BLACKDUCK®

2024
オープンソース・セキュリティ &
リスク分析レポート

オープンソース・サプライチェーンの
セキュリティ対策ガイド

目次

エグゼクティブ・サマリー

このレポートでは、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の観点から、オープンソース・ソフトウェアの製作者と利用者の双方がオープンソースの責任ある管理を実践できるように、いくつかの提言を示していきます。ソフトウェアを提供する側も利用する側もソフトウェア・サプライチェーンの一員であり、使用するアプリケーションを上流および下流からのリスクから保護する必要があります。このレポートでは、以下の内容について考察します。

- なかなか消えないオープンソースのセキュリティ上の懸念
- 開発者がもっと積極的にオープンソース・コンポーネントを最新の状態に維持すべき理由
- ソフトウェア・サプライチェーン管理におけるソフトウェア部品表（SBOM）の必要性
- AI コーディング支援ツールによってもたらされるセキュリティおよび IP コンプライアンスのリスクから身を守る方法

これまで 10 年近くにわたり、オープンソース・セキュリティ & リスク分析（OSSRA）レポートは「コードの中身を把握していますか」という問いかけを主要なテーマとしてきました。オープンソースの利用拡大と AI 生成コードの増加により、ますます多くのアプリケーションがサードパーティ・コードを使用して構築されている今、この問いかけの重要性は今まで以上に高まっています。

コードの中身を完全に可視化できなければ、そのソフトウェアにどのようなリスクが含まれているのかを自社はもちろん、取引先のベンダーやエンドユーザーも自信を持って答えることができません。ソフトウェア・サプライチェーンのセキュリティ対策は、コードにどのようなオープンソース・コンポーネントが含まれているかを把握し、それぞれのライセンス、コード品質、潜在的な脆弱性を特定することから始まります。

2024 OSSRA について

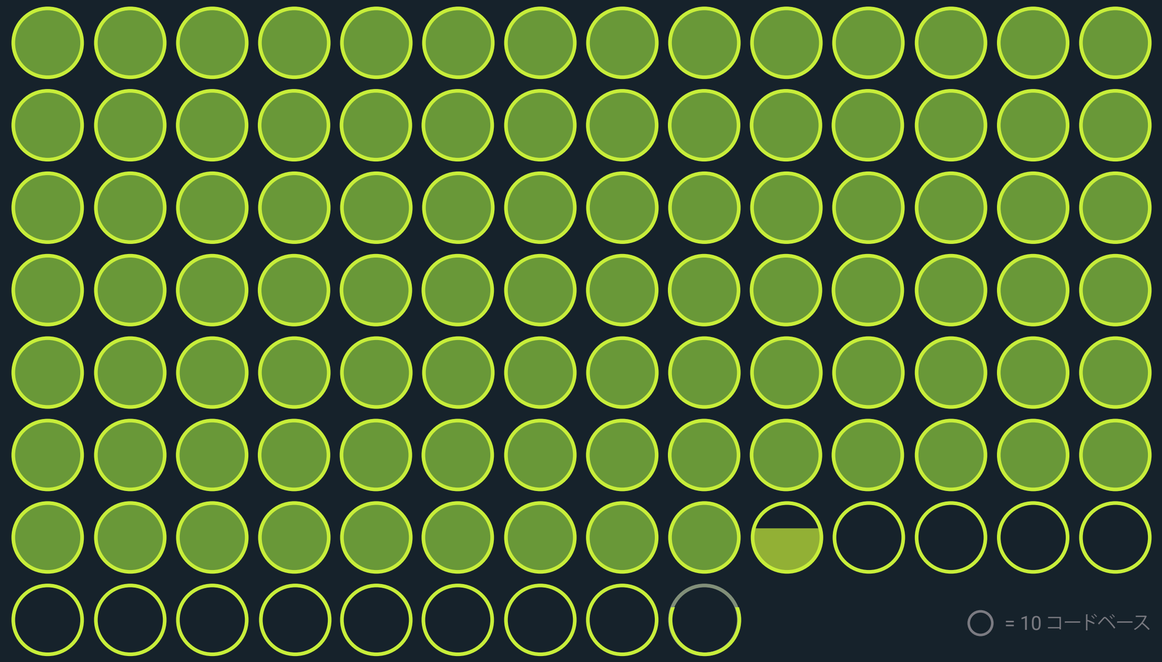
2024 年で 9 回目の発行を迎えた「オープンソース・セキュリティ & リスク分析（OSSRA）レポート」は、商用ソフトウェアに含まれるオープンソースのリスクについての現状をセキュリティ、コンプライアンス、ライセンス、およびコード品質の面から詳しく分析しており、セキュリティ、法務、リスク、開発チームにオープンソースのセキュリティおよびライセンスのリスク状況をよりよく理解していただくことを目的として、その分析結果を発表しています。

今回のレポートは、2023 年に 17 業種で 1,067 の商用コードベースから得られた匿名化された調査結果をブラック・ダック Black Duck® 監査サービス・チームが分析したデータを使用しています。このチームは、これまで 20 年以上にわたって世界中のセキュリティ、開発、法務チームに対し、セキュリティおよびライセンス・コンプライアンス・プログラムの強化を支援してきました。また、主に顧客の合併・買収（M&A）取引においてソフトウェアのリスクを特定することを目的として、毎年数千ものコードベースを監査しています。

この監査では、組織で使用しているアプリケーションに含まれるオープンソース、サードパーティ・コード、web サービス、および API を網羅した、最新かつ正確なソフトウェア部品表（SBOM）も提供しています。この監査サービス・チームは、Black Duck KnowledgeBase ™のデータに基づいてライセンス・コンプライアンスおよびセキュリティの潜在的リスクを特定しています。Black Duck KnowledgeBase には、31,000 を超えるフォージおよびリポジトリからブラック・ダック 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 780 万以上のオープンソース・コンポーネントのデータが登録されています。

OSSRA レポートが指摘しているように、オープンソースはほとんどのソフトウェアで使用されており、オープンソースの不適切な管理はさまざまなリスクをもたらす可能性があります。オープンソースは、今や企業や個人が使用しているすべてのアプリケーションの基盤となっています。オープンソースを効果的に特定、追跡して管理することは、ソフトウェア・セキュリティ・プログラムの成功に欠かせないだけでなく、ソフトウェア・サプライチェーンのセキュリティを強化するための重要な要素でもあります。

概要



1,067 = 2023 年に**スキャン**したコードベース

936 = **リスク診断**も実施したコードベース

84%

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

74%

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

96%

全コードベースのうち、
オープンソースを
含むものの割合

77%

全コードベースに**占める**
オープンソース・コードの割合

53%

全コードベースのうち、
ライセンスの競合が
見つかったものの割合

31%

全コードベースのうち、
ライセンスのない、
または**カスタム・ライセンス**を
使用しているものの割合

10 年

リスク診断を受けた
コードベースの **14%** が
公開から **10 年を超える**
脆弱性を含んでいた

2.8 年

リスク診断を受けた
コードベースに含まれていた
脆弱性の公開からの年数は
平均 **2.8** 年

24 カ月

リスク診断を受けた
コードベースの **49%** が、
過去 24 カ月にわたり
開発活動実績のなかった
コンポーネントを使用

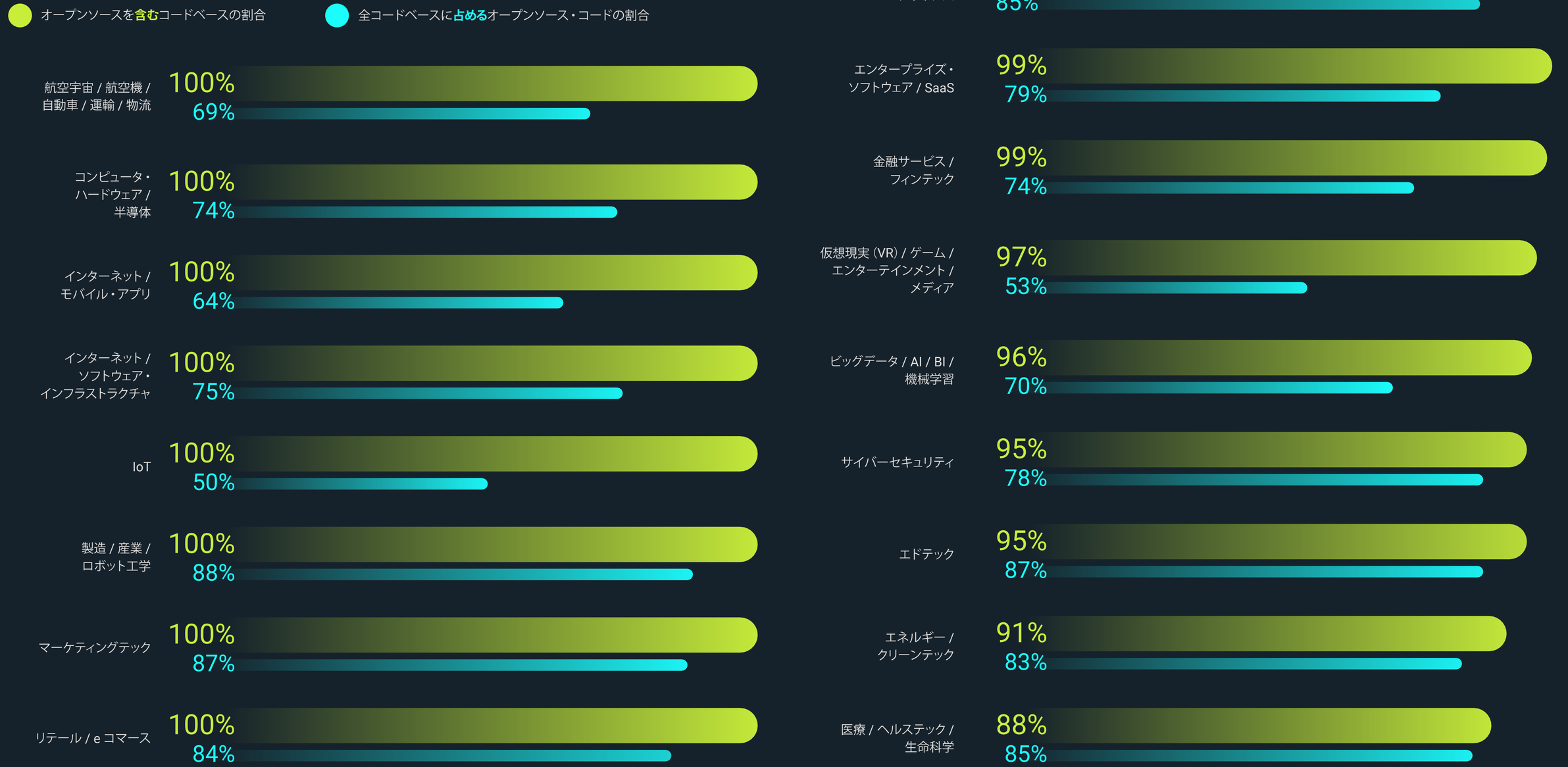
12 カ月

リスク診断を受けた
コードベースの **1%** が、
12 カ月以上メンテナーによる
アップデート / パッチ適用のない
コンポーネントを使用

91%

最新バージョンよりも 10 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 1：スキャンした全 1,067 コードベースの業種別の状況



オープンソースの脆弱性とセキュリティ

監査について

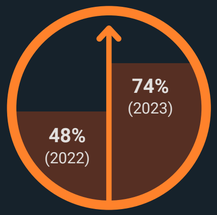
Black Duck 監査では、すべてのコードベースに対してオープンソースのライセンス・コンプライアンス検査を実施していますが、脆弱性および運用リスクの診断は任意（オプトアウト方式）で実施しています。2023 年に Black Duck 監査サービス・チームは 1,067 件の監査を実施しましたが、このうち 88%（936 件）が脆弱性 / 運用リスクの診断も受けています。2024 年版 OSSRA レポートの「オープンソースの脆弱性とセキュリティ」および「オープンソースのリスクに影響する運用面の要因」のセクションに示したデータは、リスク診断も受けた 936 のコードベースを母集団としていますが、「オープンソース・ライセンス」のセクションに示したデータは全 1,067 のコードベースを母集団としています。

今年の OSSRA レポートでは、Black Duck 監査サービス・チームが分析した 1,067 のコードベースを基本データとしており、このうち 96% がオープンソースを含んでいました。また、スキャンした全ソースコードおよびファイルの 77% がオープンソース・コードを起源としていました。

1 つの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は、平均で 526 個でした。このことは、セキュリティ・テストの自動化が非常に重要、場合によっては不可欠とも言えることを物語っています。コンポーネント数が少なければ手動テストでも対処可能かもしれませんが、これほどの規模になるとそれは事実上不可能であり、ソフトウェア・コンポジション解析（SCA）のような自動ソリューションが必要です。手動テストとは異なり、自動セキュリティ・テストは迅速かつ継続的に実行できるため、開発者は開発プロセスの早い段階で問題を特定でき、デリバリのスケジュールや生産性への影響を抑えることができます。

リスク診断を受けたコードベースの 84% に既知のオープンソース脆弱性が少なくとも 1 つ含まれていました。また、高リスク脆弱性を含んでいたのは 74% で、これは 2022 年の 48% から大きく上昇しています。高リスク脆弱性とは、実際に攻撃を受けたことがあるか、概念実証コード（エクスプロイト）が存在する、またはリモート・コード実行の脆弱性に分類されるものを言います。

84% のコードベースが少なくとも **1 つのオープンソースの脆弱性**を含んでいる



高リスク脆弱性を含むコードベースの割合はこの 1 年で **54% 増加**

高リスク脆弱性がこの 1 年で 54%（26 ポイント）も増加したことを 1 つの要因で説明することはできませんが、景気の低迷とそれに伴うレイオフの結果、脆弱性の特定と修正に当たる人員が削減されたことも要因の 1 つと考えられます。また、最新バージョンよりも 10 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が 91% もありました。このことからも明らかなように、オープンソース利用者のほとんどが、使用しているコンポーネントをアップデートできていません。

過去 24 カ月にわたり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が 49% あり、12 カ月以上メンテナーによるアップデート / パッチ適用のないコンポーネントを使用しているコードベースも 1% ありました。

大まかに言って、「メンテナー」とはオープンソース・プロジェクトを主導するコントリビューターを指す用語です。メンテナーは、ソースコードのどの部分をビルド / リリースに進めるのかを最終的に判断することもあれば、小規模なプロジェクトではすべてのコード・レビューを実施してメンテナー自身の名義でコードをホスティングすることもあります。また、プロジェクトの方向性についてメンテナーが最終的な意思決定を下す場合もあります。日々の作業内容はさまざまですが、プル・リクエストやその他の貢献のレビュー、ソフトウェアの新バージョンのリリース、セキュリティ上の問題修正のトリアージと処理、コミュニティの管理とモデレーションなどもメンテナーの活動に含まれます。

ほとんどのメンテナーは、自身が関与しているオープンソース・プロジェクトを最新の状態に維持する活動に熱心に取り組んでいます。事実、多くの企業で自社ソフトウェアが依存しているオープンソース・プロジェクトをメンテナンスするための人員を雇用しています。オープンソースの利用者側でも、これと同様の熱心な取り組みを奨励する必要があります。例えば、使用しているバージョンを常に把握する、定期的なアップデート・サイクルを確立する、メンテナーとコントリビューターの健全なエコシステムが形成されたプロジェクト以外からはダウンロードしないなどのソフトウェア・ハイジーン（衛生管理）を励行する、といったことが求められます。

ソフトウェアに存在するセキュリティ上の弱点や脆弱性を特定・分類するのによく使用されるリストとして、CWE（共通脆弱性タイプ一覧）と CVE（共通脆弱性識別子）があります。ブラック・ダックは、顧客に対して Black Duck Security Advisor（BDSA）と呼ばれる独自のレポートを発行しており、NVD（脆弱性情報データベース）の CVE 通知よりも高度で詳細な情報を迅速に提供しています。BDSA は顧客の SBOM に含まれるコンポーネントに影響する脆弱性についての詳細情報と実践的なアドバイスを提供し、オープンソース脆弱性がもたらす可能性のあるリスクを完全に可視化できるように支援します。

上位 10 の脆弱性のうち 8 つは、Pillar CWE がいずれも **CWE-707** です。CWE-707 は、セキュリティ要件が満たされておらず、クロスサイト・スクリプティングや SQL インジェクションなどの悪用につながる脆弱性です。

図 2 に示したように、CWE-20、79、80、97、937 はいずれも CWE-707 が Pillar CWE となっています。CWE-707 は、データを上流コンポーネントから読み出す、または下流コンポーネントへ送信する前にセキュリティ要件が満たされていないことに関する脆弱性です。入力を適切に無効化していないと、クロスサイト・スクリプティング（XSS）や SQL インジェクションなどの悪用につながることがあります。XSS はよくある危険なセキュリティ攻撃で、このレポートの上位 10 の脆弱性の大半に関連しています。

XSS は、主に JavaScript で記述された悪意のある不正な形式のコードを攻撃者が送信し、web サイトの欠陥を悪用することで発生します。この入力を適切に無効化またはエスケープしていないと、攻撃者は本来信用できるはずのホストを操作して悪意のあるタスクを実行させることができます。しかしほとんどの XSS 攻撃は、ホスト自体ではなく web アプリケーションの他のユーザーを最終的な標的としています。いったん悪意のあるスクリプトが挿入されると、それを使用してセッション・クッキーのような機密情報を盗むことができます。

XSS は OSSRA の上位 10 の脆弱性リストに入っているだけでなく、OWASP Top 10（最も重大な web アプリケーション・セキュリティ・リスクのトップ 10 リスト）にも含まれています（XSS は「A03:2021 - インジェクション」に分類）。XSS の脆弱性が蔓延しているのは、組織と顧客、およびユーザー同士のやりとりの接点として web ベースのアプリケーションへの依存度が高まっていることに理由があります。これは、e コマース企業、銀行、インターネット・サービス・プロバイダー、保険会社など、どれだけ多くの企業が顧客やパートナーとのエンゲージメントのために web での体験を提供しているかを考えれば分かることです。

このデータからも、開発チームがオープンソース・コンポーネントを常に最新の状態に維持する必要があるのは明らかで、特に jQuery のようにユーザー数の多いオープンソース・コンポーネントの場合はなおさらです。旧バージョンの脆弱なオープンソースを使用していると、深刻な結果を招くことがあります。例えば、今回の監査で見つかった脆弱性で 2 番目に多かったのは BDSA-2020-0686（CVE-2020-11022）で、jQuery のバージョン 1.2 から 3.5.0 より前までに存在する XSS の脆弱性でした。この脆弱性があると、たとえサニタイズしてあっても、信頼できないソースから HTML を jQuery の DOM 操作メソッドの 1 つに渡した際に、信頼できないコードを実行してしまう場合があります。

この問題は jQuery 3.5.0 で修正されましたが、今回の OSSRA では、セキュリティ診断を受けたコードベースの 1/3 が今もこの脆弱性のあるバージョンの jQuery を使用していることが明らかになっています。これらのバージョンでは、悪意のあるデータを使用してシステムに侵入することが可能で、パスワードや信用情報などの機微なデータが流出する危険があります。先に述べた通り、XSS はアプリケーションにおいて最も一般的な脆弱性の 1 つであり、一般的には HTML や JavaScript などブラウザのさまざまな言語インタープリターに対するコード・インジェクション攻撃が使用されます。

リスクを軽減する：jQuery などの有名なオープンソースを安全に使用するためのヒント

今回の監査で最も多く見つかった上位 10 のオープンソース・コンポーネントは、いずれも JavaScript で作成されていました。また、今回の監査で見つかった脆弱性の大半は JavaScript ライブラリに関するもので、特に目立ったのが旧バージョンの jQuery に存在する脆弱性でした。

- 最新バージョンの jQuery を使用する。このレポートで指摘したように、jQuery は旧バージョンの多くに脆弱性が存在します。
- 最新の脆弱性情報を入手するために Black Duck Security Advisory などのセキュリティ・アドバイザリー・サービスの契約を検討する。jQuery には今も新しい脆弱性がたびたび見つかっています。
- コードに存在する潜在的な脆弱性を特定・回避するためにセキュア・コーディング・フレームワークを使用する。
- ソフトウェア開発ライフサイクル全体でコード品質とセキュリティ上の欠陥に対処するために静的解析、ソフトウェア・コンポジション解析、動的解析ツールなどの自動化されたセキュリティ・テストを使用する。

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、人気の高さゆえの問題が存在します。このレポートに示した jQuery の脆弱性にはいずれも既にパッチが存在するにもかかわらず、今回の監査で最も多くの脆弱性が見つかったのが jQuery でした。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。

ソフトウェア・サプライチェーンへの脆弱性混入を防ぐための対策

- ソフトウェア部品表（SBOM）を作成して維持管理する。ソフトウェア・サプライチェーン攻撃からの防御において、すべてのオープンソース・コンポーネントを網羅した正確かつ最新の SBOM を用意しておくことは、露出を評価し、コードが高い品質とコンプライアンス、セキュリティを維持できていることを確認する上で非常に重要です。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、および各コンポーネントのライセンス、バージョン、パッチ適用状況を網羅的にリストアップした SBOM は、悪意あるパッケージを使用した攻撃など、サプライチェーン攻撃に対する強力な防御となります。
- 最新の情報を収集する。新たに特定された悪意あるパッケージ、マルウェア、公開されたオープンソース脆弱性についての情報入手手段を確立しておく必要があります。ニュースフィードや定期的に発行されるアドバイザリーなど、自社の SBOM に登録されているオープンソース・コンポーネントに影響する問題について実践的なアドバイスや詳細情報を得ることのできる手段を見つけるようにします。
- コード・レビューを実施する。ダウンロードしたソフトウェアのコードをプロジェクトに組み込む前に検査します。ここでは、既知の脆弱性が存在しないかチェックします。さらに深い洞察を得るには、ソースコードの静的解析を実行し、未知のセキュリティ上の弱点を見つけることも検討する必要があります。
- 先手を打つ。現時点で脆弱でないからと言って、そのコンポーネントがいつまでも脆弱でないとは限りません。また、意図的に作られた悪意のあるパッケージはそもそも「脆弱性」として検出されないこともあります。セキュリティ上の問題が将来的に発生するのを防ぐには、実装する前にコンポーネントの健全性と出所に注意を払う必要があります。
- 自動化されたソフトウェア・コンポジション解析（SCA）ツールを使用する。SCA ツールは、ソフトウェアのセキュリティ上の問題を自動で特定、管理、軽減してくれるため、開発者はコーディングに専念できます。このようなツールは、オープンソースおよびサードパーティ・コードの評価が可能です。

上位 10 の脆弱性のうち 8 つは Pillar CWE が同じ

CWE プロジェクトでは、最も抽象度が高く、関連するすべての Class/Variant の元となる弱点を「Pillar Weakness」と定義しています。

図 2：上位 10 の CVE/BDSA

BDSA-2021-3651
関連する CWE：CWE-546
影響を受けるコンポーネント：jQuery

BDSA-2020-0686 (CVE-2020-11022)
関連する CWE：CWE-707
影響を受けるコンポーネント：jQuery

BDSA-2020-0964 (CVE-2020-11023)
関連する CWE：CWE-80
影響を受けるコンポーネント：jQuery
Pillar CWE：CWE-707

BDSA-2019-1138 (CVE-2019-11358)
関連する CWE：CWE-20
影響を受けるコンポーネント：jQuery
Pillar CWE：CWE-707

BDSA-2017-2930 (CVE-2015-9251)
関連する CWE：CWE-79
影響を受けるコンポーネント：jQuery
Pillar CWE：CWE-707

BDSA-2014-0063
関連する CWE：CWE-79
影響を受けるコンポーネント：jQuery
Pillar CWE：CWE-707

BDSA-2015-0567
関連する CWE：CWE-937
影響を受けるコンポーネント：jQuery
Pillar CWE：CWE-707

BDSA-2019-0423 (CVE-2019-8331)
関連する CWE：CWE-97
影響を受けるコンポーネント：Bootstrap
Pillar CWE：CWE-707

BDSA-2020-1173 (CVE-2020-7656)
関連する CWE：CWE-80
影響を受けるコンポーネント：jQuery
Pillar CWE：CWE-707

BDSA-2022-1987 (CVE-2022-31160)
関連する CWE：CWE-79
影響を受けるコンポーネント：jQuery UI
Pillar CWE：CWE-707

一部の BDSA に CVE がない理由

公表されたオープンソース・ソフトウェアの脆弱性に関する情報を収集する方法としてまず考えられるのが、NVD（脆弱性情報データベース）などの公的な情報源です。しかし NVD で CVE のエントリが報告されるまでにはタイムラグがあり、NVD で公開される脆弱性情報には一貫して即時性が問題となってきました。事実、多くの脆弱性は最初に発表されてから NVD で公開されるまでに相当なタイムラグがあり、その平均期間を 1 カ月としている調査結果もあります。

NVD でもう 1 つ問題となるのが、脆弱性情報がしばしば不完全であるという点です。NVD で公開される CVE の多くは脆弱なバージョンの範囲を明記しておらず、情報が少なすぎてあまり役に立たないことがあります。これは主に、エントリを調査するのに十分なリソースが存在しないことに理由があります。

NVD だけに頼って脆弱性情報を収集するのが賢いやり方でないのは明らかです。多くの商用 SCA ソリューションは、NVD よりも充実した脆弱性データを提供しており、問題をより正確に特定し、必要であればより迅速に修正できるように支援もしてくれます。例えば、ブラック・ダックの SCA ソリューションである Black Duck のユーザーには、ブラック・ダックのセキュリティ・リサーチ・チームが特定したオープンソース脆弱性のアドバイザリーが BDSA として提供されます。多くの場合、ユーザーのコードベースに影響する脆弱性情報が通知されるのは BDSA の方が早く、NVD で公開されるより数日～数週間早いこともあります。また、提供される脆弱性データも BDSA の方がより網羅的で、セキュリティ分析情報、技術内容の詳細、アップグレード / パッチ・ガイダンスなども他の商用ソリューションより充実しています。例えば、2023 年に見つかった上位 10 の脆弱性のうち、NVD に関連する CVE が存在しない BDSA が 3 つ存在します。

BDSA-2021-3651

BDSA によると、一部のバージョンの jQuery には、コメント内にハイジャックされたドメインへの参照が含まれています。これも一種のセキュリティ上の問題であり、jQuery バージョン 3.6.1 で対策がなされたように、ハイジャックされたドメインへのリンクを削除するのが最も安全です。そうしないと、ドメインのステータスに気付いていないユーザーが、ハイジャックされたサイトへのリンクをクリックして不特定の攻撃を受ける可能性があります。コードを実行してもこれらのサイトへ誘導されることはありませんが、開発者などコードにアクセスできる人が誤って悪意のあるサイトへのリンクをクリックしてしまうこともあるため、この問題への注意を喚起することには意味があります。

ブラック・ダックの CyRC チームは、この BDSA に「情報（informational）」タグを付けるよう助言しています。このタグは、ベンダーから提供される修正策がコードまたは製品ドキュメント内での警告の形を取る場合、またはベンダーが CVE を拒否または調査結果に異議を唱え、報告された脆弱性をコンポーネントの期待される設計上の動作であると見なしている場合に使用されます。

カテゴリ

CWE-546：疑わしいコメント

CVSS v3.1 スコア

5.10

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、ユーザー入力値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。これにより、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。

BDSA によると、ある関数が HTML 文字列を解析して DOM ノードの配列にします。イベント属性に埋め込まれたスクリプトがこの関数に渡されると、ただちに実行されます。このため、信用できない入力をこの関数に渡す前に適切にサニタイズしておかないと、関数の呼び出し元が XSS 攻撃を受ける可能性があります。攻撃者は、悪意のある HTML を細工して攻撃対象に送り込むことで、この脆弱性を悪用できます。これを処理すると、その中に含まれる任意の web スクリプトがシステム上で実行されてしまいます。

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。

カテゴリ

CWE: 79：web ページ生成時における入力の不適切な無害化（「クロスサイト・スクリプティング」）

CAPEC-588：DOM ベース XXS

CVSS v3.1 スコア

8.60

BDSA-2015-0567

これも古い脆弱性で、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。

カテゴリ

CWE カテゴリ A9：既知の脆弱性を含むコンポーネントの使用

CAPEC-251：ローカル・コード・インクルージョン（CAPEC [Common Attack Pattern Enumeration and Classification] では、包括的なスキーマと分類法と共に攻撃パターンをカタログ化して公開しています）

CVSS v3.1 スコア

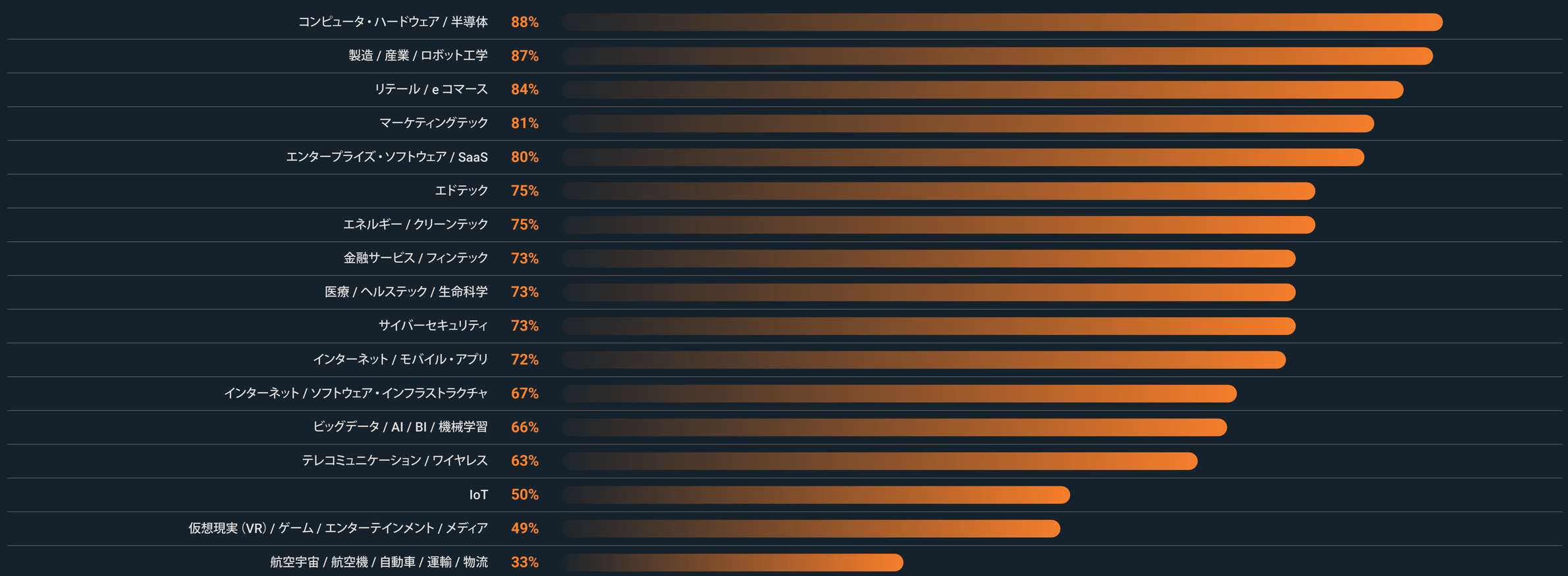
7.9

業種別に見た脆弱性

高リスクに分類される脆弱性（深刻度スコアが 7 以上のもの）を含むコードベースの割合が最も多かったのはコンピュータ・ハードウェア / 半導体の 88% で、製造 / 産業 / ロボット工学の 87%、リテール /e コマースの 84% が僅差で続いています。

その他のセクターでも高リスク脆弱性は多く見つかっており、最も少ない航空宇宙 / 航空機 / 自動車 / 運輸 / 物流でさえコードベースの 1/3 に高リスク脆弱性が含まれていることが憂慮されます。図 1 に示したように、オープンソースは調査したすべての業種のコードベースに含まれており、どの業種でもコードベースの大半をオープンソースが占めています。図 3 が示すように、これらのコードベースには既知のオープンソース脆弱性も非常に多く含まれており、悪用可能な状態のままとなっています。

図 3：業種別に見た高リスク脆弱性

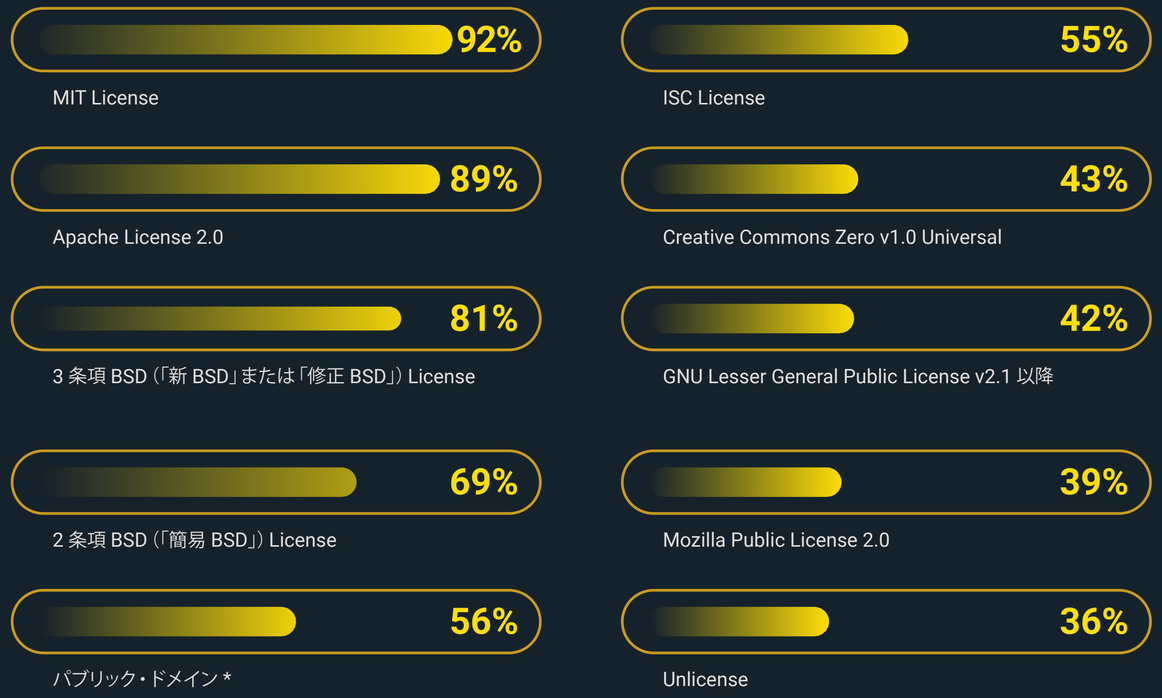


オープンソース・ライセンス

ソフトウェア・サプライチェーン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に関するコンプライアンスも必要です。オープンソース・コンポーネントやライブラリを使用してソフトウェアを開発している場合、これらのコンポーネントにオープンソース・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。ソフトウェアの中にライセンス・コンプライアンスの違反が 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、製品の市場投入の遅れなどを招くことがあります。

Black Duck 監査サービス・チームが 2023 年に監査したコードベースのうち、半分以上（53%）にライセンスの競合があるオープンソースが含まれていました。

図 4：最も多く見つかった上位 10 のライセンスを含むコードベースの割合



* パブリック・ドメインであることを表明しているが、Unlicense や Creative Commons など特定のパブリック・ドメイン・ライセンスを使用していないコンポーネント

2023 年に Black Duck 監査サービスが監査したオープンソースの 92% に MIT License が見つかりました。MIT License はプロプライエタリ・ソフトウェアでの再利用を認めている寛容型ライセンスのため互換性が高く、他のソフトウェア・ライセンスと競合するリスクはあまりありません。企業がソフトウェアにサードパーティ・コンポーネントを組み込む場合、そのコンポーネントはほぼ間違いなく MIT、Apache、BSD、ISC、Unlicense などの有名な寛容型ライセンスで提供されているはずです。

なお、「低リスク」のような用語は単なるガイドラインであり、各ライセンスが適用されるオープンソース・ソフトウェアを使用するかどうかの判断材料とすべきではありません。例えば、Apache 2 ソフトウェアは一般的に低リスク・ライセンスを使用していると考えられており、GNU General Public License 3.0（GPLv3）でライセンスされるプロジェクトに含めることはできますが、GPLv3 ソフトウェアを Apache プロジェクトに含めることはできません。Apache ソフトウェア財団のライセンス理念と GPLv3 の作者の著作権法の解釈の結果、このようなケースでは 2 つのライセンス間に互換性がありません。この場合、組織の政策・法務チームに相談してライセンス・コンプライアンスに関する指導を受けるのが開発者にとって最も安全な戦略です。

2023 年の監査で、ライセンス競合の原因として最も多かったのは Creative Commons 系ライセンスでした。中でも、Creative Commons ShareAlike 3.0（CC-SA 3.0）だけでライセンス競合の 17% を引き起こしています。

Black Duck 監査では、スニペット（ソースコードにコピー & ペーストされた小さなコード・ブロック）も非常に多く見つかっています。これらの多くは人気ブログ・サイト「Stack Overflow」から引用されていますが、このサイトで一般にアクセス可能なユーザーからの投稿にはすべて自動的に Creative Commons ShareAlike ライセンスが適用されます。残念なことに、この包括ライセンスはサイトに投稿されたコード・スニペットにも適用されます。「残念」というのは、これらのライセンスはソフトウェアに適用することを想定されていないためです。事実、Creative Commons の web サイトにある FAQ にも「Creative Commons ライセンスをソフトウェアに使用することは推奨しません」と明記されています。CC-SA ライセンスは、状況によっては GNU Public License と同様の「感染力」（コピーレフト・ライセンスの著作物から派生した著作物にも同じコピーレフト・ライセンスが適用されること）があるように読むことができ、法的な観点で懸念となる可能性があります。

ライセンス・リスクを理解する

米国など多くの裁判管轄地では、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。

最も寛容なオープンソース・ライセンスでさえ、そのソフトウェアの使用と引き換えにユーザーが負う義務を規定しています。コードベースに含まれるオープンソースのライセンスが、コードベース全体のライセンスと競合するような場合、ライセンス・リスクが生じる可能性があります。オープンソース・プロジェクトに適用されるコピーレフト・ライセンスとして最も一般的なのが GNU General Public License（GPL）です。商用のクローズドソース・ソフトウェアに GPL でライセンスされるコードが含まれていると、競合が発生します。

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、ライセンシーにとって望ましくない条件が課せられたり、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ることがあります。カスタマイズしたライセンスの代表例と言えるのが、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。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。

2023 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・ライセンスを使用したものは 31% あり、これは前年とほぼ同じ割合でした。オープンソース監査では、Stack Overflow とは異なり、利用規約やソフトウェア利用条件が明記されていないサイトからのコード・スニペットが見つかることも珍しくありません。オープンソース・コードにライセンス条項が存在しない理由としてもう 1 つよくあるのが、開発者がコード・スニペットを使用する際に、そのスニペットに関連するライセンスを含めるのを忘れてしまうことです。また、最近では AI コーディング支援ツールの利用拡大によって、関連するライセンスの欠落したコードの問題が起こるケースも増えています（次のセクションを参照）。

図 5：競合が見つかった上位 10 のライセンスを含むコードベースの割合

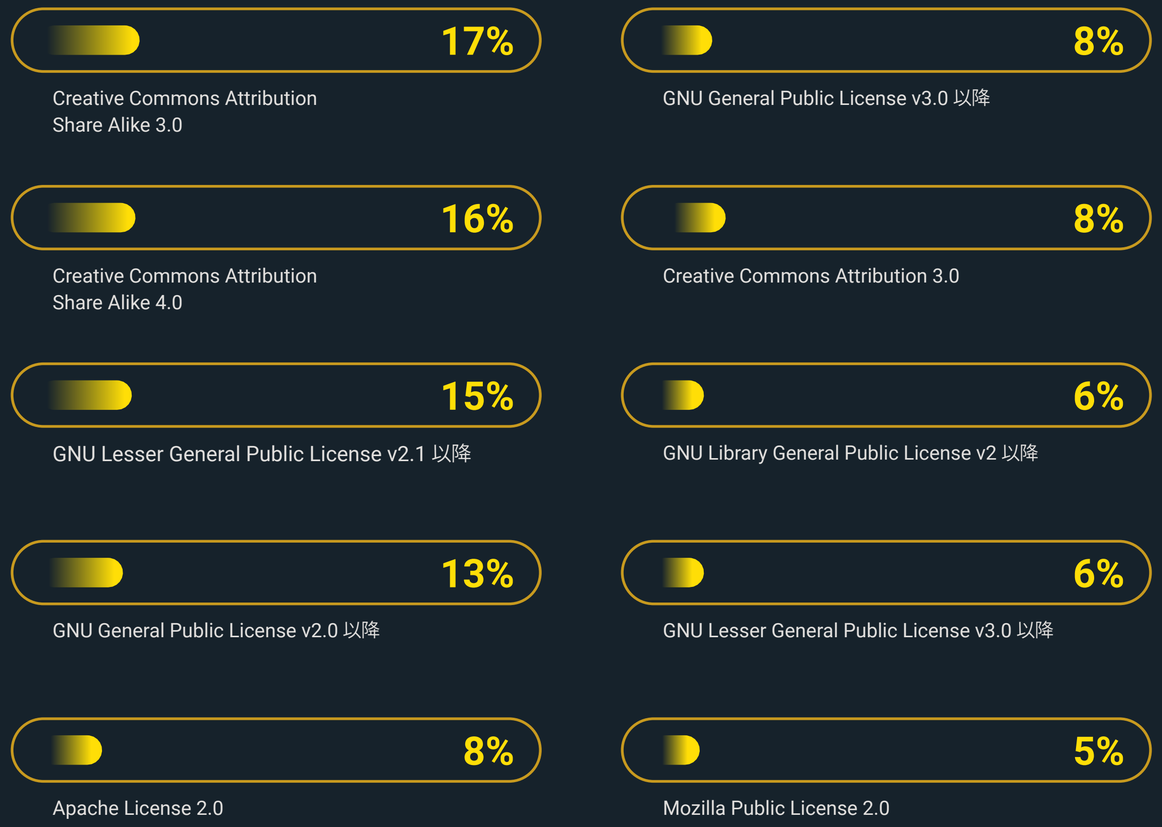
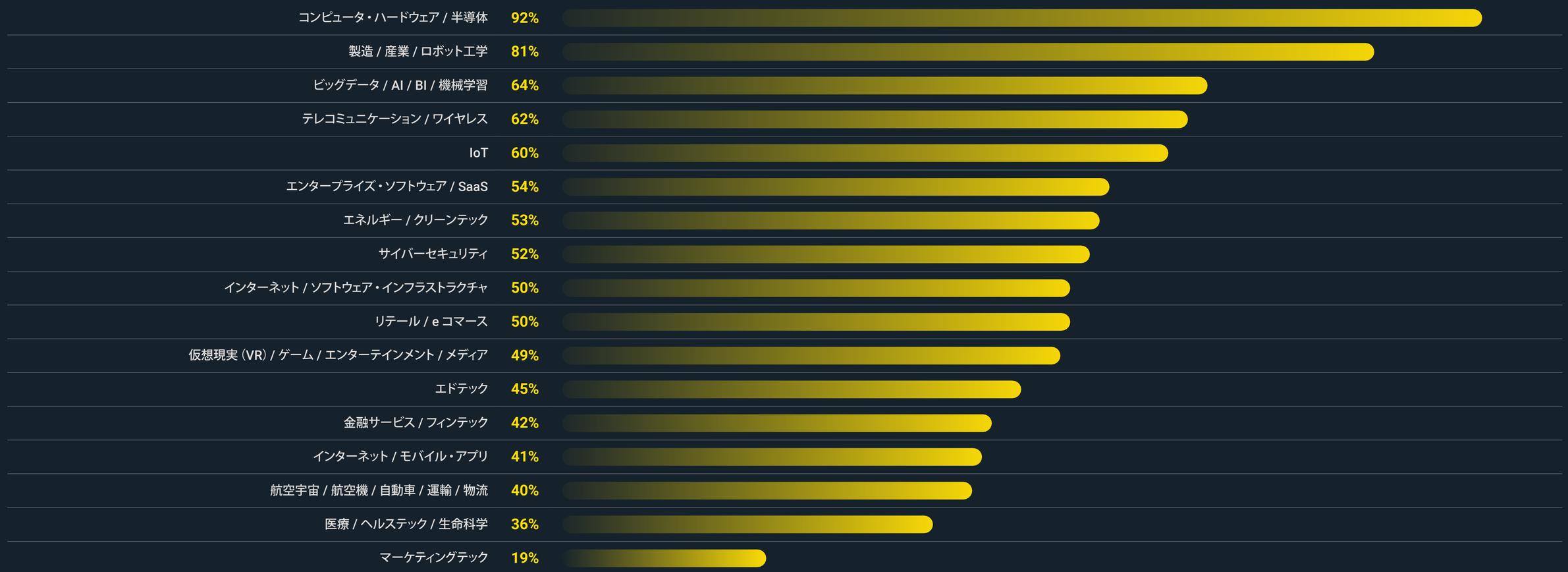


図 6 に示すように、いくつかの業種ではオープンソース・ライセンスのリスクが特に高くなっています。これにはいくつかの理由が考えられます。

- 高い割合でライセンスの競合が見つかった業種の多くは、ソフトウェアをオンプレミス型製品としてライセンス・頒布する傾向があること。制限の強いライセンスの多くは、このような方法で頒布されるソフトウェアに対してのみ適用されます。ライセンスの競合が少ない業種は、サブスクリプション・ベースや SaaS タイプのデプロイが多いと考えられます。これらは伝統的に「頒布」とは見なされず、同じライセンス条件であってもその適用を受けません。
- 半導体 / ハードウェア企業はソフトウェアとファームウェアへの依存度が高く、その多くがオープンソース・コードを組み込んでいること（図 1 参照）。複雑なシステム・オン・チップ（SoC）デザインには、さまざまなソースから数百万行ものコードが組み込まれています。これほどの規模でライセンスと義務を追跡するのは非常に困難であると考えられます。
- オープンソースは、ハードウェア製品に不可欠な低レベル・システム・ソフトウェア、ファームウェア、ドライバーなどにも非常に多く含まれていること。これらの多くには GPL タイプの「コピーレフト」ライセンスが適用されており、頒布する場合には共有に関する厳しい要件が課せられます。
- 企業間、およびハードウェア設計者とメーカー間のソフトウェア・サプライチェーンでファームウェア、ドライバー、ツールが共有されることで、SBOM のインベントリを使用した出所やライセンスの追跡がなされないまま、オープンソースが拡散している。

図 6：業種別に見たライセンスの競合を含むコードベースの割合



AI コーディング支援ツールによってもたらされるセキュリティおよび IP コンプライアンスのリスクから身を守る方法

AI コーディング支援ツールが使われるようになって、生成されたコードの所有権、著作権、およびライセンスが新たな問題となっています。一例として、GitHub Copilot（開発者がコードを入力するとオートコンプリート・スタイルの候補を提示するクラウド・ベースの AI ツール）が著作権法とソフトウェア・ライセンス要件の両方に違反しているとして、GitHub、Microsoft、OpenAI に対する集団訴訟が起こされています。さらにこの訴訟では、Copilot が提案するコードはライセンスされた著作物を帰属表示や著作権表示なしに、あるいは元のライセンス条項に違反して使用しているとの主張もなされています。

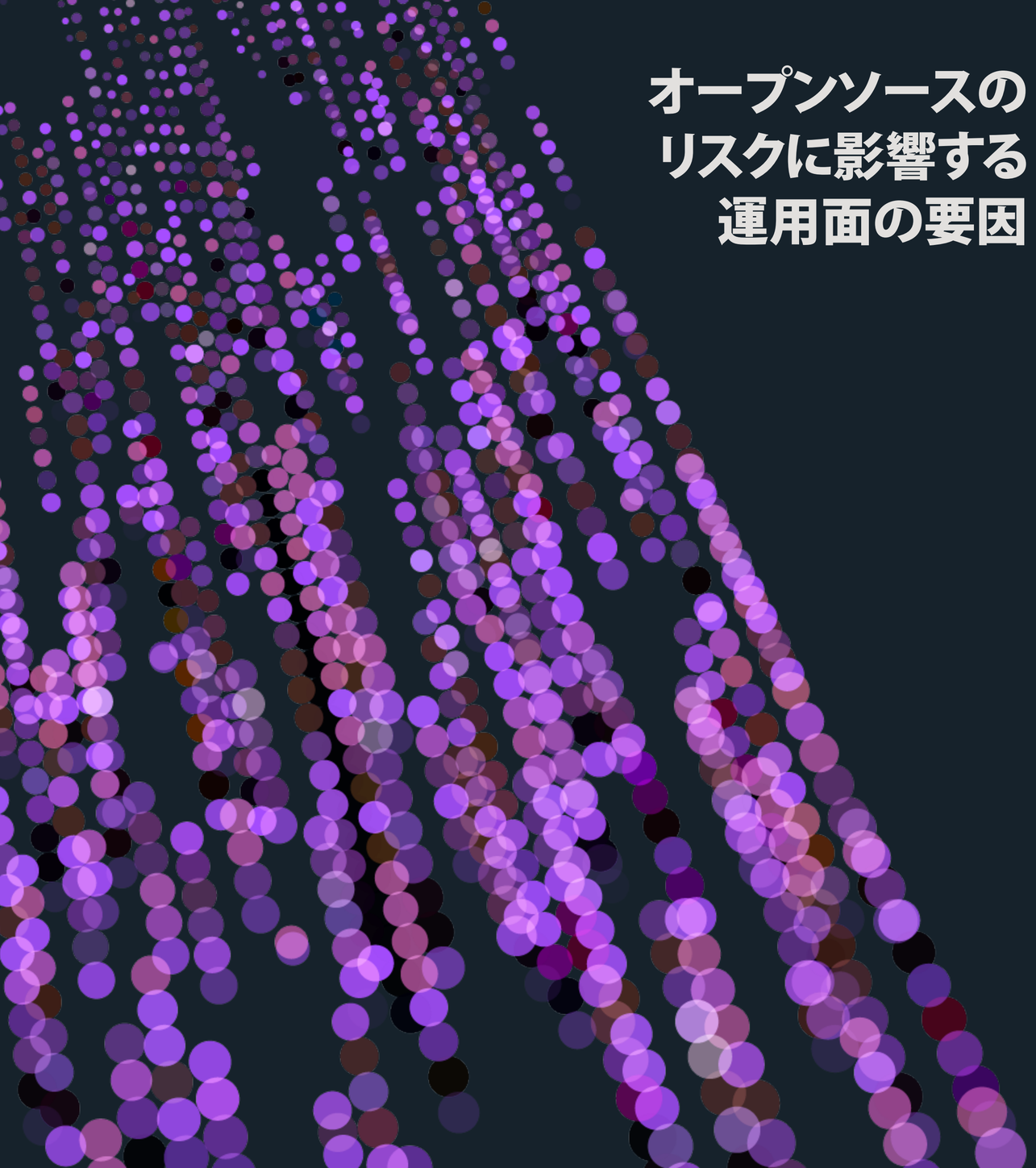
Copilot に対するこの訴訟は、AI 生成コードを取り巻く法律面での複雑さを浮き彫りにしています。ソフトウェア開発者がライセンスまたは著作権法違反に問われる事態を回避しようとするなら、この問題が裁判所または政府の判断によって解決されるまで AI コーディング支援ツールの使用を控えるのが最も確実です。

それでも AI 支援ツールを使用したいのであれば、不必要なリスクを避けるように十分な注意を払う必要があります。少なくとも、AI が提案するコードにオープンソース・ライセンスの適用を受けるソースコードも含まれるのか、そして含まれるのであれば、そのコードを強調表示したり、提案から完全に除外したりすることが可能かどうかを AI ツール・ベンダーに問い合わせる必要があります。

もう 1 つの解は、コード・スキャナーを使用することです。例えば、ブラック・ダックの Black Duck にはコード・スニペット解析機能があり、ソースコードをスキャンして、出所となっているオープンソース・プロジェクトをコード行ごとに照合することができます。これにより、開発チームは AI コーディング支援ツールによって挿入されたオープンソース・コードに適用されるライセンスとそれに伴う使用条件を特定できるようになります。

ソフトウェア・サプライチェーンのガバナンスにおけるオープンソース・ライセンス管理のベスト・プラクティス

- ソフトウェアに含まれるすべてのサードパーティ・ソフトウェア・コンポーネント（オープンソースと商用ソフトウェアの両方を含む）の完全なインベントリを作成する。
- AI コーディング支援ツールが生成するコードは、ライセンス違反や知的財産権侵害を引き起こす可能性があることに注意する。
- すべてのコンポーネントのライセンス条件を評価し、これらが製品の意図する使用と競合しないか確認する。
- ライセンスの種類によっては互いに競合するものがあるため、各種コンポーネントのライセンス同士の互換性を確認する。
- 自動スキャン・ツールを使用し、各コンポーネントのライセンスの義務と制限を特定および追跡する。
- 定期的なライセンス・スキャンやライセンス・コンプライアンス手続きの定期的な見直しなど、継続的なライセンス・コンプライアンス徹底のプロセスを確立する。
- 新しいライセンスや見慣れないライセンスに対するレビュー・プロセスおよびワークフローを確立する。
- ライセンス・クリアランス作業（特定のコンポーネントのライセンスを製品で使用することが許容可能かどうかを企業が判断するプロセス）を適切に優先順位付けして実行できるように、法務、技術、およびビジネスに関するステークホルダー間での効果的なコミュニケーションを確立する。
- コンプライアンス活動を記録に残して将来の監査を円滑にするため、ライセンスの評価やコンプライアンス手順などすべてのライセンス・クリアランス活動を文書化する。



オープンソースのリスクに影響する運用面の要因

オープンソースを使用する場合は、活発なコミュニティに支えられたコンポーネントのみを使用するのが理想です。例えば、Linux は数百の団体に属する数千人もの開発者によって日々改善が続けられています。しかし、Black Duck 監査サービス・チームがオプションのリスク診断を実施した 936 のコードベースのうち、過去 2 年間に新たな開発活動実績のなかったオープンソースを含むものが 49% ありました。特に小規模なプロジェクトの場合、プロジェクトのメンテナンスが止まっているということは、機能のアップデート、コードの改良、見つかったセキュリティ問題の修正がまったく行われていないことになります。

これは、オープンソース・プロジェクトでは珍しい問題ではありません。2022 年の時点でメンテナンスされていた Java および JavaScript のオープンソース・プロジェクトのうち、20% 近くが 2023 年にはメンテナンスが止まっており、これらのプロジェクトが脆弱性や悪用にさらされているという報告もあります。オープンソースは、その大部分がボランティアのコントリビューターとメンテナーの活動の産物です。Microsoft、RedHat、Google など一部の組織はオープンソース・プロジェクトへの参加やそのメンテナンスを奨励するインセンティブ・プログラムを実施していますが、そのような組織はごくわずかです。メンテナーによるプロジェクトのメンテナンスが止まると、セキュリティ・リスクの増大という結果を招きます。

2023 年に分析したコードベースのうち、**88%** が**リスク診断**も実施

リスク診断を受けたコードベースのうち、**49%** が**過去 2 年間に新たな開発活動実績のなかった**オープンソースを使用

オープンソース利用者に求められるメンテナンス・プラクティスの改善

Black Duck 監査サービス・チームが 2023 年にオプションのリスク診断を実施した 936 のコードベースのうち、最新バージョンよりも 10 バージョン以上前のコンポーネントを使用しているものが 91% ありました。

オープンソース利用者がコンポーネントを最新の状態にしていないのには、それなりの理由が存在することもあります。コンポーネントが最新の状態でないことに気付いていたとしても（残念ながら、そのようなケースは多くありませんが）、新バージョンを適用することで得られるメリットよりも、意図しない結果を招くリスクの方が大きいと開発チームが判断することもあります。例えば、組み込みソフトウェアであれば、外部ソースからしか混入経路のないエクスプロイトはそれほど大きなリスクにならないと考えることもできます。

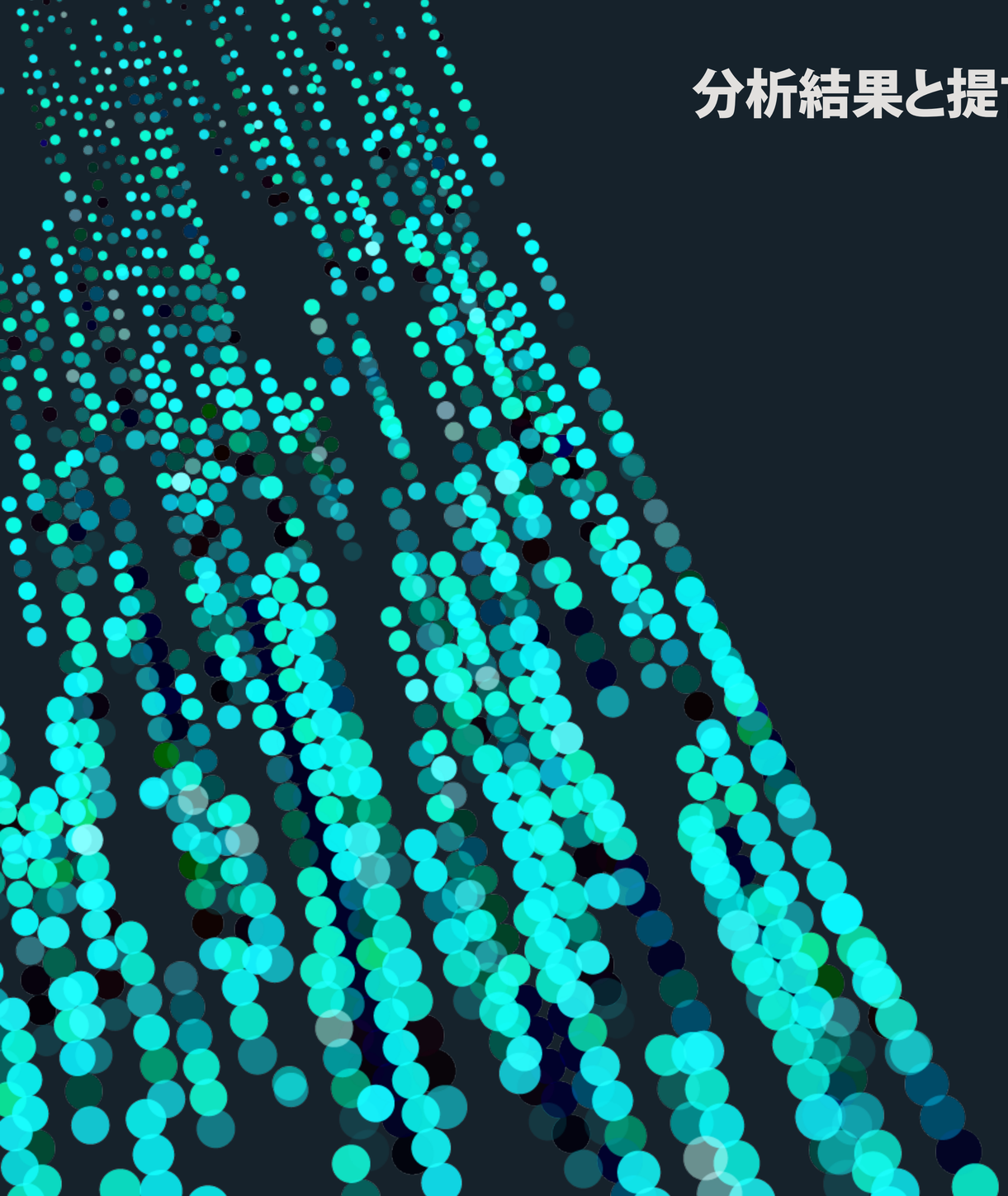
あるいは、時間やリソースの不足が原因の場合もあります。多くのチームは新規コードの開発とテストで既に手一杯で、特に深刻度の高い問題でなければ、既存のソフトウェアのアップデートを後回しにすることがあります。ブラック・ダックが発行したレポート「世界の DevSecOps の現状 2023」では、調査した 1,000 人の IT セキュリティ専門家のうち、デプロイ済みアプリケーションの重大なセキュリティ・リスク / 脆弱性へのパッチ適用に 3 週間かかるとした回答が 28%、1 カ月かかるとした回答が 20% ありました。これらの数字は、オープンソースだけでなくプロプライエタリ、商用、およびサードパーティ・ソフトウェアを含むすべての脆弱性に関するものです。

これまで 10 年近くにわたって 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、その管理についても異なるアプローチが必要です。例えば、パッチの取り扱いについても商用ソフトウェアとオープンソース・ソフトウェアではまったく違います。通常、商用ソフトウェアを購入する際は、ベンダー管理プログラムの一環として何らかのレビューが必要とされます。これに対し、オープンソースは開発者が自己裁量でダウンロードできることがしばしばです。組織によっては、「寛容型ライセンスのコードのみを使用すること」といったガイドラインが用意されていることもありますが、そのような組織は少数派です。

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（緊急アップデートのことが多い）がダウンロード可能になるとベンダーから通知を受け取る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、利用者自身がコンポーネントのステータスを常に確認し、新しいバージョンが利用可能になったらダウンロードする必要があります。

現実的に考えて、使用しているオープンソースのバージョンを常に把握する方法は 1 つしかありません。それは、オープンソースの正確かつ包括的なインベントリを作成し、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自動プロセスによって監視することです。

分析結果と提言



個人開発者か大企業かを問わず、リスク軽減を目的としたソフトウェア・サプライチェーンのセキュリティ・プラクティスの維持管理には全員が責任を負います。ソフトウェア・サプライチェーンに対する攻撃が増えている中、オープンソースの使用、コンポーネント、および依存関係を効果的に管理することがリスク管理にとってこれまで以上に死活的な意味を持つようになっています。自社製品にオープンソースを使用している組織（すなわち、このレポートで見てきたように事実上すべての組織）は、セキュア・ソフトウェア開発プラクティスの一環としてオープンソースのリスクを事前対処的に管理することが求められます。

米国サイバーセキュリティ・社会基盤安全保障庁（CISA）が 2023 年 12 月に発表した「Securing the Software Supply Chain: Recommended Practices for Managing Open Source Software and Software Bill of Materials（ソフトウェア・サプライチェーンのセキュリティ対策：オープンソース・ソフトウェアとソフトウェア部品表の管理に推奨されるプラクティス）」には、ソフトウェア・サプライチェーンでオープンソースを使用する際の詳細なガイドラインが示されています。これには、以下の内容が含まれます。

- **オープンソースを製品のセキュア・ビルド・プロセスに統合する際は、内製コンポーネントの場合と同じポリシーおよびプロセスを使用する。**
  セキュリティ・チームは通常、オープンソースに関するセキュリティ・ポリシー、プロセス、およびツールを定義しています。開発者は、事前審査済みの内部リポジトリから必要な機能を備えたコンポーネントを選ぶのが理想です。このようなコンポーネントは、最初に SCA セキュリティ解析ツールによる脆弱性診断を受けており、開発 / ビルド・ステージでもさらにスキャンを実行することにより、問題を可能な限り早く見つけることができます。

- **オープンソースのアップデートを追跡し、問題と脆弱性を監視する。**
  脆弱性が特定されたら、影響を受けるソフトウェアを評価し、そのコンポーネントの使用率と製品内で使用されているかどうかを特定します。このようなコンポーネントはアップデートの必要があり、コンポーネントのメンテナンスが既に止まっている場合は、代替ソリューションを検討することが強く推奨されます。

- **SBOM を使用する。**
  コードを正確かつ完全に管理するには、ソフトウェアにどのようなコンポーネントが含まれているかを理解することが不可欠です。SBOM は、ソフトウェアに含まれるコンポーネントの詳細およびサプライチェーンの関係を記述した正式な記録となるものです。SBOM はソフトウェアの透明性を高め、コンポーネントの出所を文書化します。脆弱性管理に関しては、SBOM は脆弱性の特定と修正に役立ちます。コード品質の観点からは、SBOM の存在はサプライヤーがソフトウェア開発ライフサイクル全体にわたってセキュア・ソフトウェア開発プラクティスを使用していることの表れと考えることもできます。

米国大統領令 EO 14028「Improving the Nation's Cybersecurity（国家のサイバーセキュリティの改善）」では、組織はソフトウェアの購入者に対して SBOM を直接提供するか web サイトに公開するように求められることがあり、政府 / 非政府機関はソフトウェア製品が SBOM の最小要素の要件に適合していることを確認するために SBOM をレビューするように求められることがあると述べています。また、この大統領令は米国商務省電気通信情報局（NTIA）に対し、SBOM に必要な活動とデータ、および SBOM の要件を満たすフォーマット例を示した「ソフトウェア部品表（SBOM）の最小要素」を発行するよう指示しました。最も広く使用されている機械可読性のある SBOM フォーマットとして、SPDX と CycloneDX の 2 つが特定されています。2023 年 6 月に、米国行政管理予算局（OMB）は以前の覚書を更新する覚書 OMB 23-16 を公表しました。この中で、連邦政府機関が次のものを要求することを認めています。

- ソフトウェアの重大性または各機関で決定したその他の基準に基づく SBOM の提供
- NTIA で定義されたいずれかのフォーマットの SBOM の提供

セキュア・ソフトウェア開発フレームワークを構築する

ソフトウェア製作者は、顧客とユーザーのためにソフトウェア・サプライチェーンのセキュリティ対策をとる上で極めて重要な役割を担います。米国国立標準技術研究所（NIST）は、標準化された方法で安全にソフトウェアを開発する際にベースラインとなる一連のプラクティスをセキュア・ソフトウェア開発フレームワーク（SSDF）としてまとめています。NIST SSDF への準拠を証明することは、米国政府が直接または間接的に調達するすべてのソフトウェアに対する要件となる見込みであることが米国政府によって示されており、近い将来、ソフトウェア・サプライヤーは SSDF への適合を自己証明することが必要になると考えられます。

ブラック・ダックの SSDF 準備状況評価（SSDF Readiness Assessment）サービスなど、組織のソフトウェア開発プラクティスが SSDF のプラクティスとタスクに合致しているかどうかを判定してくれる診断ツールを使用すると、自社のソフトウェア開発プロセスが SSDF の標準に準拠していることを確実に証明できます。同様に、ブラック・ダックの SBOM サービスは、Black Duck 監査サービスのプロセスに基づいてソフトウェアの完全なセキュリティ監査を実施して SBOM を生成します。これは、まだ独力で SBOM を生成する能力がなく、ベースライン SBOM を必要としている組織にとって価値のあるサービスです。

規制上または契約上の理由から SBOM の作成が義務付けられているソフトウェア製作者は、監査済み SBOM の提供を求められることがあります。また、ソフトウェア利用者側でもサプライヤーが生成した SBOM を監査したい場合があります。いずれの場合も、ソフトウェア監査の分野で定評のある、信頼のおける第三者が必要です。ブラック・ダックの SBOM 監査 / 検証サービスは、Black Duck 監査サービスの実績あるプロセスに基づいてソフトウェアを監査し、クライアントが生成した SBOM がサプライチェーンを正確に反映しているかどうかを確認します。

コードの中身を把握する

以下に、ここまでの内容をまとめます。

- スキャンした 1,000 以上のコードベースのうち、96% にオープンソースが含まれていた
- 全ソースコードおよびファイルの 77% がオープンソースを起源としていた
- コードベースの 53% にオープンソース・ライセンスの競合が見つかった
- セキュリティ・リスク診断を受けたコードベースの 84% に脆弱性が見つかり、74% に高リスク脆弱性が見つかった
- セキュリティ・リスク診断を受けたコードベースの 91% が、最新バージョンよりも 10 バージョン以上前のコンポーネントを使用していた

ソフトウェアを開発しているにせよ、利用しているにせよ、そのソフトウェアにはまず間違いなくオープンソース・コンポーネントが含まれています。では、どのようなコンポーネントが含まれており、それらにセキュリティまたはライセンス・リスクがあるかどうかを正確に把握できているでしょうか。コードベースの 96% 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。リスク診断を受けたコードベースの 91% が最新バージョンよりもかなり古いオープンソースを使用していることを考えると、オープンソース利用者はコードをもっと積極的に最新の状態に維持する必要があります。それが人気の高いオープンソース・コンポーネントであれば、なおさらです。

オープンソースを最新の状態に維持することは、チームによるコード開発と同じ優先順位で扱う必要があります。コードに含まれるコンポーネント、およびそのバージョン、ライセンス、出所などの詳細を記述した SBOM を作成し、維持管理することが必要です。また、特にメンテナーによる活動が活発な人気のあるプロジェクトのオープンソース・ライブラリを使用している場合は、定期的なアップデート・サイクルを確立することが求められます。

コードを包括的に可視化し、予防的なソフトウェア・ハイジーンのプラクティスを徹底しなければ、ソフトウェアはオープンソース脆弱性を悪用した潜在的な攻撃や IP コンプライアンスの問題にさらされます。大切なのは、コードの中身を確実に把握することです。そのために、まずは自動 SCA ツールを使用してセキュリティ、コード品質、ライセンスの問題を SDLC の早期段階で見つけることから始めてください。

コードを包括的に可視化し、予防的なソフトウェア・ハイジーンのプラクティスを徹底しなければ、ソフトウェアはオープンソース脆弱性を悪用した潜在的な攻撃や IP コンプライアンスの問題にさらされます。

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。

用語

コードベース
アプリケーションまたはサービスを構成するコードおよび関連ライブラリ。

バイナリ解析
静的解析の一種。ソースコードを入手できないアプリケーションの内容を特定します。

CWE（共通脆弱性タイプ一覧）
ソフトウェアとハードウェアの弱点のタイプを 3 つの階層にまとめた、コミュニティ作成によるリスト。CWE には 600 を超えるカテゴリがあり、これにはバッファー・オーバーフロー、パス / ディレクトリ・ツリー・トラバーサル・エラー、レース・コンディション、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、安全でない乱数などのクラスが含まれます。

CVE（共通脆弱性識別子）
一般公開されている情報セキュリティの欠陥をリストにしたもの。

Black Duck Security Advisory（BDSA）
オープンソース脆弱性に関するタイムリーで一貫性のある詳細な情報。BDSA は、ブラック・ダックの顧客に向けてオープンソース脆弱性およびアップグレード / パッチ・ガイダンスの補足情報をいち早く提供します。BDSA では、脆弱性の即日通知、具体的な軽減策のガイダンスと回避策の情報、深刻度スコア、参考情報など多くのものが提供されます。

ソフトウェア・コンポーネント
開発者がソフトウェアの部品として追加できる作成済みコード。カレンダーなどのユーティリティの場合もあれば、アプリケーション全体をサポートする包括的なソフトウェア・フレームワークの場合もあります。

依存ファイル
あるソフトウェアが別のソフトウェア・コンポーネントを使用する場合、そのソフトウェアの動作は使用するコンポーネントに依存することになるため、そのコンポーネントを依存ファイルと呼びます。1 つのアプリケーションまたはサービスには多くの依存ファイルが存在することがあり、それらの依存ファイルがさらに別のコンポーネントに依存していることもあります。

スニペット
開発者がコードにコピー & ペーストする再利用可能なコードの小さなブロック。オープンソースのスニペットしかソフトウェアに含まれていなくても、そのソフトウェアのユーザーはそのスニペットに関連するライセンスに従う必要があります。

オープンソース・ライセンス
オープンソース・コンポーネント（またはコンポーネントのコード・スニペット）をソフトウェアで使用する場合、そのコンポーネントの使用や再頒布の方法など、エンドユーザーが従うべき義務を記述した条文。ほとんどのオープンソース・ライセンスは、次の 2 つのカテゴリのいずれかに分類されます。

寛容型ライセンス
使用に関する制限が少ないものを寛容型ライセンスと呼びます。一般に、この種のライセンスでは元のコードの作者に帰属する著作権を表示することが主な条件となります。

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必要があり、要請があった場合は変更を含むソースコードを提供する必要があるという互恵型の義務が含まれます。コピーレフト・ライセンスのオープンソースをソフトウェアに組み込んで使用すると、コードベース全体の知的財産権が疑問視されることがあるため、営利企業からは敬遠されます。

ソフトウェア・コンポジション解析（SCA）
アプリケーション・セキュリティ・ツールの一種で、オープンソース・ソフトウェアの管理プロセスを自動化するために使用します。SCA ツールをソフトウェア開発ライフサイクルに統合すると、コードベースに含まれるオープンソースの特定、リスク管理および緩和のための推奨事項の提示、ライセンスへのコンプライアンス・チェックなどが可能です。

ソフトウェア部品表（SBOM）
コードベースに含まれるソフトウェア・コンポーネントと依存ファイルを網羅したインベントリ（目録）のこと。多くの場合、ソフトウェア・コンポジション解析（SCA）ツールによって生成されます。米国商務省電気通信情報局（NTIA）は、「SBOM にはソフトウェア・コンポーネントと依存ファイルに関する機械可読なインベントリ、これらコンポーネントに関する情報、およびそれらの階層関係を含めるべき」としています。SBOM は企業間やコミュニティ間での共有を想定しているため、内容とフォーマット（人間可読性と機械可読性の両方を備えたもの）に一貫性が求められます。米国政府のガイドラインでは現在、SPDX（Software Package Data Exchange）と CycloneDX の 2 つが承認済みの標準フォーマットとして指定されています。