOM を更新し、常に正確性を維持します。

例えば、Black Duck® SCA をソフトウェア開発ライフサイクルに統合すると、ソフトウェアを開発しながら SBOM を生成できます。また、Black Duck SCA にはサードパーティ SBOM をインポートする機能もあり、これらのコンポーネントを関連するプロジェクトに追加したり、継続的にリスクを分析したり、アプリケーション・ライフサイクルの一部として生成されるレポートや SBOM に追加したりできます。ソフトウェア・デリバリ・パイプラインの最後に、解析したソフトウェアの SBOM を業界標準のファイル・フォーマットにエクスポートできます。

## 推移的依存関係の追跡

推移的依存関係は、あるソフトウェア・コンポーネントが別のコンポーネントに依存しており、そのコンポーネントがさらに別のコンポーネントに依存している場合に発生します。このような依存関係は複雑で、手動での追跡は困難ですが、SCA ツールを使用して SBOM を生成すると、すべての推移的依存関係を容易に特定および追跡でき、ソフトウェア・サプライチェーンを完全に可視化できます。

## 脆弱性、ライセンスと IP、およびコード品質の問題に対処する

SBOM があれば、セキュリティ侵害が起こる前に脆弱性を特定して対処できます。SBOM を分析することにより、セキュリティ・チームはサードパーティ・コンポーネントに含まれる脆弱性を特定し、それらが最新バージョンかどうかや、適切に設定されているかどうかを確認できます。この予防的なアプローチにより、セキュリティ侵害のリスクを減らし、ソフトウェアが堅牢な基盤の上に構成されていることを確認できます。

SBOM を使用すれば、すべてのコンポーネントがコンプライアンス要件を満たしているかどうかを確認でき、ソフトウェア・コンポーネントに関連するライセンスの追跡が容易になります。このように、SBOM はライセンス・コンプライアンスのリスクを管理して必要な義務を履行するためのよりオープンでスケーラブルなアプローチを可能にします。また、SBOM は旧バージョンのコンポーネントやサポート活動の終了しているコンポーネントなど、セキュリティ・リスクや性能上の問題を引き起こす可能性のあるコンポーネントの特定にも役立ちます。これらの情報は、アップデートに優先順位を付けるのに役立ちます。コードの品質上の問題を早期に特定して対処することで、セキュリティ侵害のリスクを減らし、ソフトウェアの性能と保守性を高めることができます。

## SBOM の採用

このように多くの利点がある SBOM は、セキュア・ソフトウェア開発プラクティスにおいてますます不可欠な存在となっています。SBOM の採用はまだ発展途上であり、採用率も業種や組織によって異なりますが、ソフトウェア・サプライチェーンのリスクを管理し、ソフトウェアのセキュリティを確保する上で SBOM が重要な役割を果たすという認識は高まっています。Linux Foundation が 2022 年に発表したレポートでは、SBOM を作成する企業の割合が 2022 年には 78% に達すると予測されています。Censuswide が 2023 年に実施した調査では、ベンダーに対して SBOM を要求している大企業（年間売上高 5,000 万米ドル超）は全体の 60% に達しています。

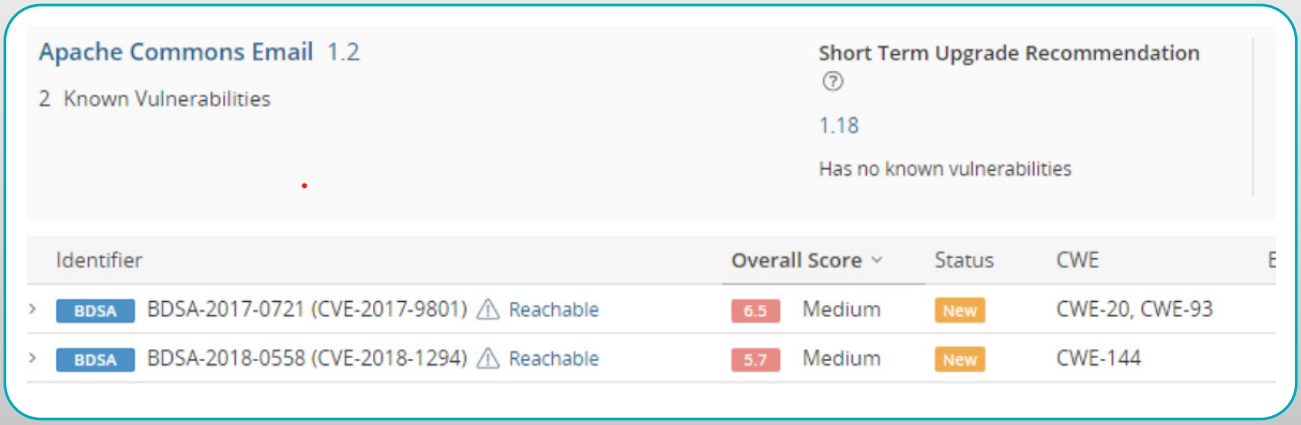
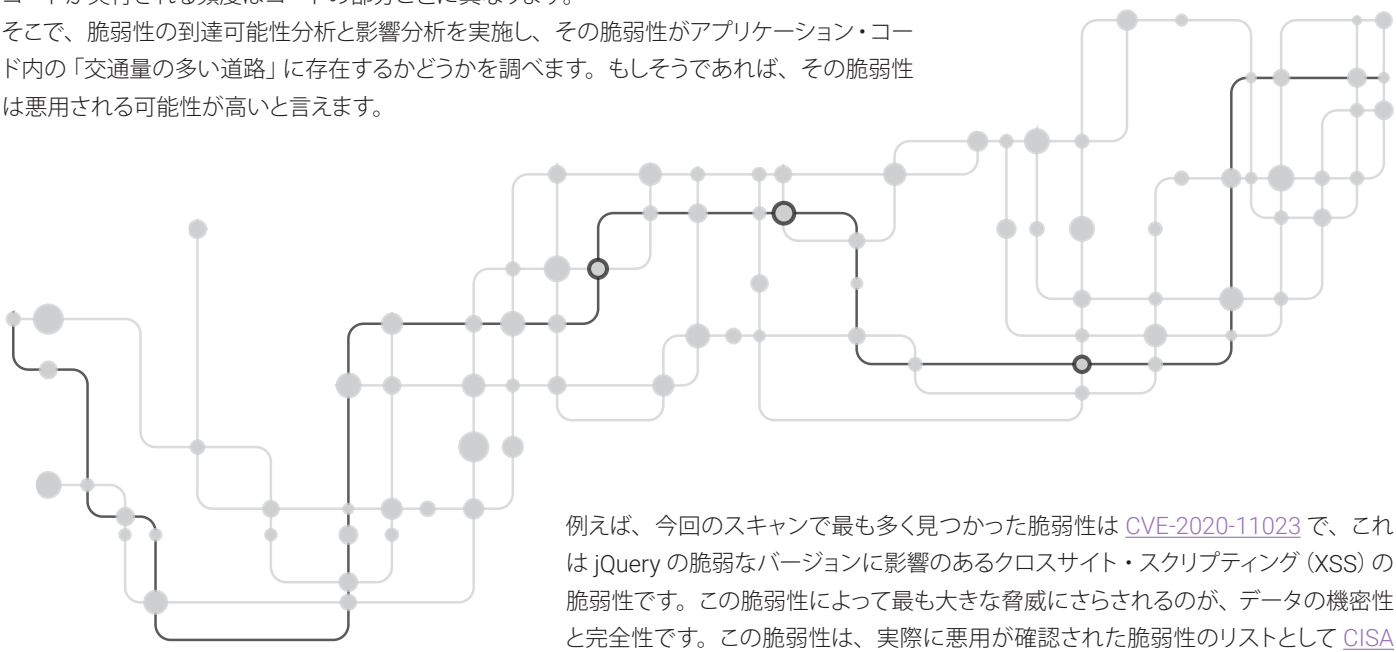



図 4：Black Duck SCA による脆弱性の影響分析

# 脆弱性の影響を分析する

SCA ツールを使用してアプリケーションに含まれる脆弱性を特定した後、どの脆弱性から対処すべきかを決めるにはどうすればよいでしょうか。脆弱性が存在するということは、潜在的な弱点があることを意味しますが、それが容易に悪用できるかどうか、すなわち悪意のある攻撃者がそれを利用してシステムやアプリケーション、ネットワークを容易に侵害できるかどうかは別問題です。

例えば、都市間の複雑な道路網を考えてみましょう。その中には、交通量の多い道路もあればほとんど使われていない道路もあります。ソフトウェア・アプリケーションの場合も同様に、コードが実行される頻度はコードの部分ごとに異なります。
そこで、脆弱性の到達可能性分析と影響分析を実施し、その脆弱性がアプリケーション・コード内の「交通量の多い道路」に存在するかどうかを調べます。もしそうであれば、その脆弱性は悪用される可能性が高いと言えます。



例えば、今回のスキャンで最も多く見つかった脆弱性は CVE-2020-11023 で、これは jQuery の脆弱なバージョンに影響のあるクロスサイト・スクリプティング（XSS）の脆弱性です。この脆弱性によって最も大きな脅威にさらされるのが、データの機密性と完全性です。この脆弱性は、実際に悪用が確認された脆弱性のリストとして CISA が公開している KEV（Known Exploited Vulnerabilities）カタログにも掲載されています。状況によっては、これらの要因を考慮してコードベース内の他の脆弱性よりもこの脆弱性への対処を優先させることもあるでしょう。

悪用可能性は、エクスプロイト・コードの有無、脆弱性悪用の複雑さ（攻撃の難易度）、悪用によって攻撃者が得られる見返りの大きさなど、さまざまな要因によって決まります。例えば、2014 年に見つかった Heartbleed 脆弱性は OpenSSL の暗号ライブラリに見つかった重大なセキュリティ上の欠陥で、攻撃者はこの脆弱性を容易に悪用して機微な情報を盗むことができたため、無数の Web サイトとサーバーが影響を受けました。有名なところでは、カナダ国税庁（CRA）も被害を受けており、この脆弱性がハッカーに悪用されてカナダ国民の社会保険番号（SIN）が流出したことが報告されています。CRA はこの問題に対処するために一時的にオンライン・サービスを停止し、確定申告の期限を延長することを余儀なくされました。Web セキュリティ企業の CloudFlare は、SSL 証明書の発行者が顧客の SSL 証明書を取り消して再発行するのにかかる費用を毎月約 40 万米ドルと見積もっています。

## SCA ツールの効果とは

SCA ツールは、重大度、悪用可能性、潜在的な影響度などさまざまな要因を考慮して脆弱性に優先順位を付けます。このため、最も重大度の高い脆弱性から優先的に対処し、修正作業を最適化することができます。また、無関係な問題はフィルターで除外されるため、膨大な指摘への対応に忙殺されるという困った問題も解決できます。単に脆弱性を特定するだけでは不十分です。なぜなら脆弱性の数は非常に多く、どの脆弱性を優先的に修正するかを理解するにはインテリジェントな方法が必要となるためです。

Black Duck SCA は、悪用可能性、修正ガイダンス、重大度スコア、コール・パス解析などの要因に基づいて脆弱性に優先順位を付けます。Black Duck は脆弱なコードが実際に実行される可能性を評価し、その可能性の高い脆弱性には「Reachable（到達可能）」のラベルを付け、優先的な修正が必要な脆弱性であることを示します。

# Log4j と Equifax：2 つの教訓から学ぶコード可視化の必要性

メディアの喧騒に埋もれて見えにくくなった面もありますが、2021 年の Log4Shell の脆弱性と 2017 年の Equifax のデータ流出事件はいずれも、自社が使用しているオープンソースを可視化することの重要性を再確認させてくれます。いずれの事例でも、被害拡大の要因となったのは、使用しているオープンソース・コンポーネントに対する意識の欠如でした。

Log4j の場合、多くの開発チームがどこでどのように Log4j を使用しているか、あるいはそもそもアプリケーションで Log4j を使用しているかどうかさえ把握していませんでした。このオープンソースのロギング・ユーティリティはアプリケーション内で何階層もの深い依存関係に埋もれており、基本的なコード・レビューではその存在を可視化できなかったことが主な原因です。CISA は連邦政府機関に対し、各機関が使用するソフトウェアに含まれる Log4j ライブラリのすべてのインスタンスを特定し、各システムが Log4Shell の悪用に対して脆弱かどうかを調べ、影響を受けるサーバーにはパッチを適用するという一連の作業を 10 日以内に完了させるよう指示しました。このため、多くのチームがクリスマス休暇を返上して脆弱なバージョンの Log4j を特定し、致命的なセキュリティ侵害となる可能性を未然に防ぐ作業に奔走することになりました。

| インシデント | 脆弱なコンポーネント | 被害拡大の要因 | 教訓 |
|---|---|---|---|
| Log4j | Log4j2 ライブラリ | ソフトウェア・サプライチェーンの可視化の欠如、アプリケーションで Log4j を使用していたことへの認識の欠如 | ログ出力ライブラリのように些細に見えるものも含め、アプリケーションに含まれるすべてのオープンソース・コンポーネントを把握することが必要 |
| Equifax | Apache Struts フレームワーク | IT 資産の網羅的な目録の欠如 | すべてのソフトウェア資産の正確な目録を維持することが必要 |

2017 年に Apache Struts の脆弱性悪用によって起こった Equifax のデータ流出事件は、事件発生から時が経ち、パッチ適用の必要性を訴えるべき責任者が適切な上層部に伝えていなかったことが原因であるかのように単純化して説明されていますが、この事件に関する議会報告書にもあるように、真の原因はそれよりもはるかに複雑です。

Equifax は当時、数年間にわたって買収を繰り返しており、そのたびに同社のテクノロジー・インフラストラクチャは複雑さと不透明さを増していました。この攻撃で主な標的とされた Web ベースの異議申し立て / 情報開示アプリケーションも、こうした継ぎはぎだらけの IT インフラストラクチャの 1 つでした。

Equifax の事件当時の CIO は、そのアプリケーションで使用していたソフトウェアを同社が明確に把握していなかったことを議会で証言していま

「脆弱なシステムにいつ、どのようにパッチを適用するかなど、情報に基づいた正確なリスク判定を行うには、IT 環境内にどのような資産が存在するかを組織として把握することが極めて重要である。」

—Equifax 社データ侵害に関する多数派報告書（第 115 議会、2018 年 12 月）より

す。ソフトウェアの構成内容を可視化できていなかったことは、事件の 2 年前にも同社の社内監査で報告されていた既知の問題でした。この監査レポートでは、次のように報告されています。「網羅的な IT 資産の目録もなければ、ネットワークも正確に文書化されていない。（中略）資産の正確な目録が存在しないため、すべての資産についてパッチ適用と構成が適切に行われていることを確認するのが困難であった。（中略）すべての IT 資産のステータスを十分に理解していないと、Equifax のシステムのセキュリティと安定性を確保することは極めて困難である」。

Equifax の問題がここまで深刻化したもう 1 つの要因は、このアプリケーションが同社の古いシステムの多くと同様に、いわゆる「レガシー」であったことです。2017 年時点では、この Web アプリケーションの内部動作に詳しい社員は Equifax にほとんど残っておらず、名目上このアプリケーションの管理を担当していた人物でさえこのアプリケーションに Apache Struts ソフトウェアが含まれていることを知りませんでした。

Equifax の事例は、オープンソースの可視性の欠如についてまた別の側面を浮き彫りにしています。それは、「仮に自社のコードを把握できていたとしても、他社のコードの中身まで把握できていますか」という問いかけです。他社のコードに依存している企業なら、このことは心に留めておく必要があります。

## 最も多く見つかった高リスクおよび重大なリスクのある脆弱性

今回の監査で見つかった高リスクの脆弱性について詳しく見ていく前に、CVE、CWE、BDSA という用語の意味を明確にしておきましょう。CVE（共通脆弱性識別子）とは、公開された既知の脆弱性に標準識別子を与えたものです。新しい脆弱性が見つかると、その脆弱性に一意の CVE ID が付与されるため、セキュリティ専門家や開発者は特定の脆弱性に関する情報をすばやく簡単に参照して追跡することができます。NVD（脆弱性情報データベース）は、CVE 標準に基づいて脆弱性を特定および記述しています。

これに対し、CWE（共通脆弱性タイプ一覧）とは、ソフトウェアおよびハードウェアの弱点をタイプごとにまとめたリストで、コミュニティによって作成されています。CWE はセキュリティ上の弱点を記述する共通の言語となるもので、弱点の特定、軽減、および予防に役立ちます。図 5 は、今回の監査で見つかった脆弱性の原因となっている主な CWE をまとめたもので、この図から有益な情報を読み取ることができます。

例えば、今回の監査で見つかったオープンソース脆弱性の 70% 以上が不適切な入力確認（インジェクションや XSS 攻撃につながる可能性のある弱点）に関連付けられています。このため、開発およびセキュリティ・チームは自社開発コードに対して適切な検証手法の使用に注力するとともに、サードパーティ・コードに対しては SAST および DAST を使用して定期的にテストを実施し、脆弱性の早期発見を継続する体制を整える必要があります。

| CWE | この CWE に関連付けられる脆弱性が見つかったコードベースの割合 | | 説明 |
|---|---|---|---|
| CWE-20 | 71% |  | **不適切な入力確認：**制御フローまたはデータ・フローに影響する可能性のある入力をソフトウェアがまったく検証していないか、適切に検証していません。この弱点は、バッファー・オーバーフロー、SQL インジェクション、クロスサイト・スクリプティングなどの脆弱性につながることがあります。 |
| CWE-400 | 70% |  | **リソースの枯渇：**メモリ、CPU 時間、ディスク領域などのシステム・リソースの割り当てと解放をソフトウェアが適切に制限していません。この弱点は DoS 攻撃につながることがあります。 |
| CWE-200 | 60% |  | **情報漏えい：**ソフトウェアが、パスワード、クレジットカード番号、個人データなどの機微な情報を権限のない者に晒しています。これは、安全でないデータ保存、平文での送信、不適切なアクセス制御など、さまざまな形で発生します。 |
| CWE-79 | 56% |  | **クロスサイト・スクリプティング：**ユーザーの入力を HTML ページに含める前にソフトウェアがまったくニュートラライズしていないか、適切にニュートラライズしていません。これにより、攻撃者は悪意のあるスクリプトを挿入してユーザー・データを盗んだり、ユーザーのブラウザーを乗っ取ったりできます。 |
| CWE-185 | 48% |  | **不正な正規表現：**ソフトウェアが正規表現を不適切に指定しており、データのマッチングや比較が正しく行われていません。正規表現は、同値分割、境界値分析、ロバストネス・テストなどの手法を使用して徹底的にテストを行う必要があります。 |
| CWE-770 | 48% |  | **制限またはスロットリングなしのリソースの割り当て：**ソフトウェアが制限やスロットリングなしにリソースを割り当てており、攻撃者が過剰にリソースを使用して DoS 攻撃を引き起こす可能性があります。 |
| CWE-80 | 44% |  | **クロスサイト・スクリプティング (Basic XSS)：**この弱点は CWE-79 と似ており、スクリプトのインジェクションに使用可能な HTML タグをニュートラライズできていないという弱点です。 |
| CWE-1321 | 39% |  | **オブジェクトプロトタイプ属性の不適切に制御された変更 ( プロトタイプの汚染 )：**ソフトウェアが攻撃者によるオブジェクトのプロトタイプ変更を許しているという弱点で、そのオブジェクトのすべてのインスタンスが影響を受け、予期しない動作や権限昇格を引き起こす可能性があります。 |
| CWE-1333 | 36% |  | **非効率な正規表現の複雑さ：**ソフトウェアがあまりにも複雑な正規表現を使用しており、CPU の過剰な使用や DoS 攻撃につながることがあります。 |
| CWE-502 | 31% |  | **信頼できないデータのデシリアライゼーション：**ソフトウェアが信頼できないデータをデシリアライズしており、攻撃者による任意のコードの実行や、アプリケーション・ロジックの操作を許してしまう可能性があります。 |

図 5：スキャンしたコードベースで最も多く見つかった CWE

このレポートに示した Black Duck Security Advisory（BDSA）は、ブラック・ダックの CyRC が収集、整理している Black Duck ユーザー専用の脆弱性データ・フィードです。BDSA は、幅広い脆弱性に関して脆弱性データベース（NVD：National Vulnerability Database）よりも深いレベルまでカバーしています。重大度、影響度、悪用可能性の指標など、脆弱性に関するより詳細な情報をいち早く提供するだけでなく、BDSA には実践的な修正ガイダンスも含まれており、修正済みバージョン、パッチ情報、エクスプロイト、利用可能な回避策などの詳細な情報により、短時間での修正が可能です。

脆弱性に関するガイダンスは、NVD が CVE レコードとして提供しているものよりも、CyRC チームが提供しているものの方が充実しています。また、BDSA は影響を受ける可能性のあるコンポーネントのバージョンともクロスチェックおよび検証されており、特定の脆弱性によって影響を受けるコンポーネントとバージョンの対応関係をより多く、より正確に特定しています。

それでは、今回のスキャンで最も多く見つかった高リスクおよび重大なリスクのあるオープンソース脆弱性について詳しく見ていきましょう。

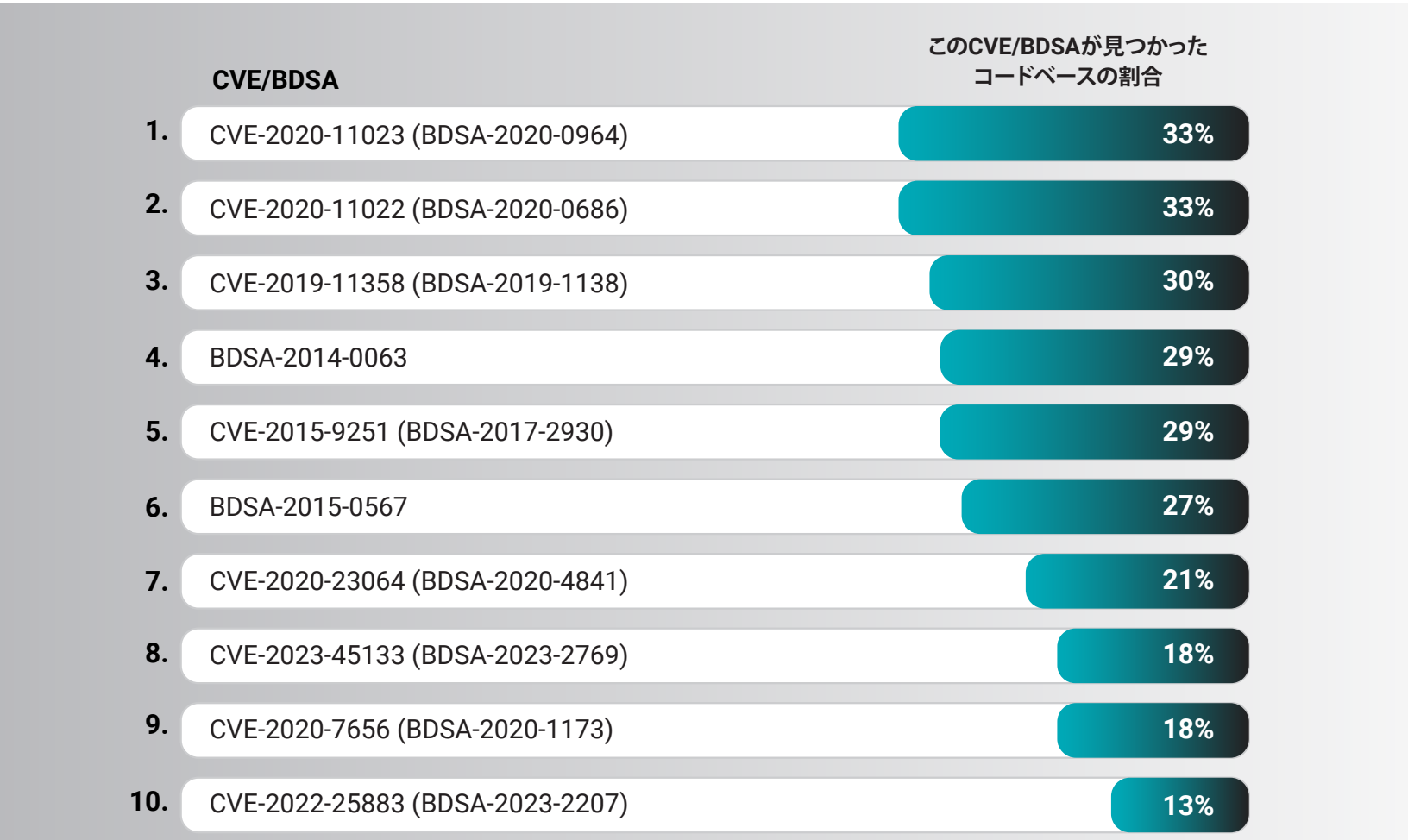



図 6：最も多く見つかった高リスクおよび重大なリスクのある脆弱性

## 1. CVE-2020-11023

CVE-2020-11023 は、スキャンしたコードベースのほぼ 1/3（32.6%）で見つかりました。これは、jQuery バージョン 1.0.3 以上 3.5.0 未満に影響する XSS の脆弱性です（本レポート発行時点での jQuery の最新の安定版バージョンは 2023 年 8 月にリリースされた 3.7.1）。CVE-2020-11023 は、CISA の KEV カタログに掲載されており、実際に悪用された事例が存在します。

この脆弱性があると、信頼できないソースからの <option> 要素を含む HTML を jQuery が処理する方法を操作することにより、信頼できないコードが実行されることがあります。この脆弱性の影響は非常に広範囲に及び、Debian Linux、Fedora、Drupal、Oracle 製品など jQuery を使用しているさまざまなシステムに影響します。

CVE-2020-11023 に関連する CWE は、「CWE-79：クロスサイト・スクリプティング」です。この CWE は、ここに挙げた CVE の多くが関連するセキュリティ上の問題で、ユーザーからの入力を適切にサニタイズしないで Web ページに取り込んでいるという弱点です。潜在的に悪意のある入力をニュートラライズしていないと、攻撃者はユーザーのブラウザーに悪意のあるスクリプトを注入して実行できます。

## 2. CVE-2020-11022

CVE-2020-11022 も jQuery に存在する XSS の脆弱性で、バージョン 1.2 以上 3.5.0 未満に影響します。この欠陥があると、攻撃者は jQuery の DOM 操作メソッドが入力を処理する方法を悪用して、JavaScript コードを Web ページに注入することができます。CVE-2020-11023 とは異なり、この脆弱性は CISA の KEV カタログには掲載されておらず、重大度の低いリスクと考えることができます。

CVE-2020-11022 に関連する CWE も、「CWE-79：クロスサイト・スクリプティング」です。

## 3. CVE-2019-11358

CVE-2019-11358 は、バージョン 3.4.0 未満の jQuery に存在するプロトタイプ汚染の脆弱性です。プロトタイプの汚染は、攻撃者がオブジェクトのプロトタイプを操作すると発生し、そのプロトタイプを継承するすべてのオブジェクトに影響する可能性があります。これは、予期しない動作、サービス拒否、または任意のコード実行につながることがあります。

CVE-2019-11358 に関連する CWE は、「CWE-1321：オブジェクトプロトタイプ属性の不適切に制御された変更 ( プロトタイプの汚染 )」です。この CWE は、プログラム実行時のコード・リソース管理に関するさまざまな弱点を 1 つのカテゴリにまとめた「CWE-913：動的に操作されるコードリソースの不適切な制御」の子に当たる CWE です。

## 4. BDSA-2014-0063

これは 2014 年 1 月に初めて問題として報告されたかなり古い脆弱性で、ユーザー入力を検証していない場合に jQuery に存在する潜在的な XSS の脆弱性に関するものです。この脆弱性には、対応する CVE がありません。

この脆弱性により、攻撃者は任意の Web スクリプトを挿入し、攻撃対象のセッション・クッキーを盗むことが可能になります。この脆弱性は jQuery 3.0.0-rc1 で軽減されました。しかし、この軽減策は悪意ある入力をサニタイズしておらず、依然としてスクリプトの実行を許してしまいます。このパーサーは、コンテキストが指定されていない、または Null/ 未定義として与えられた場合、新規ドキュメントを作成するようにデフォルトの動作が変更されています。これにより、解析された HTML はドキュメントに挿入されるまで実行されなくなり、作成された DOM をツールがトラバースして、関数呼び出しの後に安全でないコンテンツを削除できるようになっています。

ここに挙げたいくつかの脆弱性と同様に、BDSA-2014-0063 も「CWE-79：クロスサイト・スクリプティング」に関連します。

## 5. CVE-2015-9251

CVE-2015-9251 は jQuery に存在する XSS の脆弱性で、3.0.0 未満のバージョンに影響します。(本レポート発行時点での jQuery の最新の安定版バージョンは 2023 年 8 月にリリースされた 3.7.1)。

この脆弱性は、クロスドメインの AJAX リクエストを行う際に dataType option オプションを指定していないと発生し、悪意のある JavaScript レスポンスが実行されることがあります。CVE-2015-9251 に関連する主な CWE は、「CWE-79：クロスサイト・スクリプティング」ですが、「CWE-94：コードインジェクション」および「CWE-74：インジェクション」にも関連することに注意してください。

## 6. BDSA-2015-0567

これも jQuery に存在する古い脆弱性（任意のコード実行の脆弱性）で、対応する CVE はありません。パッチ未適用の UglifyJS パーサーを使用するバージョンの jQuery には、細工された JavaScript ファイルを使用して任意のコードを実行される脆弱性があります。最終的には、攻撃者による不正なコード実行が可能になります。この脆弱性は、1.12.0 および 2.2.0 で修正されました。この BDSA に関連する CWE は、「CWE-1395：脆弱なサードパーティ製コンポーネントの使用」です。

## 7. CVE-2020-23064

CVE-2020-23064 は、jQuery バージョン 2.2.0 から 3.x（3.5.0 未満）に存在する XSS の脆弱性です。この脆弱性があると、攻撃者は <options> 要素の処理を悪用して任意のコードを実行できます。この脆弱性の CVSS v3 ベース・スコアは 6.1 で、重大度は「中」に分類されています。

CVE-2020-23064 に関連する CWE は、「CWE-79：クロスサイト・スクリプティング」です。

## 8. CVE-2023-45133

CVE-2023-45133 は、JavaScript コンパイラとして人気のある Babel に存在する脆弱性です。これは @babel/traverse パッケージ、およびすべてのバージョンの babel-traverse に影響します。この欠陥を悪用するように細工された悪意のあるコードを処理しようとすると、コンパイル時に任意のコードが実行されることがあります。

CVE-2023-45133 に関連する CWE は、「CWE-697：不適切な比較」と「CWE-184：不完全なブラックリスト」の 2 つです。これらの CWE は、コンパイル時に特定の入力を適切に検証および処理していないと脆弱性が発生することを示しています。

## 9. CVE-2020-7656

CVE-2020-7656 も jQuery に存在する XSS の脆弱性で、1.9.0 未満のバージョンに影響します。この脆弱性は、load メソッドが空白文字を含む <script> HTML タグを適切に処理・除去できないために発生し、悪意のあるスクリプトが実行されてしまう可能性があります。

CVE-2020-7656 に関連する CWE は、「CWE-79：クロスサイト・スクリプティング」です。

## 10. CVE-2022-25883

CVE-2022-25883 は、特定の Node.js システムで使用される semver パッケージに存在する、正規表現によるサービス拒否（ReDoS）の脆弱性です。この脆弱性は特定のバージョンの semver パッケージに影響し、信頼できないユーザー・データを範囲として処理した場合に引き起こされます。ReDoS 攻撃は正規表現に存在する脆弱性を悪用して過剰な処理時間を発生させるもので、システム・リソースの枯渇によるサービス拒否につながる可能性があります。

CVE-2022-25883 に関連する CWE は、「CWE-1333：非効率的な正規表現の複雑さ」です。この脆弱性はシステムの性能と安定性に大きな影響を与え、サービスやアプリケーションの停止を招く可能性があります。

# データが語ること

CWE を理解することは、開発者とセキュリティ専門家の両方にとって非常に重要です。ソフトウェアの弱点を標準的な方法でカテゴリ分類して記述した CWE を使用することで、意思疎通を図りながらセキュリティ・リスクに共同で対処することが容易になります。よくある弱点を理解することにより、開発者はセキュア・コーディング・プラクティスを実践して脆弱性を防ぐことができるようになり、セキュリティ・チームは潜在的な脅威を効果的に特定して軽減することができます。

今回の調査結果で CWE-79 の弱点が多く存在していたこと、そして jQuery の脆弱性を悪用したクロスサイト・スクリプティング攻撃に関する CVE も多く見られたことは、Web 開発において入力バリデーションが極めて重要であることを物語っています。ユーザー入力を適切にサニタイズしていないと、深刻な結果を招く可能性があります。

jQuery そのものが安全でないということではありません。事実、jQuery は十分にメンテナンスされたオープンソース・ライブラリで、多数のユーザー、開発者、メンテナーによるコミュニティが形成されています。しかし今回の監査で最も多くの脆弱性が見つかったコンポーネントが jQuery で、スキャンした全コードベースの約 1/3 が脆弱な jQuery コンポーネントを使用していました。これらの脆弱性は旧バージョンの jQuery に影響するものばかりで、しかも利用可能なパッチが存在しているにもかかわらず、です。jQuery に限らずすべてのオープンソースのユーザーに言えることですが、旧バージョンのソフトウェアに存在する潜在的なセキュリティ・リスクを意識し、これらのリスクを軽減するための対策をとることが重要です。

開発者は、クロスサイト・スクリプティングやその他のインジェクション攻撃を防ぐために入力バリデーションやサニタイズなどの手法を優先的に実施することの重要性を今回のデータから読み取ることができます。Coverity® 静的解析や Black Duck® Continuous Dynamic（本番環境でも利用可能な DAST ツール）などのセキュリティ解析ツールを使用すれば、フォームへの入力や API パラメーターなどユーザーが送信したデータを適切にチェックできていないことから生じる潜在的な脆弱性を特定できます。これにより、想定外のデータ・フォーマットや値を受け取るのを防ぎ、SQL インジェクションやクロスサイト・スクリプティングなどさまざまなインジェクション攻撃のリスクを軽減できます。

セキュリティ・アドバイザリで常に最新情報を収集し、脆弱なソフトウェアには迅速にパッチを適用することは、悪用のリスクを最小に抑える上での基本です。jQuery や Babel などのライブラリやフレームワークを定期的にアップデートすることは、既知の脆弱性からシステムを保護する上で非常に重要なことです。

# 業種別の分析結果

図 1（p.5）に示したように、脆弱性の観点から特にリスクの高い業種は、インターネット / モバイル・アプリ（スキャンしたコードベースの 100% に高リスクの脆弱性が存在）、マーケティングテック（同 88%）、コンピュータ・ハードウェア / 半導体（同 87%）、エドテックおよびエンタープライズ・ソフトウェア /SaaS（同 86%）でした。

全 16 業種のうち、その割合が最も低かったエネルギー / クリーンテックでさえ、コードベースの 60% に高リスクの脆弱性が存在しています。このように、業種を問わずゆうに半分以上のコードベースに高リスクの脆弱性が含まれていることは、攻撃者にとって格好の標的となっていることを意味しています。

# オープンソース・ライセンス

「プログラマーたる者は、テクノロジーだけでなく
法律も理解する必要がある」

– エリック・オールマン（プログラマー）

Black Duck 監査では、オープンソース・ライセンスのコンプライアンス診断を必ず実施しています。2024 年に Black Duck 監査チームは 965 件の監査を実施しました。このセクションのデータは、これらの診断結果に基づいています。

- ライセンスの競合が見つかったコードベースの割合：56%
- ライセンスのない、またはカスタム・ライセンスのオープンソースを使用しているコードベースの割合：33%

オープンソースを効果的に管理するには、セキュリティだけでなくライセンスへのコンプライアンスも必要です。使用しているオープンソース・コンポーネントやライブラリにライセンスが適用されることは知っていても、果たしてそのライセンスの詳細まで理解できているでしょうか。さらに重要な点として、経営幹部や顧問弁護士が自社開発ソフトウェアとの関連においてオープンソース・ライセンスの詳細に意識を向けているでしょうか。

ソフトウェアの中にライセンスへのコンプライアンス違反が 1 つあるだけで、法的な問題、収益源となるはずの知的財産権の喪失、時間のかかる修正作業、さらには製品の市場投入の遅れなどを招くことがあります。今回の監査結果でも、顧客のコードベースの 56% にライセンスの競合が見つかっており、これらの顧客はこうした事態に直面する可能性があります。

## ライセンスの競合、派生ライセンス、ライセンスの欠如がリスク要因となる理由

米国やその他の裁判管轄地では、ソフトウェアなどの著作物は無方式主義により独占的に著作権保護されます。著作者からライセンスという形で明示的に権利を付与されない限り、他者がソフトウェアを使用、複製、頒布、改変することは違法となります。

オープンソース・ソフトウェアに関して言えば、あるオープンソース・コンポーネントのライセンスがプロジェクトまたはコードベース全体に対して宣言した全体のライセンスと衝突する場合に、ライセンス宣言の競合が発生します。これがよく起こるのは、GNU General Public License（GPL）のような制限の強いライセンスのコンポーネントを商用プロジェクトに組み込んだケースで、この場合、プロジェクト全体のソースコードの開示が要求されることがあります。ライセンス宣言の競合の重大度は一概には言えず、ライセンスの適用範囲によってプロジェクト全体に影響が及ぶこともあれば、特定のファイルにしか影響しないこともあります。また、1 つのプロジェクト内で 2 つのオープンソース・ライセンスの間に互換性がない場合は、コンポーネント・ライセンス間の競合が発生します。

ほとんどのオープンソース・ライセンスには、ライセンスされたコードのごく一部（スニペット）をプロジェクトに組み込んだ場合にも競合が発生するような文言があります。これまで、このような競合は開発者がライセンスに問題のあるオープンソース・プロジェクトからコードをコピー & ペーストして使用した場合に発生するのが一般的でした。しかし現在では、オープンソースを使用して学習した生成 AI モデルの台頭により、AI ツールがライセンスを無視してスニペットを盗用するケースが見られるようになっています。

標準的なオープンソース・ライセンスではなく、その派生ライセンスや一部をカスタマイズしたライセンスの場合も、ライセンシーにとって望ましくない条件が課せられることがあるため、知的財産権（IP）の問題やその他の影響について法的な立場からの評価が必要になります。カスタマイズしたライセンスの例としてよく挙げられるのが、JSON ライセンスです。JSON ライセンスは、ベースとなっている寛容型の MIT ライセンスに「このソフトウェアは善い目的に使用されるべきで、邪悪な目的に使用してはならない」という文言を追加しています。その精神には感心しますが、この曖昧な文言はさまざまな意味に解釈できるため、特に M&A に関係する場合は、多くの弁護士が JSON ライセンスのソフトウェアを使用しないように助言しています。2016 年以降、Apache Software Foundation はすべての Apache プロジェクトにおいてこのライセンスを使用したソフトウェアの使用を禁止しています。

2024 年に監査したコードベースのうち、ライセンスの存在を確認できない、またはカスタム・ライセンスのコードを使用したものが 33% ありました。開発者がサービス利用条件を明示せず、またはソフトウェア利用条件に言及せずにコードを公開することはよくあります。また、開発者が標準ライセンス条件の書き換えや文言の追加（上記の「善い目的、悪い目的」など）によって、あるいは使用、義務、制限に関する条件をコードのコメントに追加することによってコードの使用を許諾することも珍しくありません。多くの場合、このようなタイプの改変には法務上の精査が必要です。

## 推移的依存関係がライセンス競合に与える影響

今回の監査で見つかったコンポーネント・ライセンス間の競合の 30% 近くは、推移的依存関係（直接依存関係および全体的なソフトウェアが機能するために必要なコンポーネント）が原因で発生していました。推移的依存関係が GPL のような制限の強いライセンスを使用している場合、直接依存関係がより寛容なライセンスを使用していたとしても、アプリケーション全体のライセンスに影響する可能性があります。制限の強いライセンスの多くは、派生した著作物に対しても同じライセンス条件を適用することを要求するためです。

## 2024 年に見つかったオープンソース・ライセンス Top 10

| ライセンス | このライセンスを含むコードベースの割合 | リスク * | OSI approved |
|---|---|---|---|
| MIT License | 92% | 低 | はい |
| Apache License 2.0 | 90% | 低 | はい |
| 3 条項 BSD（「新 BSD」または「修正 BSD」）License | 85% | 低 | はい |
| 2 条項 BSD（「簡易 BSD」）License | 74% | 低 | はい |
| ISC License | 61% | 低 | いいえ |
| 一般的なパブリック・ドメイン | 57% | 用途による | はい |
| GNU Lesser General Public License v2.1 以降 | 48% | 高 | はい |
| Unlicense | 47% | 低 | はい |
| Creative Commons Zero v1.0 Universal | 46% | 用途による | いいえ |
| Mozilla Public License 2.0 | 45% | 中 | はい |

* このリスク分類は単なるガイドラインであり、オープンソース・ソフトウェアの使用可否の最終判断の材料となるものではありません。ライセンス・コンプライアンスについては、社内のポリシー / 法務チームに相談してください。

図 7：オープンソース・ライセンス Top 10

# オープンソース・ライセンスの種類：寛容型、弱いコピーレフト、互恵型の違いとは

## 低リスク：寛容型ライセンス

一般に、寛容型ライセンスにはそれほど多くの制限条項がありません。自分のソフトウェアを頒布する際に、著作権表示を適切に行っていればそれで十分なことがほとんどです。つまり、著作権表示をそのまま残していれば、オープンソース・ソフトウェアを使用および改変できます。現在使用されている最もポピュラーなライセンスは MIT ライセンスと Apache ライセンスの 2 つで、これらはいずれもこのカテゴリに属します。ブラック・ダックでは、寛容型ライセンスを低リスク・ライセンスに分類しています。

## 中リスク：弱いコピーレフト・ライセンス

一般に、コピーレフト・ライセンスには、改変および拡張したバージョンも元のコードと同じ条件でリリースすることが要求される互恵型の義務が含まれます。弱いコピーレフト・ライセンスでは通常、ソースコードに対して改変を加えた場合、そのコードを元のライセンスと同じ条件で頒布することが要求されます。これらのライセンスの中には、何が改変に当たるかを明示的に定義しているものもあります。例えば、オープンソース・コードを改変せずに自社開発コードにコピーすることも改変としているライセンスもあります。このようなライセンスの義務に従うには、ソースコード（元のコード、改変したコード、および新規に追加したコード）を開示する必要があります。このカテゴリに属する有名なオープンソース・ライセンスとしては、Mozilla Public License があります。ブラック・ダックでは、弱いコピーレフト・ライセンスを中リスク・ライセンスに分類しています。

## 高リスク：互恵型 / コピーレフト・ライセンス

有名なオープンソース・ライセンスの中でも、GNU General Public License v2.0 以降や GNU Lesser General Public License v3.0 以降などは非常に制限の強いライセンスです。オープンソース・ソフトウェアを自社開発ソフトウェアに組み込む方法によっては、大きなリスクに直面する可能性もあります。最悪の場合、自社で開発したソフトウェアに同じライセンスを適用してロイヤリティ・フリーでリリースすることが求められることもあります。ブラック・ダックでは、こうした制限の強いライセンスを高リスク・ライセンスに分類しています。

# オープンソース・ライセンスのリスクを SCA で管理するには

パッケージ・ソフトウェア、組み込みソフトウェア、あるいは商用の SaaS ソフトウェアを開発している組織にとって、オープンソース・ライセンスのコンプライアンスには十分な注意を払う必要があります。使用しているオープンソース・コンポーネントのライセンス・タイプと条件を特定し、それらが自社ソフトウェアのパッケージングおよび頒布と互換性があることを確認することが必要です。商用製品ではなく、社内のみで使用するソフトウェアであっても、ソフトウェアに含まれるオープンソース・コンポーネントのライセンス条件に従う必要があります。

リスク管理の第一歩は、自動化された SCA ツールを使用して、ソフトウェアに含まれるすべてのオープンソース・コンポーネント、および使用しているバージョンと関連ライセンスについての最新の情報を反映した正確な SBOM を生成することから始まります。これらのコンポーネントに関連するライセンスの文言を集約し、自社のソフトウェア頒布およびライセンス条件と両立しないコンポーネント、または自社のソフトウェアに含まれる他のコンポーネントのライセンスと両立しないコンポーネントがあればすぐに指摘できるようにしておきます。どれほど寛容なオープンソース・ライセンスでも著作権表示の義務はあるため、すべてのライセンスの義務を満たしていることを確認する必要があります。

Black Duck SCA を使用すると、開発、セキュリティ、コンプライアンスの各チームがオープンソースの使用に起因するリスクを管理できます。Black Duck のマルチファクター・オープンソース検出、および 780 万を超えるコンポーネントを登録した KnowledgeBase により、あらゆるアプリケーションやコンテナに対してライセンス情報を含む正確な SBOM を生成できます。オープンソース・コンポーネントは有名なライセンスを使用していることがほとんどですが、それ以外にも、Black Duck は自社開発ソフトウェアに制限を課す可能性のあるオープンソース・ライセンスのデータを 2,500 件以上登録するなど、より一層の情報を提供しています。Black Duck を使用してオープンソースを追跡および管理してライセンスの問題を回避することにより、訴訟による巨額の損失を防ぎ、重要な知的財産を保護することができます。

# 業種別に見たライセンス競合

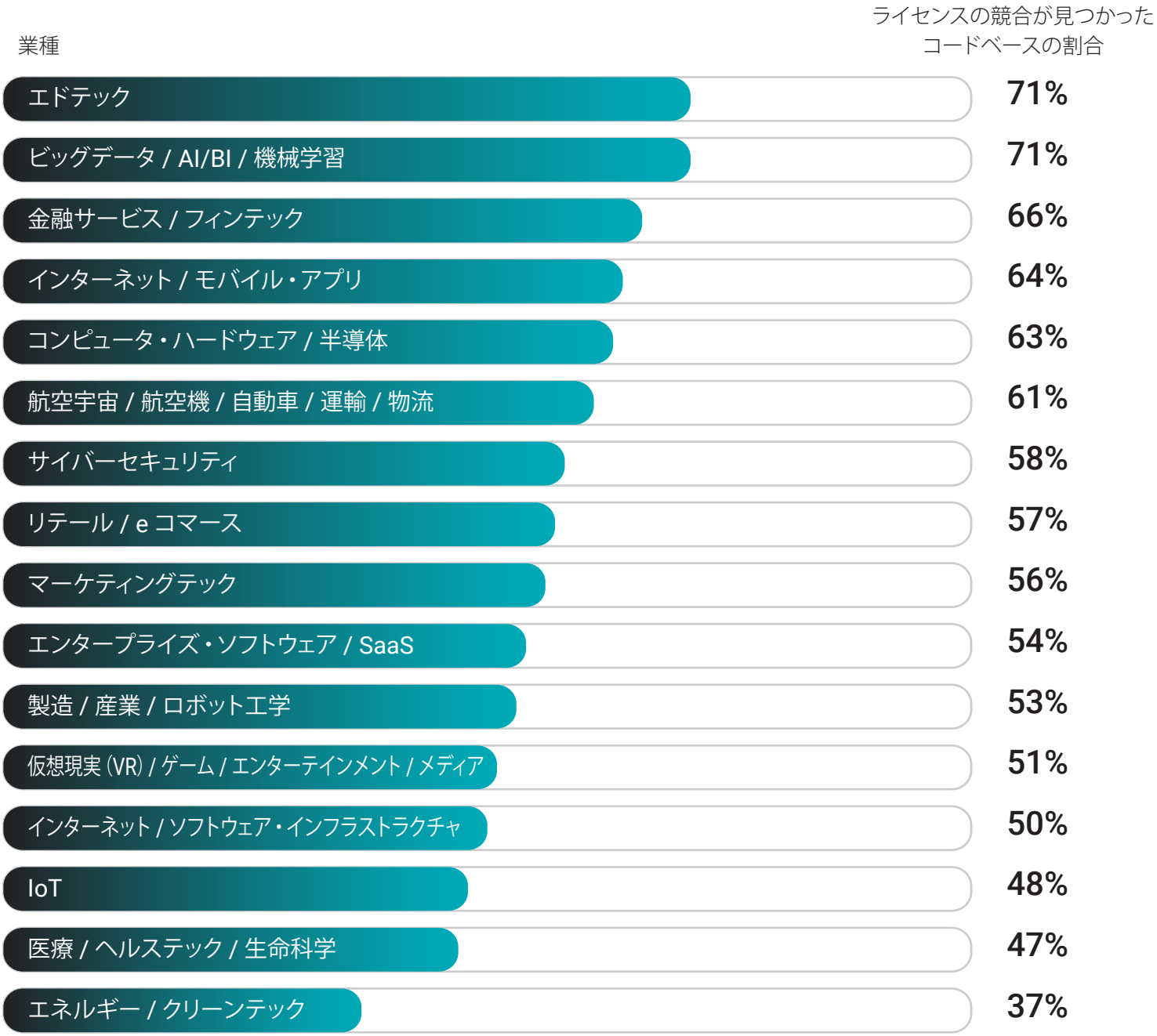



図 2：ライセンスの競合が見つかったコードベースの業種別割合

図 2 (p.6 にも掲載) は、コードベースにどれだけライセンスの競合が蔓延しているかを業種別に示したものです。ソフトウェアのライセンス競合によって発生する法的および運用面の問題とそれに伴うコストを防ぐために、事前にライセンスを特定することがいかに重要であるかがこの図から読み取れます。

## 高リスクの業種

- ビッグデータ /AI、金融サービス / フィンテック、コンピュータ・ハードウェアなどのハイテク業界では、ライセンス競合を含むコードベースの割合が非常に高くなっています。その理由として、これらの業種はソフトウェアとサービスに大きく依存しており、さらにそれらのアプリケーションがオープンソース・コンポーネントに依存していることが考えられます。
- これらの業種では、ソフトウェアをオンプレミス製品としてライセンス供与して頒布する傾向もあります。制限の強いライセンスのほとんどは、このような形態で頒布されるソフトウェアに適用されます。ライセンス競合の割合の低い業種はサブスクリプション型や SaaS 型のデプロイが多い傾向にあり、こうした形態は伝統的に「頒布」とは見なされず、同じライセンス条件は適用されません。
- エドテックも非常に高い割合となっているのは意外ですが、これは教育用ソフトウェアおよびプラットフォームにライセンスの問題が多く潜んでいることを示しています。オンライン学習やデジタル教育ツールの台頭により、エドテック業界はここ数年で急成長しています。スタートアップや中小企業など、ソフトウェア・ライセンスの問題を十分に扱えるだけのリソースとノウハウを持たないエドテック企業が多く存在します。

## 中リスクの業種

- 航空宇宙、サイバーセキュリティ、製造、エンタープライズ・ソフトウェアなどは、中リスクの業種に分類されます。これらの業種は、高リスクの業種に比べればライセンス競合の割合は低いものの、それでもかなり高い確率でライセンスの問題が見つかっています。

## 低リスクの業種（ただしノーリスクではない）

- 医療、エネルギーなどの業種では、ライセンスの競合があまり見られませんでした。しかし、これらの企業がライセンス競合の問題と無縁というわけではありません。例えば、医療業界の企業は電子カルテや医療画像システム、遠隔医療プラットフォーム、AI 支援型診断ツールなど幅広いソフトウェアを利用しています。このような複雑なエコシステムでは、サードパーティのコンポーネントやライブラリを多数統合することも多く、ライセンスの問題が起こる可能性が高くなります。

# M&A を計画しているなら

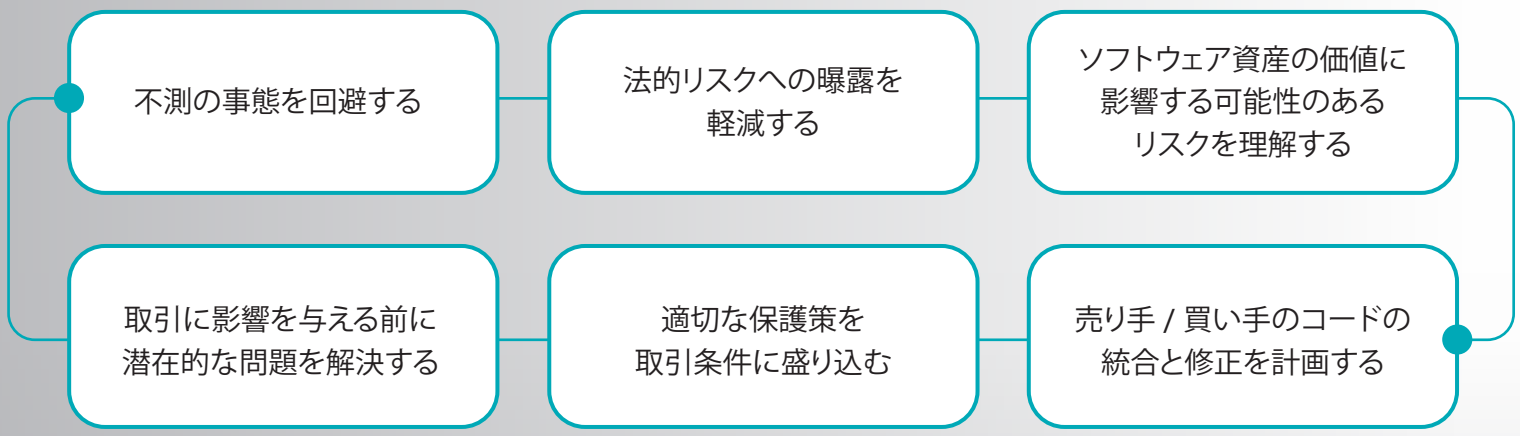
ライセンス条件を十分に理解し、さまざまなライセンス間の競合を特定するのは容易ではないため、合併・買収（M&A）取引に売り手または買い手として関与する計画があるなら、組織内の知的財産権専門家に関与してもらうか、外部の法律専門家の助言を求めることを推奨します。特にパッケージ・ソフトウェアや組み込みソフトウェアを開発している場合、出荷されるソフトウェアにはライセンス条件がより明確に適用されることが多く、出荷後にライセンスの問題を軽減することは困難なため、初手を間違えないことが重要です。企業のコードベースにどのようなオープンソース・コードが含まれているかを知ることは、その利用と再利用を適切に管理し、ソフトウェア・ライセンスへのコンプライアンスを確保し、脆弱性にいち早くパッチを適用する上で非常に重要なことであり、これらはいずれもビジネス・リスクを軽減するための基本的な事項です。

テクノロジー企業の M&A 取引において買い手側となる場合は、ソフトウェア・デュー・ディリジェンスの一環としてオープンソース監査を実施することが重要です。コード監査によって、買い手は知的財産の価値に影響する可能性のあるリスクがソフトウェアに潜んでいるかどうかを把握し、こうしたリスクに対処するにはどのような修正が必要かを理解できます。オープンソース監査は、コードの構成要素をより正確に理解しようと考えている企業にとっても大きな価値があります。例えば、専門の監査チームは Black Duck SCA などさまざまなツールを利用してコードベースに含まれるオープンソース・コンポーネントをもれなく特定し、これらのコンポーネントに法的なコンプライアンスの問題があれば指摘し、重大度に基づいた優先順位付けを行います。

監査では、オープンソース・コンポーネントに影響するセキュリティ上の既知の脆弱性だけでなく、バージョン、重複、およびコンポーネントの開発活動状態といった情報も明らかになります。また、買収対象企業のソフトウェア開発プロセスの洗練度についての手がかりも得られます。オープンソースがほとんどすべてのソフトウェアで使用されている現在、企業がソフトウェア開発においてオープンソースを適切に管理できていないということは、それ以外の部分も適切に管理できていない可能性が考えられます。

買い手側は、取引条件を設定する前に、買収対象のコードに問題のあるオープンソースが含まれているかどうかを特定する必要があります。ここでも、広く深い知見を得るには、信頼のおける第三者による監査を実施するのが最適です。たとえ寛容型ライセンスのオープンソースであっても、これを特定しないと買い手側はライセンスの著作権表示の義務に従うことができないため、きちんと特定しておく必要があります。売り手側は、自社コードの構成要素や、オープンソースのセキュリティおよびライセンスに関するリスクをどの程度適切に管理しているかについての質問に回答できるようにしておく必要があります。企業のコードには把握できていないオープンソースが大量に含まれているという現状を考えると、デュー・ディリジェンスで不測の事態が発生するのを避けるために、先手を打って事前に監査を受けておくのがよいでしょう。

M&A 取引においてオープンソース監査を実施して、オープンソース・コードおよびサードパーティのコンポーネントとライセンスを特定することにより、法的およびセキュリティ上の潜在的な問題を事前に見つけることができます。




結論を言えば、買収によって取得したコードに含まれるオープンソース・コンポーネントには、金銭面でもブランド価値の面でも重大なリスクが潜んでいる可能性があります。このようなリスクを買い手側のデュー・ディリジェンスの一環として評価することは、M&A 取引の意思決定プロセスに欠かすことができません。

# リスクに影響するメンテナンスおよび運用面の要因

オープンソース・コンポーネントは、強力なコミュニティによって維持されているものだけを使用するのが理想です。結局のところ、大規模で活発な開発者コミュニティによるサポートの存在こそが、オープンソースの推進者たちがソフトウェアを最初に投入したときに提唱した OSS の重要な利点の 1 つでした。つまり、熱心な開発者コミュニティがコード品質とセキュリティの向上に取り組み、管理対象のプロジェクトを定期的に改良していくというシナリオです。

しかし残念なことに、このような状況が現実のものとなっているオープンソース・プロジェクトは多くありません。Black Duck を含む各社の SCA データを分析した Linux Foundation の「Census III of Free and Open Source Software」レポートで報告されているように、現在広く使用されているオープンソースの多くはごく少数のコントリビューターが開発とメンテナンスを行っており、一般に考えられているように数千人または百万人規模の開発者によって支えられているわけではありません。機能の改良、セキュリティや安定性の向上など、アップデートの提供に取り組んでいるコントリビューターが元々少ない上に、時間が経つにつれてその数がますます減少しているのがほとんどすべての OSS プロジェクトの現状です。

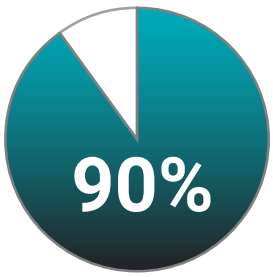
メンテナーがプロジェクトのメンテナンス活動をやめてしまうと、結果的にセキュリティ・リスクが高まることは今回の監査データからも見てとれます。

## 旧バージョンのコンポーネント

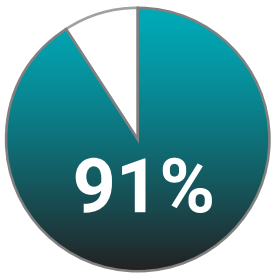
- 旧バージョンになってから 4 年超が経過したオープンソース・コンポーネントを使用しているコードベースの割合は 90% にものぼっていました。これは、古い依存関係の問題が蔓延しており、セキュリティ上の脆弱性や互換性の問題を引き起こす可能性があることを示しています。

## 開発活動実績のないコンポーネント

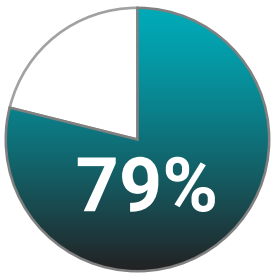
- 過去 2 年間に開発活動実績のなかったコンポーネントを使用しているコードベースも、91% という高い割合で存在していました。これは、アップデートの提供が止まっている OSS に依存しているアプリケーションおよび Web サービスが多数存在しており、まだ発見されていない、またはパッチ未適用のセキュリティ上の欠陥に対して脆弱なままになっている可能性があることを示唆しています。
- 過去 24 カ月間に活動実績のなかったコンポーネントの最新バージョンを使用しているコードベースの割合は、全体の 79% でした。これは、たとえ最新のコンポーネントを使用していても、それが活発にメンテナンスされているとは限らないことを示唆しています。
- 過去 24 カ月間に活動実績のなかったコンポーネントの最新でないバージョンを使用しているコードベースの割合は、全体の 88% でした。このようなコンポーネントを使用していると、リスクはさらに大きくなります。



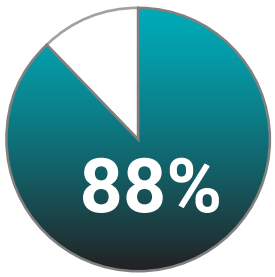

旧バージョンになってから
4 年超が経過した
オープンソース・
コンポーネントを使用している
コードベースの割合




過去 2 年間に
新たな開発活動実績のなかった
コンポーネントを使用している
コードベースの割合




過去 24 カ月間に
活動実績のなかった
コンポーネントの
最新バージョンを使用している
コードベースの割合




過去 24 カ月間に
活動実績のなかった
コンポーネントの最新でない
バージョンを使用している
コードベースの割合

## アップデートの遅れ

- 大多数のコードベース（91%）が、最新バージョンでない OSS コンポーネントを使用しています。
- さらに悪いことに、最新バージョンよりも 10 バージョン以上前のオープンソース・コンポーネントを使用しているコードベースの割合も全体の 90% に達しています。

最新バージョンには重要なバグ修正やセキュリティ・パッチが含まれていることが多いため、アップデートを怠ると、リスクと技術的負債が増えることになります。

オープンソース・コンポーネントを最新バージョンにアップデートしていないのには、それなりの理由が存在することもあります。例えば大型アップデートの場合、変更点が多すぎて既存のコードが動作しなくなる可能性があります。特に、現在使用しているバージョンが既に最新バージョンより数バージョン前の場合はなおさらです。また、コードを適応させるための工数が現実的でないこともあります。アップデートには、開発、テスト、デプロイに長時間かかることがあります。リソースの限られた小規模なチームやプロジェクトの場合、より重要なタスクを優先せざるを得ないこともあります。

これまで 10 年間にわたって OSSRA レポートで繰り返し指摘してきたように、オープンソースは商用ソフトウェアとは異なります。それはどちらが良い、悪いというものではなく、単に異なる存在であるという意味です。したがって、そのメンテナンスについても異なるアプローチが要求されます。例えば、商用ソフトウェアを使用している組織であれば、パッチやアップデートはソフトウェアにプッシュ配信されるか、少なくともアップデートがダウンロード可能になるとベンダーから通知が届くというスタイルに慣れています。しかしオープンソースではそのようなことはめったになく、ユーザー自身でコンポーネントのステータスを常に確認しなければならないことがほとんどです。

こうした現状の中で、最新アップデートを見逃さないようにするにはどうすればよいでしょうか。

**プロジェクトの Web サイトやリポジトリをフォローする。**ほとんどのオープンソース・プロジェクトは、Web サイトやブログ、リポジトリ（GitHub など）で最新リリースの発表や変更履歴の提供を行っています。メーリング・リストや RSS フィードに登録して最新情報を得ることもできます。しかし、現在の一般的なアプリケーションに含まれるオープンソース・コンポーネントの数を考えると、手動での追跡は現実的でなく、自動化したアプローチの重要性が高まっています。

**パッケージ管理ツールを使用する。**npm、pip、Bundler など多くのパッケージ管理ツールには、アップデートが利用可能になると通知を表示して、更新プロセスを自動化する機能があります。今回のスキャンで検出されたオープンソースのほとんどのダウンロード元である npm には、パッケージをアップグレードするためのさまざまなツールが用意されています。例えば、「npm outdated」を実行すると、アップデートが利用可能なパッケージの一覧が表示されます。

**バージョン追跡ツールを使用する。**Dependabot や Renovate などのツールには、プロジェクトの依存関係を監視し、アップデートのプルリクエストを自動的に作成する機能があります。

**特定および監視ツールを使用する。**Black Duck SCA などのセキュリティ・ツールには、コードベースをスキャンしてオープンソース・コンポーネントの脆弱性を見つけ、セキュリティ・パッチを含むアップデートが利用可能になると通知する機能があります。

使用しているオープンソースを常に把握するための有効なソリューションは 1 つしかありません。それは、オープンソースの正確かつ網羅的な目録を作成し、ソフトウェアに含まれるオープンソースの脆弱性、アップグレード、および全体的な健全性を監視するプロセスを自動化することです。

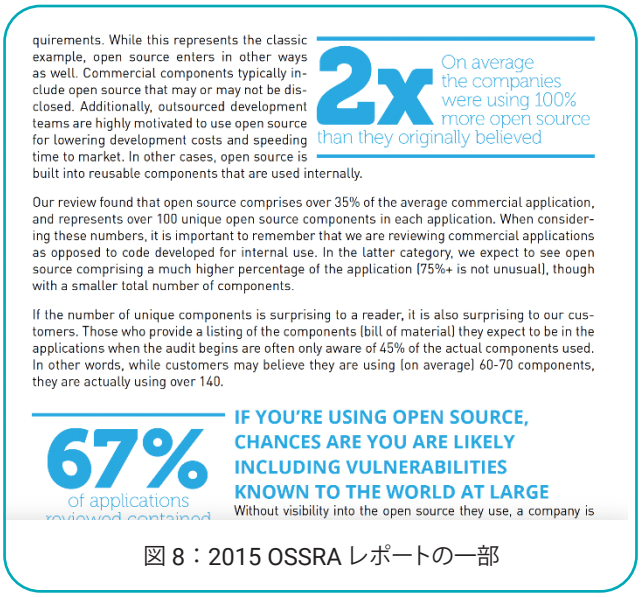
# まとめ：時は流れても

> 「リスクが生じるのは、自分が何をしているかを理解していないからだ」
>
> – ウォーレン・バフェット（バークシャー・ハサウェイ会長兼 CEO）、チャーリー・マンガー（同副会長）

これまで 10 年間にわたり、ブラック・ダックの「オープンソース・セキュリティ & リスク分析」レポートは一貫して「自社のコードの中身を把握していますか」という問いかけをテーマとしてきました。2015 年から時は流れ、表面的な数字は変化しています（ほとんどの場合、大きく増加しています）が、その問いかけは今も変わりません。

ソフトウェア・サプライチェーンのどちらの側にいるのか、すなわちパイプラインの上流でソフトウェアを開発しているのか、下流で複数のベンダーからのソフトウェアを利用しているのかにかかわらず、また、そのソフトウェアがオンプレミスかクラウドか、組み込みかモバイルかにかかわらず、そのソフトウェアにはまず間違いなくオープンソース・コードが含まれています。では、具体的にどのような OSS コンポーネントが含まれているか、そしてそれらがセキュリティ、コード品質、ライセンスに関するリスク要因となるかどうかを正確に把握できているでしょうか。

97% のコードにオープンソースが含まれている現状では、コードの中身を可視化することを優先課題とする必要があります。また、90% のコードベースが最新バージョンよりもはるかに古いバージョンのオープンソースを使用している現状では、コードを最新の状態に維持することを全員が心がける必要があります。特にそれが人気のあるオープンソース・コンポーネントであればなおさらです。


quirements. While this represents the classic example, open source enters in other ways as well. Commercial components typically include open source that may or may not be disclosed. Additionally, outsourced development teams are highly motivated to use open source for lowering development costs and speeding time to market. In other cases, open source is built into reusable components that are used internally.

2x On average the companies were using 100% more open source than they originally believed

Our review found that open source comprises over 35% of the average commercial application, and represents over 100 unique open source components in each application. When considering these numbers, it is important to remember that we are reviewing commercial applications as opposed to code developed for internal use. In the latter category, we expect to see open source comprising a much higher percentage of the application (75%+ is not unusual), though with a smaller total number of components.

If the number of unique components is surprising to a reader, it is also surprising to our customers. Those who provide a listing of the components (bill of material) they expect to be in the applications when the audit begins are often only aware of 45% of the actual components used. In other words, while customers may believe they are using (on average) 60-70 components, they are actually using over 140.

67% of applications reviewed contained

IF YOU'RE USING OPEN SOURCE, CHANCES ARE YOU ARE LIKELY INCLUDING VULNERABILITIES KNOWN TO THE WORLD AT LARGE

Without visibility into the open source they use, a company is

図 8：2015 OSSRA レポートの一部

- **現代のソフトウェアには、驚くほど多くの OSS が使われています。**監査した商用コードベースの 97% にオープンソースが含まれており、業種によってはすべてのコードベースにオープンソースが含まれていました。自社がそのような業種に該当しているかを確認してみてください。
- **オープンソースは手動では管理できません。**アプリケーション 1 つあたりに含まれるオープンソース・ファイルの数の平均は過去 4 年間で 3 倍に増加しています。コードを複雑化させている大きな要因の 1 つは、推移的依存関係です。スキャンで検出されたオープンソース・コンポーネントの 64% が推移的依存関係でした。
- **セキュリティ上の脆弱性に関するリスクが蔓延しています。**セキュリティ診断を受けたコードベースの大半（81%）に高リスクまたは重大なリスクのある脆弱性が見つかっています。これらの脆弱性の多くは、旧バージョンのオープンソース・コンポーネントを使用していることが原因で発生しています。
- **これらの脆弱性の多くは、特定のコーディング上の弱点が原因で発生しています。**今回見つかったオープンソース脆弱性のうち、不適切な入力確認に関連するものが 71% もありました。
- **コードの中身を可視化するには、ソフトウェア部品表（SBOM）が不可欠です。**SBOM は、リスク、脆弱性、ライセンス・コンプライアンス、ソフトウェア品質、および M&A デュー・ディリジェンスの管理に重要な役割を果たします。
- **ライセンスのリスクもよく見られる問題です。**監査したコードベースの半分以上（56%）にライセンスの競合が見つかっており、その多くは両立性のない推移的依存関係によって引き起こされています。また、ライセンスのない、またはカスタム・ライセンスの OSS コンポーネントを使用しているコードベースが全体の 33% ありました。
- **旧バージョンのコンポーネントの使用は大きな問題を引き起こします。**監査したコードベースのほとんど（91%）に旧バージョンのコンポーネントが含まれていました。また、最新バージョンより 11 バージョン以上前のコンポーネントを含んでいるコードベースも全体の 90% に達しています。

# 主な提言

**SCA を導入する。**SCA ツールを使用して SBOM を生成し、脆弱性を特定し、ライセンス・コンプライアンスを管理します。

**リスク管理に優先順位を付ける。**ビジネスの核心部分に影響する高リスクの脆弱性およびライセンスの問題に集中的に対処します。

**定期的に OSS をアップデートする。**セキュリティ・アドバイザリを利用して最新情報を収集し、特に jQuery のように人気のあるライブラリなど、ソフトウェアに脆弱性が見つかったら迅速にパッチを適用します。

**セキュア・コーディング・プラクティスを確立する。**入力バリデーション、サニタイズ、およびサードパーティ・コードの定期的なセキュリティ・テストには特に重点を置くようにします。

**OSS のメンテナンス状況をチェックする。**プロジェクトの Web サイトをフォローする、パッケージ管理ツールを使用する、自動化されたセキュリティ・サービスを利用するなどして、オープンソース・コンポーネントのアップデート提供状況を常に確認します。

**SBOM を作成する。**自社のコードに含まれるすべてのオープンソース・コンポーネントについて、ライセンス、バージョン、出所の情報を含めた詳細な SBOM を作成します。

**OSS 管理を SDLC に組み込む。**オープンソース管理をセキュア・ソフトウェア開発フレームワークに組み込み、CISA や NIST が示しているベスト・プラクティスに従うようにします。

**M&A を計画している場合は、Black Duck 監査を使用して買収対象のソフトウェアを精査する。**M&A のようなリスクの高い状況では、高度なツールとノウハウを持った信頼できる第三者に買収対象のソースコードを監査してもらう必要があります。

このレポートの冒頭でも述べたように、オープンソース・ソフトウェアには多くの利点がある一方、そこには大きなリスクもあり、こうしたリスクを積極的に管理する必要があります。潜在的な問題を回避するには、ソフトウェア・サプライチェーンの包括的な可視化、堅牢なセキュリティ・プラクティス、そしてライセンスとメンテナンスに対する予防的なアプローチが必要です。SCA ツール、SBOM、および適切な衛生管理を導入することは、現在のソフトウェア環境においては選択肢ではなく必須の課題です。ここに示した提言を実践することで、リスクを軽減し、オープンソース・ソフトウェアの利点を安全かつ効果的に活かし続けることが可能になります。

**必要なのは、コードの中身を把握し、そこに不確実性を残さないことです。**

**謝辞**

2025「オープンソース・セキュリティ & リスク分析」レポートは、ブラック・ダックの監査、CyRC、プロフェッショナル・サービス、およびマーケティングの各チームによる貢献により、ブラック・ダックによって作成されました。

Nancy Bernstein、Conor Brolly、Kevin Collins、Scott Handy、Clement Pang、Merin McDonell、Mike McGuire、Phil Odence、Liz Samet、Mark Van Elderen、および Jack Taylor の各氏に感謝します。

執筆：Fred Bals

## ブラック・ダックについて

ブラック・ダックは、業界で最も包括的かつ強力で信頼できるアプリケーション・セキュリティ・ソリューション・ポートフォリオを提供します。ブラック・ダックには、世界中の組織がソフトウェアを迅速に保護し、開発環境にセキュリティを効率的に統合し、新しいテクノロジーで安全に革新できるよう支援してきた比類なき実績があります。ソフトウェア・セキュリティのリーダー、専門家、イノベーターとして認められているブラック・ダックは、ソフトウェアの信頼を築くために必要な要素をすべて備えています。
詳しくは www.blackduck.com/jp をご覧ください。

**ブラック・ダック・ソフトウェア合同会社**
www.blackduck.com/jp


BLACKDUCK®