opentext™

OpenText™ Fortify Static Code Analyzer

ソフトウェアバージョン: 24.2.0

ユーザガイド

ドキュメントリリース日: 2024年5月
ソフトウェアリリース日: 2024年5月

法的通知

Open Text Corporation

275 Frank Tompa Drive, Waterloo, Ontario, Canada, N2L 0A1

著作権表示

商標表示

ドキュメントの更新情報

このドキュメントのタイトルページには、次の識別情報が記載されています。

- ソフトウェアバージョン番号
- ドキュメントリリース日。ドキュメントが更新されるたびに変更されます
- ソフトウェアリリース日。ソフトウェアのこのバージョンのリリース日付を示します

このドキュメントは、OpenText™ Fortify Static Code Analyzer CE 24.2向けに7月 01, 2024に作成されました。最新の更新を確認する場合や、最新のドキュメントを使用しているかを確認する場合は、次のサイトをご覧ください。

https://www.microfocus.com/support/documentation

目次

序文

カスタマサポートへのお問い合わせ

サポートWebサイトにアクセスして、次の作業を実行できます。

- ライセンスとエンタイトルメントの管理
- 技術サポートリクエストの作成と管理
- ドキュメントやナレッジ記事の閲覧
- ソフトウェアのダウンロード
- コミュニティの探索

https://www.microfocus.com/support

詳細情報

Fortifyソフトウェア製品について詳しくは、次のリンクを参照してください。

https://www.microfocus.com/cyberres/application-security

ドキュメントセットについて

Fortifyソフトウェアのドキュメントセットには、すべてのFortifyソフトウェア製品およびコンポーネントのインストールガイド、ユーザガイド、および展開ガイドが含まれています。また、新機能、既知の問題、および最新の更新情報について説明するテクニカルノートとリリースノートもあります。これらのドキュメントの最新バージョンには、次の製品ドキュメントWebサイトからアクセスできます。

https://www.microfocus.com/support/documentation

リリース間のドキュメント更新のお知らせを受け取るには、OpenText FortifyコミュニティのFortify製品のお知らせを購読してください。

https://community.microfocus.com/cyberres/fortify/w/announcements

Fortify製品の機能紹介ビデオ

YouTubeのFortify Unpluggedチャネルで、Fortifyの製品と機能を紹介するビデオをご覧いただけます。

https://www.youtube.com/c/FortifyUnplugged

変更ログ

次の表に、このドキュメントで行われた変更を示します。このドキュメントの改訂版は、変更が製品の機能に影響を与える場合にのみ、ソフトウェアリリース間で発行されます。

ソフトウェアリリース/ドキュメントバージョン	変更点
24.2.0	追加: • "Bazelとの統合" ページ126 • "CMakeとの統合" ページ128 更新: • デフォルトのスキャンポリシーが変更されました(「"分析へのスキャンポリシーの適用" ページ48」を参照) • Fortify Static Code Analyzerと一緒に配布されていないJDKバージョンを指定して変換に使用するためのオプションが追加されました(「"Javaコマンドラインオプション" ページ54」を参照) • デフォルトのPythonバージョンが変更されました(「"Pythonコマンドラインオプション" ページ80」を参照) • デフォルトのPHPバージョンが変更されました(「"PHPコマンドラインオプション" ページ105」を参照) 削除: • -apexオプションおよび対応する設定プロパティは非推奨になり、ApexおよびVisualforceコードの変換に必要ではなくなりました
23.2.0	追加: • "仮想環境でのPythonの変換" ページ82 • "Solidityコードの分析" ページ103 • "Gradle統合のトラブルシューティング" ページ130 • "Fortify Static Code Analyzer Gradleプラグインの使用" ページ130 更新:

ソフトウェアリリース/ドキュメントバージョン	変更点
	• Fortify Static Code AnalyzerをインストールするDockerfileの例を改善しました(「"Fortify Static Code AnalyzerをインストールするDockerfileの作成" ページ34」を参照) • 変換フェーズで生成されるコードに関する考慮事項を追加しました("変換フェーズ" ページ45) • MBSファイルにソースコードを含める手順を追加しました(「"モバイルビルドセッション" ページ46」を参照) • デフォルトのJDKバージョンが1.8から11に変更されました(「"Javaコマンドラインオプション" ページ54」および「"Kotlinコマンドラインオプション" ページ63」を参照) • Javaバイトコードを変換する手順を改善しました(「"Javaバイトコードの変換" ページ59」を参照) • NPM依存関係を除外する手順を改善しました(「"NPM依存関係の変換の管理" ページ76」を参照) • 圧縮されたJavaScriptファイルの変換を有効にするプロパティを追加しました(「"JavaScriptおよびTypeScriptのプロパティ" ページ202」を参照) • ApexおよびPowerShellのルールプロパティを追加しました(「"fortify-rules.properties" ページ217」を参照)。 削除: • Fortify Static Code Analyzerイメージをコンテナとして実行する`-fcontainer`オプションに関する情報は、不要になったため削除しました。
23.1.0	追加: • "分析へのスキャンポリシーの適用" ページ48 • "DartおよびFlutterコードの変換" ページ92 • ルールに使用できる新しいプロパティ(「"Fortify Static Code Analyzerのプロパティファイル" ページ185」および「"fortify-rules.properties" ページ217」を参照) 更新: • Fortify Static Code AnalyzerのインストールをFortifyアプリケーションとツールのインストールから分離しました(「"Fortify Static Code Analyzerのインストール" ページ28」を参照)

ソフトウェアリリース/ドキュメントバージョン	変更点
	• .NETプロジェクトの新しいコマンドライン構文(「"Visual Studioプロジェクトのコマンドライン構文" ページ67」を参照) • 新しいスキャンポリシー分析オプション(「"分析オプション" ページ138」および「"変換と分析フェーズのプロパティ" ページ187」を参照) • フィルタファイルとスキャンポリシーファイルに使用される新しいフィルタタイプ(「"フィルタファイルによる問題の除外" ページ180」を参照)
22.2.0	更新: • Java変換の警告処理に関する説明を更新しました(「"Javaの警告の処理" ページ57」を参照) • Kotlinコードのデフォルトメソッドの互換モードをサポートするオプションを追加しました(「"Kotlinコマンドラインオプション" ページ63」を参照) • Salesforce ApexとVisualforceの分析を改善しました(「"Salesforce ApexおよびVisualforceコードの変換" ページ101」を参照) • Fortify Software Security Centerサーバからセキュリティコンテンツを更新するときに資格情報を提供するためのオプションを追加しました(「"セキュリティコンテンツの更新" ページ152」および「"fortifyupdateコマンドラインオプション" ページ153」を参照) • サンプルプロジェクトは、Fortify Static Code Analyzerおよびアプリケーションインストーラに含まれなくなりました。サンプルは別のパッケージとして提供されます(「"サンプルプロジェクト" ページ23」を参照)。

第1章: はじめに

このガイドでは、Fortify Static Code Analyzerを使用して、ほとんどの主要なプログラミングプラットフォームでコードをスキャンする方法について説明します。このガイドの対象は、セキュリティの監査およびセキュアなコーディングの責任者です。

このセクションでは、次のトピックについて説明します。

Fortify Static Code Analyzer

Fortify Static Code Analyzerは、さまざまな言語でセキュリティ固有のコーディングルールおよびガイドラインの違反を検索する一連のソフトウェアセキュリティアナライザです。Fortify Static Code Analyzer言語技術では、迅速で正確に修正できるように、アナライザで違反を特定して優先順位を付けることができる豊富なデータが提供されています。Fortify Static Code Analyzerでは、よりセキュアなソフトウェアを提供し、セキュリティコードレビューの効率性、一貫性、および完全性を向上させるのに役立つ分析情報が生成されます。お客様固有のセキュリティルールを組み込むことのできる設計になっています。

サポートされている言語、ライブラリ、コンパイラ、およびビルドツールのリストについては、ドキュメント『*Fortifyソフトウェアシステム要件*』を参照してください。

最高レベルでは、Fortify Static Code Analyzerを使用して次の処理を実行します。

1. スタンドアロンプロセスとしてFortify Static Code Analyzerを実行するか、ビルドツールにFortify Static Code Analyzerを統合する
2. ソースコードを中間変換形式に変換する
3. 変換されたコードをスキャンし、セキュリティ脆弱性分析結果を生成する
4. OpenText™ Fortify Audit Workbenchで結果(通常はFPRファイル)を開くか、分析用にOpenText™ Fortify Software Security Centerをアップロードしてスキャンの結果を監査する、または画面に表示された結果を直接操作する。

注: Fortify Audit WorkbenchまたはFortify Software Security Centerで結果を開いて表示する方法については、それぞれ『*OpenText™ Fortify Audit Workbenchユーザガイド*』または『*OpenText™ Fortify Software Security Centerユーザガイド*』を参照してください。

アナライザについて

Fortify Static Code Analyzerは、8つの脆弱性アナライザ(Buffer、Configuration、Content、Control Flow、Dataflow、Null Pointer、Semantic、およびStructural)で構成されています。各アナライザでは、実行された分析に対応するタイプに必要な情報を提供するために特別にカスタマイズされた別のタイプのルールが受諾されます。ルールは、ソースコード内のセキュリティの脆弱性や安全でない状態が発生する可能性がある要素を特定する定義です。

次の表では、各アナライザのリストを表示して説明します。

アナライザ	説明
Buffer	Buffer Analyzerでは、バッファに保持できる以上のデータの書き込みまたは読み取りが発生するバッファオーバーフローの脆弱性が検出されます。バッファはスタックで割り当てられるか、ヒープで割り当てられます。Buffer Analyzerでは、制限付きのプロシージャ間分析を使用して、バッファオーバーフローが発生する原因になる状態であるかどうかが判断されます。バッファへの実行パスが原因でバッファオーバーフローが発生する場合は、Fortify Static Code Analyzerでバッファオーバーフローの脆弱性としてレポートされ、オーバーフローの原因になる可能性がある変数が指摘されます。バッファオーバーフローの発生原因となる変数の値が汚染(ユーザ制御)されている場合は、その値もFortify Static Code Analyzerでレポートされ、どのように変数が汚染されているのかを示すデータフロートレースが表示されます。
Configuration	Configuration Analyzerでは、アプリケーション展開環境設定ファイルの間違い、弱点、およびポリシー違反が検索されます。たとえば、Configuration Analyzerでは、Webアプリケーションでユーザセッションに適切なタイムアウトがチェックされます。Configuration Analyzerでは、正規表現分析も実行されます(「"正規表現分析" ページ49」を参照)。
Content	Content Analyzerでは、HTMLコンテンツのセキュリティ問題とポリシー違反が検索されます。Content Analyzerでは、静的なHTMLページに加えて、動的なHTMLを含むファイル(PHP、JSP、従来のASPファイルなど)でも、これらのチェックが実行されます。

アナライザ	説明
Control Flow	Control Flow Analyzerでは、一連の危険な操作が検出されます。プログラムで制御フローパスを分析すると、Control Flow Analyzerで一連の操作が特定の順序で実行されているかどうかが判断されます。たとえば、Control Flow Analyzerでは、チェックの時刻/使用時刻の問題や初期化されていない変数の時刻が検出され、XMLリーダなどのユーティリティが使用前に適切に設定されているかどうかがチェックされます。
Dataflow	Dataflow Analyzerでは、テイントデータ(ユーザ制御入力)の潜在的に危険な使用が発生する潜在的な脆弱性が検出されます。Dataflow Analyzerでは、グローバルなプロシージャ間テイント伝播分析を使用して、ソース(ユーザ入力のサイト)とシンク(危険な関数呼び出しや操作)間のデータフローが検出されます。たとえば、Dataflow Analyzerでは、ユーザが制御する無制限の長さの入力文字列が静的サイズのバッファにコピーされているかどうかが検出され、ユーザが制御する文字列を使用してSQLクエリテキストが作成されるかどうかが検出されます。
Null Pointer	Null Pointer Analyzerでは、null値が割り当てられたポインタ変数の逆参照が検出されます。Null Pointer Analyzerの検出は、プロシージャ内レベルで実行されます。問題は、null割り当て、逆参照、およびこれらの間のすべてのパスが単一の関数内で発生した場合にのみ検出されます。
Semantic	Semantic Analyzerでは、プロシージャ内レベルで関数とAPIの潜在的に危険な使用が検出されます。この特殊なロジックでは、バッファオーバーフロー、フォーマット文字列、および実行パスに関する問題が検索されますが、これらのカテゴリに限定されません。たとえば、Semantic AnalyzerではJavaで非推奨になった関数が検出され、Javaの関数およびC/C++の安全でない関数(gets()など)が検出されます。
Structural	Structural Analyzerでは、プログラムの構造または定義の潜在的に危険な欠陥が検出されます。Structural Analyzerでは、変数および関数の宣言と使用の両方を含む幅広い範囲が網羅されるため、プログラムの構造を理解することで、検査による検出が難しい場合が多いセキュアなプログラミング手法や技術の違反も特定されます。たとえば、Structural Analyzerでは、Javaサーブレット内のメンバー変数への割り当てが検出され、static final宣言されていないロガーの使用が特定され、述語が常にfalseであるために実行されないデッドコードのインスタンスにフラグが付けられます。

ライセンス

Fortify Static Code Analyzerでは、セキュリティ分析の変換フェーズと分析(スキャン)フェーズの両方を実行するためにライセンスが必要です(これらのフェーズの詳細については、「"分析プロセス" ページ44」を参照)。Fortify Static Code Analyzerのライセンスを取得する方法の詳細については、ドキュメント『*Fortify ソフトウェアシステム要件*』を参照してください。

Fortifyライセンスファイル(`fortify.license`)が必要です。オプションで、Fortify License and Infrastructure Managerを使用してFortify Static Code Analyzerの同時ライセンスを管理できます。LIMで管理される同時ライセンスを使用すると、複数のFortify Static Code Analyzerのインストールで単一のライセンスを共有できます。Fortify Static Code Analyzerのライセンスを使用してLIMを設定する方法については、『*OpenText™ Fortify License and Infrastructure Manager インストールおよび使用ガイド*』を参照してください。Fortify Static Code AnalyzerからLIMライセンスを管理する方法については、「"LIMライセンスディレクティブ" ページ148」を参照してください。

有効期限が切れたライセンスの更新

Fortify Static Code Analyzerのライセンスは1年ごとに有効期限が切れます。Fortifyのライセンスファイルを取得する方法については、ドキュメント『*Fortify ソフトウェアシステム要件*』を参照してください。

有効期限が切れたライセンスを更新するには:

- 更新されたFortifyライセンスファイルを*<sca_install_dir>*フォルダに保存します。

有効期限が切れたLIMで管理される同時ライセンスを更新する方法については、『*OpenText™ Fortify License and Infrastructure Manager インストールおよび使用ガイド*』を参照してください。

Fortify Software Security Content

Fortify Static Code Analyzerでは、ルールのナレッジベースを使用して、静的分析用のコードベースにセキュアなコーディング標準が強制的に適用されます。Fortify Software Security Centerは、変換と分析の両方に必要です。Fortify Static Code Analyzerのインストール時に、セキュリティコンテンツをダウンロードしてインストールできます(「"Fortify Static Code Analyzerのインストール" ページ28」を参照)。また、インストール後のタスクとしてfortifyupdateコマンドラインツールを使用して、Fortify Software Security Contentをダウンロードしたり、あらかじめダウンロードしておいたものをインポートしたりすることもできます(「"Fortify Software Security Contentの手動インストール" ページ34」を参照)。

Fortify Software Security Content (セキュリティコンテンツ)は、Fortify Secure Coding Rulepacksと外部メタデータで構成されています。

- Fortify Secure Coding Rulepacksには、一般的な言語およびパブリックAPIに対する一般的なセキュアコーディングのイディオムが記述されています。
- 外部メタデータには、Fortifyカテゴリから代替カテゴリ(CWE、OWASP Top 10、PCIなど)へのマッピングが含まれています。

Fortifyでは、Fortify Static Code AnalyzerとFortify Secure Coding Rulepacksの機能に追加するカスタムルールを記述する機能が提供されています。たとえば、場合によっては、専有セキュリティガイドラインを適用したり、すでにFortify Secure Coding Rulepacksの対象ではないサードパーティのライブラリや事前コンパイルされたその他のバイナリを使用するプロジェクトを分析したりする必要があります。また、外部メタデータをカスタマイズして、さまざまな分類体系(内部アプリケーションのセキュリティ基準や追加のコンプライアンス義務など)にFortifyの問題をマップすることもできます。独自のカスタムルールまたはカスタムの外部メタデータを作成する方法については、『*OpenText™ Fortify Static Code Analyzerカスタムルールガイド*』を参照してください。

Fortifyでは、セキュリティコンテンツを定期的に更新することが推奨されています。fortifyupdateを使用すると、最新のセキュリティコンテンツを取得できます。詳細については、「"セキュリティコンテンツの更新" ページ152」を参照してください。

Fortify ScanCentral SAST

OpenText™ Fortify ScanCentral SASTを使用すると、Fortify Static Code Analyzerの分析フェーズをビルドコンピュータから、この目的のためにプロビジョニングされたコンピュータのコレクションにオフロードすることでリソースを管理できます。ほとんどの言語では、Fortify ScanCentral SASTで変換フェーズと分析(スキャン)フェーズの両方を実行できます。Fortify Software Security Centerのユーザは、FPRファイルをサーバに直接出力するようにFortify ScanCentral SASTに指示できます。Fortify Static Code Analyzerのインストール時にFortify ScanCentral SASTクライアントをインストールすることもできます。

次の2つの方法のいずれかでコードを分析できます。

- ローカルビルドコンピュータ上で変換フェーズを実行し、モバイルビルドセッション(MBS)を生成します。MBSファイルを使用して、Fortify ScanCentral SASTでスキャンを開始します。このプロセスでは、ビルドコンピュータを解放することに加えて、必要に応じてリソースを追加することで、ビルドプロセスを中断することなく、システムを拡張することができます。
- Fortify ScanCentral SASTの変換でサポートされている言語でアプリケーションが記述されている場合は、分析の変換フェーズと分析(スキャン)フェーズをFortify ScanCentral SASTにオフロードできます。サポートされている特定の言語については、ドキュメント『*Fortifyソフトウェアシステム要件*』を参照してください。

Fortify ScanCentral SASTを設定および使用する方法の詳細については、『*OpenText™ Fortify ScanCentral SASTインストール、設定、および使用ガイド*』を参照してください。

Fortify Static Code Analyzerアプリケーションとツール

OpenTextでは、Fortify Static Code Analyzer、Fortify ScanCentral SAST、およびFortify Software Security Centerと統合するアプリケーションとツール(Fortify Secure Code Pluginsを含む)が用意されています。次の表では、Fortifyアプリケーションとツールインストーラを使用してインストールできるアプリケーションについて説明します。Fortifyアプリケーションとツールのインストール方法については、『*OpenText™ OpenText™ Fortify Static Code Analyzer アプリケーションおよびツールガイド*』を参照してください。

アプリケーション	説明
Fortify Audit Workbench	開発者がセキュリティ上の欠陥を迅速に修正できるように分析結果を整理、調査、および優先順位付けするのに役立つ、Fortify Static Code Analyzerのグラフィカルユーザインタフェースを提供するアプリケーション。
OpenText™ Fortify Plugin for Eclipse	プロジェクトのコードベース全体をスキャンして分析し、Eclipse IDEからJavaコードの脆弱性を特定するソフトウェアセキュリティルールを適用する機能を追加します。結果とともに、個々のセキュリティ問題の説明とそれらの解消に関する提案が表示されます。
OpenText™ Fortify Analysis Plugin for IntelliJ IDEA and Android Studio	プロジェクトのコードベース全体でFortify Static Code Analyzerスキャンを実行し、IntelliJ IDEAおよびAndroid Studioからコード内の脆弱性を識別するソフトウェアセキュリティルールを適用する機能を追加します。
OpenText™ Fortify Extension for Visual Studio	ソリューションおよびプロジェクトのセキュリティ脆弱性をスキャンして見つけ、Visual Studioにスキャン結果を表示する機能を追加します。結果には、見つかった問題のリスト、各問題が表す脆弱性のタイプの説明、それらの修正方法に関する提案が含まれます。この拡張機能には、Fortify Software Security Centerサーバに保存された監査結果を使用する修正機能も含まれています。
OpenText™ Fortify Custom Rules Editor	カスタムルールを作成および編集するためのアプリケーション。

アプリケーション	説明
Fortify Scan Wizard	Fortify Static Code Analyzerを(ローカルまたはFortify ScanCentral SASTを使用してリモートで)使用してコードをスキャンするスクリプトを準備し、必要に応じて結果をFortify Software Security Centerにアップロードできるグラフィカルユーザインタフェースを提供します。
BIRTReportGenerator ReportGenerator	FPRファイルから問題レポート(BIRT)およびレガシレポートを生成するコマンドラインツール

サンプルプロジェクト

OpenTextでは、個別にダウンロードできるサンプルプロジェクトが`Fortify_SCA_Samples_<version>.zip`アーカイブで提供されています。

このZIPファイルには、`basic`ディレクトリと`advanced`ディレクトリの2つが含まれています。各コードサンプルには、Fortify Static Code Analyzerでコードをスキャンしてその結果をFortify Audit Workbenchに表示する方法について説明する`README.txt`ファイルが含まれています。

`basic`ディレクトリには、簡単な言語固有のコードサンプルのセットが含まれています。`advanced`ディレクトリには、より高度なサンプルが含まれています。

関連ドキュメント

このトピックでは、Fortifyソフトウェア製品に関する情報を提供するドキュメントについて説明します。

注: Fortifyの製品ドキュメントは、https://www.microfocus.com/support/documentationにあります。ほとんどのガイドは、PDF形式とHTML形式の両方で提供されています。

すべての製品

以下のドキュメントには、すべての製品に関する一般情報が記載されています。特に明記されている場合を除き、これらのドキュメントは製品マニュアルのWebサイトで利用できます。

ドキュメント/ファイル名	説明
Fortifyソフトウェアのマニュアルについて About_Fortify_Docs_<*version*>.pdf	この文書では、Fortify製品のドキュメントにアクセスする方法について説明します。 **注:** このドキュメントは、製品のダウンロードにのみ含まれています。
Fortifyソフトウェアシステム要件 Fortify_Sys_Reqs_<*version*>.pdf	このドキュメントでは、Fortifyソフトウェアのこのバージョンでサポートされている環境と製品について詳しく説明します。
Fortifyソフトウェアリリースノート FortifySW_RN_<*version*>.pdf	このドキュメントでは、Fortifyソフトウェアのこのリリースで行われた変更の概要と、他の製品ドキュメントには記載されていない重要な情報について説明します。
Fortifyソフトウェア<*version*>の新機能 Fortify_Whats_New_<*version*>.pdf	このドキュメントでは、Fortifyソフトウェア製品の新機能について説明します。

Fortify ScanCentral SAST

次のドキュメントでは、Fortify ScanCentral SASTの情報について説明します。特に明記されている場合を除き、このドキュメントは製品マニュアルのWebサイト(https://www.microfocus.com/documentation/fortify-software-security-center)で利用できます。

ドキュメント/ファイル名	説明
OpenText™ Fortify ScanCentral SASTインストール、設定、および使用ガイド SC_SAST_Guide_<*version*>.pdf	このドキュメントでは、静的コード分析プロセスを合理化するためにFortify ScanCentral SASTをインストール、設定、および使用する方法について説明します。これは、リソースを大量に消費するFortify Static Code Analyzerプロセスの変換およびスキャンフェーズをオフロードするためにFortify ScanCentral SASTをインストール、設定、または使用するユーザを対象にしています。

Fortify Software Security Center

以下のドキュメントには、Fortify Software Security Centerに関する情報が記載されています。特に明記されている場合を除き、このドキュメントは製品マニュアルのWebサイト(https://www.microfocus.com/documentation/fortify-software-security-center)で利用できます。

ドキュメント/ファイル名	説明
OpenText™ Fortify Software Security Centerユーザガイド SSC_Guide_<*version*>.pdf	このドキュメントでは、Fortify Software Security Centerをデプロイして使用する方法に関する詳細情報をFortify Software Security Centerユーザに提供します。Fortify Software Security Centerを取得、インストール、設定、および使用するために必要なすべての情報を提供します。 これは、システムおよびインスタンス管理者、データベース管理者(DBA)、エンタープライズセキュリティリード、開発チームマネージャ、および開発者による使用を目的としています。Fortify Software Security Centerは、セキュリティチームのリードに、プロジェクトの履歴と現在のステータスの概要を提供します。

Fortify Static Code Analyzer

以下のドキュメントには、Fortify Static Code Analyzerに関する情報が記載されています。特に明記されている場合を除き、これらのドキュメントは製品マニュアルのWebサイト(https://www.microfocus.com/documentation/fortify-static-code)で利用できます。

ドキュメント/ファイル名	説明
OpenText™ Fortify Static Code Analyzerユーザガイド SCA_Guide_<*version*>.pdf	このドキュメントでは、多くの主要なプログラミングプラットフォームにFortify Static Code Analyzerをインストールし、コードをスキャンするために使用する方法について説明します。これは、セキュリティ監査とセキュアコーディングを担当するユーザを対象にしています。
OpenText™ OpenText™ Fortify Static Code Analyzerアプリケーションおよびツールガイド SCA_Apps_Tools_<*version*>.pdf	このドキュメントでは、Fortify Static Code Analyzerのアプリケーションとツールのインストール方法について説明します。Fortify Static Code Analyzerを使用してコードのスキャン、分析結果の確認、分析結果ファイルの操作などを行うことのできるアプリケーションとコマンドラインツールの概要を提供します。

ドキュメント/ファイル名	説明
OpenText™ Fortify Static Code Analyzerカスタムルールガイド SCA_Cust_Rules_Guide_<*version*>.zip	このドキュメントでは、Fortify Static Code Analyzerのカスタムルールを作成するために必要な情報について説明します。このガイドには、ルール作成の概念を実際のセキュリティ問題に適用する例が含まれています。 **注:** このドキュメントは、製品のダウンロードにのみ含まれています。
OpenText™ Fortify License and Infrastructure Managerインストールおよび使用ガイド LIM_Guide_<*version*>.pdf	このドキュメントでは、Fortify License and Infrastructure Manager (LIM)をインストール、設定、使用する方法について説明します。LIMは、ローカルWindowsサーバにインストールして、Dockerプラットフォーム上のコンテナイメージとして使用できます。

Fortify Static Code Analyzerアプリケーションとツール

次のドキュメントには、Fortify Static Code Analyzerのアプリケーションとツールに関する情報が記載されています。特に明記されている場合を除き、これらのドキュメントは製品マニュアルのWebサイト(https://www.microfocus.com/documentation/fortify-static-code-analyzer-and-tools)で利用できます。

ドキュメント/ファイル名	説明
OpenText™ Fortify Audit Workbenchユーザガイド AWB_Guide_<*version*>.pdf	このドキュメントでは、Fortify Audit Workbenchを使用して、ソフトウェアプロジェクトをスキャンして分析結果を監査する方法について説明します。このガイドには、バグトラッカとの統合方法、レポートの作成方法、共同監査の実行方法も記載されています。
OpenText™ Fortify Plugin for Eclipseユーザガイド Eclipse_Plugin_Guide_<*version*>.pdf	このドキュメントでは、Fortify Eclipse用Completeプラグインをインストールして使用する方法について説明します。
OpenText™ Fortify Analysis Plugin for IntelliJ IDEA and Android Studioユーザガイド IntelliJ_AnalysisPlugin_Guide_<*version*>.pdf	このドキュメントでは、Fortify IntelliJ IDEAおよびAndroid Studio用分析プラグインをインストールして使用する方法について説明します。

ドキュメント/ファイル名	説明
OpenText™ Fortify Extension for Visual Studioユーザガイド VS_Ext_Guide_<*version*>.pdf	このドキュメントでは、Fortify Extension for Visual Studioをインストールおよび使用して、コードを分析、監査、修復し、ソリューションとプロジェクトのセキュリティに関する問題を解決する方法について説明します。

第2章: Fortify Static Code Analyzerのインストール

この章では、Fortify Static Code Analyzerをインストールおよびアンインストールする方法について説明します。この章では、インストール後の基本的なタスクについても説明します。ドキュメント『*Fortifyソフトウェアシステム要件*』を参照して、各ソフトウェアコンポーネントのインストールに関する最小要件をシステムが満たしていることを確認してください。

このセクションでは、次のトピックについて説明します。

Fortify Static Code Analyzerのインストールについて

このセクションでは、Fortify Static Code Analyzerをインストールする方法について説明します。いくつかのコマンドラインツールがFortify Static Code Analyzerと一緒に自動的にインストールされます(「"コマンドラインツール" ページ151」を参照)。オプションで、Fortify ScanCentral SASTクライアントとFortify Software Security Center fortifyclientユーティリティをFortify Static Code Analyzerのインストールに含めることができます。Fortify ScanCentral SASTの詳細については、『*OpenText™ Fortify ScanCentral SASTインストール、設定、および使用ガイド*』を参照してください。

Fortify Static Code Analyzerインストール用のFortifyライセンスファイルとオプションでLIMライセンスプール資格情報を指定する必要があります。次の表には、Fortify Static Code Analyzerをインストールする方法を示します。

インストール方法	指示
標準インストールウィザードを使用してインストールを実行する	"Fortify Static Code Analyzerのインストール" 次のページ
サイレント(無人)でインストールを実行する	"Fortify Static Code Analyzerのサイレント(無人)インストール" ページ31
Windows以外のシステムでテキストベースのインストールを実行する	"Windows以外のプラットフォームでのテキストベースモードでのFortify Static Code Analyzerのインストール" ページ33

インストール方法	指示
Dockerを使用してインストールを実行する	"Dockerを使用したFortify Static Code Analyzerのインストールと実行" ページ34

パフォーマンスを最適にするには、スキャンするコードが存在するローカルファイルシステムにFortify Static Code Analyzerをインストールします。

注: Windows以外のシステムでは、書き込み権限があるホームディレクトリを持っているユーザとしてFortify Static Code Analyzerをインストールする必要があります。ホームディレクトリを持ってないルート以外のユーザとしてFortify Static Code Analyzerをインストールしないでください。

インストールが完了したら、「"インストール後のタスク" ページ39」を参照して、システムのセットアップを完了するために実行できる追加のステップを確認します。また、インストールされた環境設定ファイルを更新して、ランタイム分析、出力、およびFortify Static Code Analyzerのパフォーマンスを設定することもできます。Fortify Static Code Analyzerの環境設定オプションについては、「"環境設定オプション" ページ185」を参照してください。

Fortify Static Code Analyzerのインストール

Fortify Static Code Analyzerをインストールするには:

1. オペレーティングシステムのインストーラファイルを実行して、Fortify Static Code Analyzerセットアップウィザードを起動します。
   - Windows: `Fortify_SCA_<version>_windows_x64.exe`
   - Linux: `Fortify_SCA_<version>_linux_x64.run`
   - macOS: `Fortify_SCA_<version>_osx_x64.app.zip`
     APPインストーラファイルを実行する前に、ZIPファイルを展開します。
   - AIX: `Fortify_SCA_<version>_aix.run`

   ここで、*<version>*はソフトウェアリリースのバージョンです。次に、 **次へ(Next)**]をクリックします。
2. ライセンス契約を確認して受諾し、 **次へ(Next)**]をクリックします。
3. (オプション)インストールするコンポーネントを選択して、 **次へ(Next)**]をクリックします。
4. 一部のタイプのプロジェクト分析に必要な最小限のソフトウェアがシステムに含まれていないことがインストーラによって検出された場合、不足している要件と、その要件を必要とするプロジェクトがシステム要件のページに表示されます。 **次へ(Next)**]をクリックします。
   すべてのソフトウェア要件については、『*Fortifyソフトウェアシステム要件*』ドキュメントを参照してください。
5. Fortify Static Code Analyzerのインストール先を選択し、 **次へ(Next)**]をクリックします。
   手順3で、Fortify ScanCentral SASTクライアントをインストールに含めることを選択した場合は、パスにスペースを含まない場所を指定する必要があります。

重要 Fortifyアプリケーションとツールのインストール先と同じディレクトリにFortify Static Code Analyzerをインストールしないでください。

6. `fortify.license`ファイルへのパスを指定し、 **次へ(Next)**]をクリックします。
7. (オプション)同時ライセンスの管理にFortify License and Infrastructure Manager (LIM)を使用するには、 [LIMライセンス(LIM License)]ページで、[**はい**]を選択してから、 **次へ**]をクリックします。

注: Fortify Static Code Analyzerでライセンスが必要なタスクが実行されたら、Fortify Static Code Analyzerでライセンスプールからのリースの取得が試みられます。LIMサーバとの通信上の問題があるためにFortify Static Code Analyzerでライセンスの取得に失敗した場合は、Fortifyのライセンスファイルが使用されます。この動作を変更するには、`fortify-sca.properties`ファイル内で`com.fortify.sca.lim.WaitForInitialLicense`を使用します("LIMライセンスのプロパティ" ページ195」を参照)。

   a. LIM APIのURL、ライセンスプール名、およびプールパスワードを入力します。
   b. **次へ(Next)**]をクリックします。 [**LIM Proxy Settings**]ページが開きます。
   c. LIMサーバへの接続にプロキシサーバが必要な場合は、プロキシホスト(プロキシサーバのホスト名またはIPアドレス)とポート番号(オプション)を入力します。
   d. [**Next**]をクリックします。
8. インストールのセキュリティコンテンツを更新するには

注: インストール時にインターネットにアクセスできない展開環境の場合、fortifyupdateコマンドラインツールを使用してセキュリティコンテンツを更新できます。「"Fortify Software Security Contentの手動インストール" ページ34」を参照してください。

   a. 更新サーバのURLを入力します。セキュリティコンテンツの更新にFortifyルールパック更新サーバを使用するには、URLを`https://update.fortify.com`のままにします。Fortify Software Security Centerをアップデートサーバとして使用することもできます。
   b. (オプション)更新サーバへの接続にプロキシサーバが必要な場合は、プロキシホストとポート番号を入力します。
   c. セキュリティコンテンツを手動で更新する場合は、 [**インストール後にセキュリティコンテンツを更新する(Update security content after installation)**]チェックボックスをオフにします。
   d. **次へ(Next)**]をクリックします。
9. システムで以前のFortify Static Code Analyzerのインストールから移行するかどうかを指定します。

   以前のFortify Static Code Analyzerのインストールから移行すると、Fortify Static Code Analyzerアーティファクトファイルが保持されます。詳細については、「"Fortify Static Code Analyzerのアップグレードについて" ページ36」を参照してください。

注: `scapostinstall`コマンドラインツールを使用してFortify Static Code Analyzerアーティファクトを移行することもできます。インストール後のツールを使用して以前のFortify Static Code Analyzerのインストールから移行する方法については、「"プロパティファイルの移行" ページ39」を参照してください。

以前のインストールからアーティファクトを移行するには、次の手順に従います。

a. **[Static Code Analyzerの移行(Static Code Analyzer Migration)]**ページで **[はい(Yes)]**を選択して、 **[次へ(Next)]**をクリックします。

b. システム上の既存のFortify Static Code Analyzerのインストール場所を指定し、 **[次へ(Next)]**をクリックします。

以前のリリースからのアーティファクトのマイグレーションをスキップするには、 **[Static Code Analyzerの移行(Static Code Analyzer Migration)]**の選択を **[いいえ(No)]**のままにして、 **[次へ(Next)]**をクリックします。

10. **[インストール準備(Ready to Install)]**ページで **[次へ(Next)]**をクリックして、Fortify Static Code Analyzer、選択したコンポーネント、およびFortify Security Contentをインストールします。

セキュリティコンテンツの更新を選択した場合は、 **[セキュリティコンテンツ更新の結果(Security Content Update Result)]**ウィンドウにセキュリティコンテンツ更新の結果が表示されます。

11. **[完了(Finish)]**をクリックして、Fortify Static Code Analyzerセットアップウィザードを閉じます。

Fortify Static Code Analyzerのサイレント(無人)インストール

サイレントインストールを使用すると、ユーザプロンプトを表示せずにインストールを完了できます。サイレントでインストールするには、インストーラに必要な情報を提供するオプションファイルを作成する必要があります。サイレントインストールを使用して、複数のコンピュータにインストールパラメータを複製できます。

重要 Fortifyアプリケーションとツールのインストール先と同じディレクトリにFortify Static Code Analyzerをインストールしないでください。

Fortify Static Code Analyzerをサイレントでインストールすると、デフォルトでは、インストーラによってFortify Software Security Centerがダウンロードされません。オプションファイルでFortify Security Contentのダウンロードを有効にするか、Fortify Security Contentを手動でインストールすることができます("Fortify Software Security Contentの手動インストール" ページ34を参照)。

Fortify Static Code Analyzerをサイレントでインストールするには:

1. オプションファイルを作成します。
   a. 次の行を含むテキストファイルを作成します。

```
fortify_license_path=<license_file_location>
```

   ここで、*<license_file_location>*は`fortify.license`ファイルへのフルパスです。

   b. LIMライセンスサーバを使用するには、LIMライセンスプール資格情報を含む次の行をオプションファイルに追加します。

```
lim_url=<lim_url> lim_pool_name=<license_pool_name> lim_pool_
password=<license_pool_pwd>
```

c. `https://update.fortify.com`のデフォルトとは異なる場所をFortify Security Contentの更新に使用している場合は、次の行を追加します。

```
update_server=<update_server_url>
```

d. Fortify Security Contentのダウンロードにプロキシサーバが必要な場合は、次の行を追加します。

```
update_proxy_server=<proxy_server> update_proxy_port=<port_number>
```

e. Fortify Security Contentのダウンロードを有効にするには、次の行を追加します。

```
update_security_content=1
```

f. 必要に応じて、オプションファイルにインストール指示を追加します。
オプションファイルに追加できるインストールオプションのリストを取得するには、コマンドプロンプトを開き、インストーラファイル名と`--help`オプションを入力します。このコマンドでは、使用可能な各コマンドラインオプションの前に二重ダッシュが付けられ、使用可能なパラメータが角括弧で囲まれます。たとえば、インストールの進行状況をコマンドラインに表示する場合は、オプションファイルに`unattendedmodeui=minimal`を追加します。

メモ

- コマンドラインオプションでは大文字と小文字が区別されます。
- インストールオプションは、サポートされているすべてオペレーティングシステムで同じではあるとは限りません。`--help`を使用してインストーラを実行して、オペレーティングシステムで使用可能なオプションを確認します。

次のWindowsオプションファイルの例では、ライセンスファイルの場所、Fortify Software Security Centerサーバの場所とFortify Security Contentを取得するためのプロキシ情報、以前のリリースからの移行要求、およびFortify Static Code Analyzerのインストールディレクトリの場所が指定されています。

```
fortify_license_path=C:\Users\admin\Desktop\fortify.license update_
server=https://my_ssc_host:8080/ssc
update_proxy_server=webproxy.abc.company.com
update_proxy_port=8080
migrate_sca=1
install_dir=C:\Fortify
```

LinuxおよびmacOSのオプションファイルの例は、次のとおりです。

```
fortify_license_path=/opt/Fortify/fortify.license update_
server=https://my_ssc_host:8080/ssc update_proxy_
server=webproxy.abc.company.com update_proxy_port=8080 migrate_sca=1
install_dir=/opt/Fortify
```

2. オプションファイルを保存します。

3. オペレーティングシステムのサイレントインストールコマンドを実行します。

注: 場合によっては、インストーラを実行する前に、管理者としてコマンドプロンプトを実行する必要があります。

Windows	`Fortify_SCA_<version>_windows_x64.exe--mode unattended--optionfile <full_path_to_options_file>`
Linux	`./Fortify_SCA_<version>_linux_x64.run --mode unattended--optionfile <full_path_to_options_file>`
macOS	コマンドを実行する前に、ZIPファイルを展開する必要があります。 `Fortify_SCA_<version>_osx_x64.app/Contents/MacOS/installbuilder.sh --mode unattended --optionfile <full_path_to_options_file>`
AIX	`./Fortify_SCA_<version>_aix.run --mode unattended --optionfile <full_path_to_options_file>`

インストールが完了すると、インストーラのログファイルが作成されます。このログファイルは、オペレーティングシステムに応じて次の場所に保存されます。

Windows	`C:\Users\<username>\AppData\Local\Temp\FortifySCA-<version>-install.log`
Windows以外	`/tmp/FortifySCA-<version>-install.log`

Windows以外のプラットフォームでのテキストベースモードでのFortify Static Code Analyzerのインストール

コマンドラインでテキストベースのインストールを実行します。インストール時に、インストールを完了するために必要な情報の入力を求めるプロンプトが表示されます。Windowsシステムではテキストベースのインストールがサポートされていません。

重要 Fortifyアプリケーションとツールのインストール先と同じディレクトリにFortify Static Code Analyzerをインストールしないでください。

Fortify Static Code Analyzerのテキストベースのインストールを実行するには、次の表に示すように、オペレーティングシステム用のテキストベースのインストールコマンドを実行します。

Linux	`./Fortify_SCA_<version>_linux_x64.run --mode text`

macOS	コマンドを実行する前に、提供されたZIPファイルを展開する必要があります。 `Fortify_SCA_<version>_osx_x64.app/Contents/MacOS/installbuilder.sh --mode text`
AIX	`./Fortify_SCA_<version>_aix.run --mode text`

Fortify Software Security Contentの手動インストール

Fortify Software Security Content (Fortify Secure Coding Rulepacksとメタデータ)は、インストール時に自動的にインストールできます。ただし、Fortifyルールパック更新サーバからFortify Software Security Contentをダウンロードしてから、fortifyupdateコマンドラインツールを使用してインストールすることもできます。このオプションは、インストール時にインターネットにアクセスできない展開環境のために提供されています。

リモートサーバまたはローカルにダウンロードされたファイルからFortify Security Contentをインストールするには、fortifyupdateを使用します。

セキュリティコンテンツをインストールするには、次の手順に従います。

1. コマンドウィンドウを開きます。
2. `<sca_install_dir>/bin`ディレクトリに移動します。
3. コマンドプロンプトで「`fortifyupdate`」と入力します。
   以前にFortifyルールパック更新サーバからFortify Software Security Contentをダウンロードした場合は、`-import`オプションとZIPファイルをダウンロードしたディレクトリへのパスを使用して`fortifyupdate`を実行します。

この同じツールを使用して、Fortify Software Security Contentを更新することもできます。fortifyupdateコマンドラインツールの詳細については、「"セキュリティコンテンツの更新" ページ152」を参照してください。

Dockerを使用したFortify Static Code Analyzerのインストールと実行

DockerイメージにFortify Static Code Analyzerをインストールしてから、Fortify Static Code AnalyzerをDockerコンテナとして実行できます。

注: Fortify Static Code Analyzerは、Dockerでサポートされているプラットフォームでのみ実行できます。

Fortify Static Code AnalyzerをインストールするDockerfileの作成

このトピックでは、DockerイメージでFortify Static Code AnalyzerをインストールするDockerfileを作成する方法について説明します。

Dockerfileには、次の命令が含まれている必要があります。

1. ベースイメージに使用されるLinuxシステムを設定します。

   注: Fortify Static Code Analyzerの実行時にビルドツールを使用する場合は、イメージに必要なビルドツールがインストールされていることを確認します。サポートされているビルドツールの使用については、「"ビルド統合" ページ124」を参照してください。

2. COPY命令を使用して、Fortify Static Code Analyzerインストーラ、Fortifyライセンスファイル、およびインストールオプションファイルをDockerイメージにコピーします。

   インストールオプションファイルの作成方法については、"Fortify Static Code Analyzerのサイレント(無人)インストール" ページ31を参照してください。

3. RUN命令を使用してFortify Static Code Analyzerインストーラを実行します。

   インストーラは無人モードで実行する必要があります。詳細については、「"Fortify Static Code Analyzerのサイレント(無人)インストール" ページ31」を参照してください。

4. RUN命令を使用して、runifyupdateを実行してFortify Security Contentをインストールします。

   重要 Fortify Static Code Analyzerでプロジェクトの分析を実行するには、Fortify Security Contentのインストールが必要です。次の例では、イメージの構築中にダウンロードしたローカルファイルから、Fortify Security Contentをインストールします。updateifyupdateツールを使用したFortify Security Contentのダウンロードとインストールの詳細については、「"Fortify Software Security Contentの手動インストール" 前のページ」を参照してください。

5. Fortify Static Code Analyzerを実行できるようにイメージを設定するには、ENTRYPOINT命令を使用して、インストールされたsourceanalyzer実行可能ファイルの場所にエントリポイントを設定します。

   sourceanalyzerのデフォルトのインストールパスは`/opt/Fortify/Fortify_SCA_<version>/bin/sourceanalyzer`です。

Fortify Static Code AnalyzerをインストールするDockerfileの例は、次のとおりです。

```
FROM ubuntu:18.04 WORKDIR /app ENV APP_HOME="/app" ENV RULEPACK="MyRulepack.zip" COPY
fortify.license ${APP_HOME} COPY Fortify_SCA_24.2.0_linux_x64.run ${APP_HOME} COPY optionFile
${APP_HOME} COPY ${RULEPACK} ${APP_HOME} RUN ./Fortify_SCA_24.2.0_linux_x64.run --mode unattended \
--optionfile "${APP_HOME}/optionFile" && \ /opt/Fortify/Fortify_SCA_24.2.0/bin/fortifyupdate -
import ${RULEPACK} && \ rm Fortify_SCA_24.2.0_linux_x64.run optionFile ENTRYPOINT
["/opt/Fortify/Fortify_SCA_24.2.0/bin/sourceanalyzer"]
```

現在のディレクトリからDockerfileを使用してdockerイメージを作成するには、docker buildコマンドを使用する必要があります。例:

```
docker buildx build -f <docker_file> -t <image_name> "."
```

コンテナの実行

このトピックでは、Fortify Static Code Analyzerイメージをコンテナとして実行する方法について説明し、変換およびスキャン用のDocker実行コマンドの例を示します。

注: コンテナでFortify Static Code Analyzerを実行する際に、特にランタイムコンテナ保護も利用する場合は、Fortify Static Code Analyzerにビルドコマンド(javacなど)を実行するための適切な許可があることを確認します。

Fortify Static Code Analyzerイメージをコンテナとして実行するには、ホストファイルシステムからコンテナに次の2つのディレクトリをマウントする必要があります。

- 分析するソースファイルが含まれるディレクトリ。
- 変換フェーズとスキャンフェーズの間にFortify Static Code Analyzerビルドセッションを保存し、出力ファイル(ログとFPRファイル)をホストと共有するための一時ディレクトリ。

  このディレクトリは、Fortify Static Code Analyzerの変換コマンドとスキャンコマンドの両方で`-project-root`コマンドラインオプションを使用して指定します。

次のコマンドの例では、`/src`に入力ディレクトリ`/sources`をマウントし、`/scratch_docker`に一時ディレクトリをマウントします。この例のイメージ名は`fortify-sca`です。

変換およびスキャン用のDocker実行コマンドの例

次の例では、一時ディレクトリとソースディレクトリをマウントし、変換フェーズ用にコンテナからFortify Static Code Analyzerを実行します。

```
docker run -v /scratch_local/:/scratch_docker -v /sources/:/src -it
fortify-sca –b MyProject -project-root /scratch_docker [<sca_options>] /src
```

次の例では、一時ディレクトリをマウントし、分析フェーズ用にコンテナからFortify Static Code Analyzerを実行します。

```
docker run -v /scratch_local/:/scratch_docker -it fortify-sca –b MyProject
-project-root /scratch_docker –scan [<sca_options>] –f /scratch_
docker/MyResults.fpr
```

`MyResults.fpr`出力ファイルがホストの`/scratch_local`ディレクトリに作成されます。

Fortify Static Code Analyzerのアップグレードについて

Fortify Static Code Analyzerをアップグレードするには、現在のバージョンがインストールされている場所とは異なる場所に新しいバージョンをインストールし、以前のインストールから設定を移行します。この移行では、*<sca_install_dir>*`/Core/config`ディレクトリにあるFortify Static Code Analyzerアーティファクトファイルが保持および更新されます。

以前のリリースから設定を移行しない場合は、次のデータが変更されていれば、バックアップを保存することが推奨されています。

- *<sca_install_dir>*`/Core/config/rules`フォルダ
- *<sca_install_dir>*`/Core/config/customrules`フォルダ
- *<sca_install_dir>*`/Core/config/ExternalMetadata`フォルダ

- *<sca_install_dir>*/Core/config/CustomExternalMetadataフォルダ
- *<sca_install_dir>*/Core/config/server.propertiesファイル
- *<sca_install_dir>*/Core/config/scalesフォルダ

新しいバージョンをインストールしたら、以前のバージョンをアンインストールできます。詳細については、「"Fortify Static Code Analyzerのアンインストールについて" 下」を参照してください。

注: 以前のバージョンがインストールされたままにしておくこともできます。同じシステムに複数のバージョンがインストールされている場合は、コマンドラインからコマンドを実行すると、最近インストールされたバージョンが使用されます。

Fortify Static Code Analyzerのアンインストールについて

このセクションでは、Fortify Static Code Analyzerをアンインストールする方法について説明します。標準インストールウィザードを使用することも、アンインストールをサイレントで実行することもできます。Windows以外のシステムでは、テキストベースのアンインストールを実行することもできます。

Fortify Static Code Analyzerのアンインストール

Fortify Static Code Analyzerをアンインストールするには:

1. ご使用のオペレーティングシステムの*<sca_install_dir>*にあるアンインストールコマンドを実行します。

OS	アンインストールコマンド
Windows	`Uninstall_FortifySCA.exe` または、次の操作を実行できます。 a. [スタート] > [設定] > [アプリ] > [アプリと機能(Apps & features)] の順に選択します。 b. プログラムのリストから、 **[Fortify Static Code Analyzer *<version>*]** を選択し、 [アンインストール] をクリックします。
Linux AIX	`./Uninstall_FortifySCA_`
macOS	`Uninstall_FortifySCA_.app`

2. アプリケーション全体を削除するか、個々のコンポーネントを削除するかを指定するよう求めるメッセージが表示されます。どちらかを選択し、 **[次へ(Next)]** をクリックします。
   特定のコンポーネントをアンインストールする場合は、 [アンインストールするコンポーネントの選択(Select Components to Uninstall)] ページで削除するコンポーネントを選択し、 **[次へ(Next)]** をクリックします。

3. すべてのアプリケーション設定を削除するかどうかを指定するよう求めるメッセージが表示されます。次のいずれかを実行します。
   - アンインストールするFortify Static Code Analyzerのバージョンと一緒にインストールされたコンポーネントのアプリケーション設定を削除するには、**[はい(Yes)]**をクリックします。

     Fortify Static Code Analyzer (`sca<version>`)フォルダは削除されません。
   - **[No]**をクリックして、アプリケーション設定をシステムに保持します。

Fortify Static Code Analyzerのサイレントアンインストール

Fortify Static Code Analyzerをサイレントでアンインストールするには、次の手順に従います。

1. インストールディレクトリに移動します。
2. オペレーティングシステムに基づいて、次のコマンドのいずれかを入力します。

Windows	`Uninstall_FortifySCA_<version>.exe --mode unattended`
Linux **AIX**	`./Uninstall_FortifySCA_<version> --mode unattended`

macOS	`Uninstall_FortifySCA_`
	`<version>.app/Contents/MacOS/installbuilder.sh`
	`--mode unattended`

注: Windows、Linux、およびmacOSの場合、アンインストールするFortify Static Code Analyzerのバージョンと一緒にインストールされたコンポーネントのアプリケーション設定はアンインストーラによって削除されます。

Windows以外のプラットフォームでのテキストベースモードでのFortify Static Code Analyzerのアンインストール

Fortify Static Code Analyzerをテキストベースモードでアンインストールするには、次のように、オペレーティングシステム用のテキストベースのインストールコマンドを実行します。

1. インストールディレクトリに移動します。
2. オペレーティングシステムに基づいて、次のコマンドのいずれかを入力します。

Linux **AIX**	`./Uninstall_FortifySCA_<version> --mode text`
macOS	`Uninstall_FortifySCA_` `<version>.app/Contents/MacOS/installbuilder.sh --mode text`

インストール後のタスク

インストール後のタスクでは、Fortify Static Code Analyzerの使用を開始する準備をします。

インストール後処理ツールの実行

インストール後処理ツールを使用すると、Fortify Static Code Analyzerの以前のバージョンのプロパティファイルの移行、Fortify Security Contentの更新設定、およびFortify Software Security Centerへの接続設定を行うことができます。

Fortify Static Code Analyzerインストール後処理ツールを実行するには

1. コマンドラインから*<sca_install_dir>*/binディレクトリに移動します。
2. コマンドプロンプトで「scapostinstall」と入力します。
3. 次のいずれかを入力します。
   - 設定を表示するには、「s」を入力します。
   - 前のプロンプトに戻るには、「r」を入力します。
   - ツールを終了するには、「q」を入力します。

プロパティファイルの移行

以前のバージョンのFortify Static Code Analyzerからシステムにインストールされている現在のバージョンのFortify Static Code Analyzerにプロパティファイルを移行するには、次の手順に従います。

1. コマンドラインから*<sca_install_dir>*/binディレクトリに移動します。
2. コマンドプロンプトで「scapostinstall」と入力します。
3. 「1」と入力して、Migrationを選択します。
4. 「1」と入力して、Static Code Analyzer Migrationを選択します。
5. 「1」と入力して、Migrate from an existing Fortify installationを選択します。
6. 「1」と入力して、Set previous Fortify installation directoryを選択します。
7. 以前のインストールディレクトリを入力します。
8. 「s」と入力して、設定を確認します。
9. 「2」と入力して、移行を実行します。
10. 「y」と入力して、確認します。

ロケールの指定

Fortify Static Code Analyzerインストールのデフォルトロケールは英語です。

Fortify Static Code Analyzerインストールのロケールを変更するには、次の手順を実行します。

1. コマンドラインからbinディレクトリに移動します。
2. コマンドプロンプトで「scapostinstall」と入力します。
3. 「2」と入力して、Settingsを選択します。
4. 「1」と入力して、Generalを選択します。
5. 「1」と入力して、Localeを選択します。
6. 次のいずれかのロケールコードを入力します。
   - en (英語)
   - es (スペイン語)
   - ja (日本語)
   - ko (韓国語)
   - pt_BR (ポルトガル語(ブラジル))
   - zh_CN (簡体字中国語)
   - zh_TW (繁体字中国語)

Fortify Security Contentの更新の設定

Fortify Security Contentを取得する方法を指定します。サーバに接続する必要がある場合は、プロキシ情報を指定する必要もあります。

Fortify Security Content更新の設定を指定するには、次の手順に従います。

1. コマンドラインからbinディレクトリに移動します。
2. コマンドプロンプトで「scapostinstall」と入力します。
3. 「2」と入力して、Settingsを選択します。
4. 「2」と入力して、Fortify Updateを選択します。
5. Fortifyルールパック更新サーバのURLを変更するには、「1」と入力してから、URLを入力します。
   Fortifyルールパック更新サーバのデフォルトのURLはhttps://update.fortify.comです。

6. Fortify Security Content更新用のプロキシを指定するには、次の手順に従います。
   a. 「2」と入力してProxy Serverを選択してから、プロキシサーバの名前を入力します。プロトコルとポート番号を除外します(例: some.secureproxy.com)。
   b. 「3」と入力してProxy Server Portを選択してから、プロキシサーバのポート番号を入力します。
   c. (オプション)プロキシサーバのユーザ名(オプション4)とパスワード(オプション5)を指定することもできます。

Fortify Software Security Centerへの接続の設定

Fortify Software Security Centerに接続する方法を指定します。ネットワークでFortify Software Security Centerへのアクセスにプロキシサーバが使用されている場合は、プロキシ情報を指定する必要があります。

Fortify Software Security Centerへの接続の設定を指定するには、次の手順に従います。

1. コマンドラインからbinディレクトリに移動します。
2. コマンドプロンプトで「scapostinstall」と入力します。
3. 「2」と入力して、Settingsを選択します。
4. 「3」と入力して、Software Security Center Settingsを選択します。
5. 「1」と入力してServer URLを選択してから、Fortify Software Security CenterサーバのURLを入力します。
6. 接続のプロキシ設定を指定するには、次の手順に従います。
   a. 「2」と入力してProxy Serverを選択してから、プロキシサーバの名前を入力します。プロトコルとポート番号を除外します(例: some.secureproxy.com)。
   b. 「3」と入力してProxy Server Portを選択してから、プロキシサーバのポート番号を入力します。
   c. プロキシサーバのユーザ名とパスワードを指定するには、ユーザ名のオプション4とパスワードのオプション5を使用します。
7. (オプション)次を指定することもできます。
   - Fortify Software Security CenterサーバからFortify Software Security Contentを更新するかどうか(オプション6)
   - Fortify Software Security Centerのユーザ名(オプション7)

プロキシサーバ設定の削除

以前にFortifyルールパック更新サーバまたはFortify Software Security Centerのプロキシサーバ設定を指定し、不要になった場合は、それらの設定を削除できます。

Fortify Software Security Contentアップデートを取得したり、Fortify Software Security Centerに接続したりするためのプロキシ設定を削除するには

1. コマンドラインからbinディレクトリに移動します。
2. コマンドプロンプトで「scapostinstall」と入力します。
3. 「2」と入力して、Settingsを選択します。
4. 「2」を入力してFortify Updateを選択するか、「Software Security Center Settings」を入力して3を選択します。
5. 削除するプロキシ設定に対応する番号を入力し、マイナス記号(-)を入力して設定を削除します。
6. 削除するプロキシ設定ごとに手順5を繰り返します。

信頼された証明書の追加

Fortify Static Code AnalyzerからFortifyの他のソフトウェア製品および外部システムに接続するために、HTTPS経由の通信が必要になる場合があります。次に例をいくつか示します。

- Fortify Static Code Analyzerのデフォルトでは、ライセンス管理にLIMサーバと通信するために、HTTPS接続が必要です。

  com.fortify.sca.lim.RequireTrustedSSLCertプロパティでは、LIMサーバとの接続に信頼されたSSL証明書が必要であるかどうかが決定されます。このプロパティの詳細については、「"LIMライセンスのプロパティ" ページ195」を参照してください。
- fortifyupdateコマンドラインツールでは、Windowsシステムのインストール時に自動的に、または手動で(「"Fortify Software Security Contentの手動インストール" ページ34」を参照)、HTTPS接続を使用してFortify Security Contentが更新されます。
- Fortify ScanCentral SASTセンサとして設定されたFortify Static Code Analyzerでは、HTTPS接続を使用してコントローラと通信します。

HTTPSを使用している場合、Fortify Static Code Analyzerとそのアプリケーションは提示されたSSLサーバ証明書に標準チェックをデフォルトで適用します。これには、証明書が信頼されているかどうかを確認するチェックが含まれます。組織が独自の認証局(CA)を運営し、そのCAから発行された証明書がサーバで提示される接続がFortify Static Code Analyzerで信頼される必要がある場合は、そのCAが信頼されるようにFortify Static Code Analyzerを設定する必要があります。さもないと、HTTPS接続の使用に失敗する可能性があります。

CAの信頼された証明書をFortify Static Code Analyzerキーストアに追加する必要があります。Fortify Static Code Analyzerキーストアは、*<sca_install_dir>*/jre/lib/security/cacertsファイルにあります。keytoolコマンドを使用すると、信頼された証明書をキーストアに追加できます。

信頼された証明書をFortify Static Code Analyzerキーストアに追加するには、次の手順に従います。

1. コマンドプロンプトを開き、次のコマンドを実行します。

   ```
   <sca_install_dir>/jre/bin/keytool -importcert -alias <alias_name> -
   cacerts -file <cert_file>
   ```

   ここで:

- *<alias_name>*は追加する証明書に固有の名前です。
- *<cert_file>*はPEM形式またはDER形式の信頼されたルート証明書を含むファイルの名前です。

2. キーストアパスワードを入力します。

   注: デフォルトのパスワードは`changeit`です。

3. この証明書を信頼するように求めるプロンプトが表示されたら、 [**yes**]を選択します。

第3章: 分析プロセスの概要

このセクションでは、次のトピックについて説明します。

分析プロセス

分析プロセスは、次の4つの個別のフェーズで構成されます。

1. **ビルド統合** - Fortify Static Code Analyzerをビルドツールに統合するかどうかを選択します。ビルド統合のオプションについては、「"ビルドへのFortify Static Code Analyzer統合" ページ124」を参照してください。
2. **変換** - 一連のコマンドを使用してソースコードを収集し、ビルドIDに関連付けられた中間形式に変換します。通常、ビルドIDは変換するプロジェクトの名前です。詳細については、「"変換フェーズ" 次のページ」を参照してください。
3. **分析** - 変換フェーズで特定されたソースファイルをスキャンし、分析結果ファイルを通常はFortify Project Results (FPR)形式で生成します。FPRファイルの拡張子は.fprです。詳細については、「"分析フェーズ" ページ48」を参照してください。
4. **変換と分析の検証** - ソースファイルが適切なルールパックを使用してスキャンされ、エラーがレポートされていないことを検証します。詳細については、「"変換フェーズと分析フェーズの検証" ページ51」を参照してください。

変換コマンドや分析コマンドは、最小限の特権アクセスを持つユーザアカウントから実行することをお勧めします。ルートユーザとしてFortify Static Code Analyzerを実行したり、ルートアクセスを必要とするプロジェクトを変換したりすると、正しく動作しない場合があるため、お勧めできません。

コードの変換と分析に使用するコマンドのシーケンスを以下に示します。

1. 指定したビルドIDの既存のFortify Static Code Analyzer一時ファイルをすべて削除します。

   ```
   sourceanalyzer -b MyProject -clean
   ```

   以前に使用したビルドIDを持つプロジェクトを分析するには、常に、このステップで分析を開始してください。
2. プロジェクトコードを変換します。

   ```
   sourceanalyzer -b MyProject <files_to_analyze>
   ```

ほとんどの言語では、このステップは同じビルドIDを持つsourceanalyzerへの複数の呼び出しで構成されることがあります。詳細については、「"変換フェーズ" 下」を参照してください。

3. プロジェクトコードを分析し、結果をFortify Project Results(FPR)ファイルに保存します。

```
sourceanalyzer -b MyProject -scan -f MyResults.fpr
```

詳細については、「"分析フェーズ" ページ48」を参照してください。

パラレル処理

大規模なプロジェクトのスキャン時間を短縮するために、Fortify Static Code Analyzerは並行分析モードで実行されます。これにより、システムで使用できるCPUコアをすべて活用できます。Fortify Static Code Analyzerを実行すると、スキャンに使用できるハードウェアの全リソースが使用されると想定されるため、Fortify Static Code Analyzerの実行時にCPUを大量に消費する他のプロセスは実行しないようにしてください。

変換フェーズ

通常どおりにコンパイルされているプロジェクトを正常に変換するには、プロジェクトをビルドするために必要な依存関係があることを確認します。特定の要件がある言語については、特定のソースコードタイプの章を参照してください。

分析プロセスの最初のステップ(ファイルの変換)を実行するための基本的なコマンドライン構文は、次のとおりです。

```
sourceanalyzer -b <build_id> ... <files>
```

または

```
sourceanalyzer -b <build_id> ... <compiler_command>
```

変換フェーズは、sourceanalyzerコマンドを使用した1回以上のFortify Static Code Analyzerの呼び出しで構成されます。Fortify Static Code Analyzerでは、ビルドID(-bオプション)を使用して呼び出しが相互に関連付けられます。後続のsourceanalyzerの呼び出しでは、ビルドIDに関連付けられたファイルリストに新しく指定されたソースファイルまたは環境設定ファイルが追加されます。

変換後に-show-build-warningsディレクティブを使用して、変換フェーズで発生した警告やエラーのリストを表示できます。

```
sourceanalyzer -b <build_id> -show-build-warnings
```

ビルドIDに関連付けられたファイルを表示するには、-show-filesディレクティブを使用します。

```
sourceanalyzer -b <build_id> -show-files
```

変換フェーズに関する特別な考慮事項

プロジェクトで変換フェーズを実行する前に、次の特別な考慮事項を検討してください。

- 動的言語(JavaScript/TypeScript、PHP、Python、Ruby)を変換する場合は、1回の呼び出しですべてのソースファイルを同時に指定する必要があります。Fortify Static Code Analyzerでは、後続の呼び出し時にビルドIDで関連付けられたファイルリストに新しいファイルを追加する機能はサポートされていません。
- 生成されるコードは、スクリプト、または解析ツールなどのツールによって、自動的に生成されます。このコードは最適化され、最小化されることもあれば、大きく複雑になることもあります。したがって、このコードでFortify Static Code Analyzerから報告される脆弱性をすべて修正するのは困難なことがあるため、変換から除外することをお勧めします。`-exclude`コマンドラインオプションを使用して、このタイプのコードを変換から除外してください。

次の章では、さまざまなソースコードタイプを変換する方法について説明します。

- "Javaコードの変換" ページ53
- "Kotlinコードの変換" ページ62
- "Visual Studioプロジェクトの変換" ページ66
- "CおよびC++コードの変換" ページ72
- "JavaScriptおよびTypeScriptコードの変換" ページ75
- "Pythonコードの変換" ページ80
- "モバイルプラットフォームのコードの変換" ページ85
- "Goコードの変換" ページ89
- "DartおよびFlutterコードの変換" ページ92
- "Rubyコードの変換" ページ94
- "COBOLコードの変換" ページ96
- "Salesforce ApexおよびVisualforceコードの変換" ページ101
- "その他の言語および設定の変換" ページ103

モバイルビルドセッション

Fortify Static Code Analyzerモバイルビルドセッション(MBS)を使用すると、あるコンピュータでプロジェクトを変換し、別のコンピュータでスキャンできます。モバイルビルドセッション(MBSファイル)には、分析フェーズに必要なファイルがすべて含まれています。スキャン時間を短縮するために、ビルドコンピュータで変換を実行してから、よりスキャンに適したコンピュータにビルドセッション(MBSファイル)を移動できます。開発者は自分のコンピュータで変換を実行し、1台の強力なコンピュータのみを使用して大規模なスキャンを実行できます。

プロジェクトに正規表現分析(「"正規表現分析" ページ49」を参照)を含めるには、コマンドに`-Dcom.fortify.sca.MobileBuildSessions=true`を含めてMBSファイルを作成し、ソースコードを

MBSに含めることをお勧めします。これにより、正規表現分析が別のコンピュータ上のスキャンでも機能するようになります。

変換を実行しているシステムと分析を実行しているシステムの両方に同じバージョンのFortify Security Content (Rulepack)がインストールされている必要があります。

モバイルビルドセッションバージョンの互換性

変換コンピュータ上のFortify Static Code Analyzerバージョンと分析コンピュータ上のFortify Static Code Analyzerバージョンに互換性がある必要があります。バージョン番号の形式は、*<major>.<minor>.<patch>.<build_number>* (24.2.0.0140など)です。変換コンピュータと分析コンピュータの両方でFortify Static Code Analyzerの*<major>*および*<minor>*部分が一致している必要があります。たとえば、24.2.0と24.2.xには互換性があります。Fortify Static Code Analyzerのバージョン番号を確認するには、コマンドラインで「`sourceanalyzer -v`」と入力します。

次のコマンドを使用してMBSファイルからビルドIDとFortify Static Code Analyzerバージョンを取得できます。

```
sourceanalyzer -import-build-session <file>.mbs -
Dcom.fortify.sca.ExtractMobileInfo=true
```

モバイルビルドセッションの作成

変換を実行したコンピュータで次のコマンドを発行して、モバイルビルドセッションを生成します。

```
sourceanalyzer -b <build_id> -export-build-session <file>.mbs
```

ここで、*<file>*.mbsはFortify Static Code Analyzerのモバイルビルドセッションに指定するファイル名です。

MBSファイルにソースコードを含めるには、次のコマンドを実行します。

```
sourceanalyzer -b <build_id> -Dcom.fortify.sca.MobileBuildSessions=true -
export-build-session <file>.mbs
```

モバイルビルドセッションのインポート

スキャンを実行するコンピュータに*<file>*.mbsファイルを移動したら、モバイルビルドセッションをFortify Static Code Analyzerのプロジェクトルートディレクトリにインポートします。

モバイルビルドセッションをインポートするには、次のコマンドを入力します。

```
sourceanalyzer -import-build-session <file>.mbs
```

Fortify Static Code Analyzerのモバイルビルドセッションをインポートしたら、分析フェーズに進むことができます。変換に使用されたときと同じビルドIDでスキャンを実行します。

複数のモバイルビルドセッションを単一のMBSファイルにマージすることはできません。エクスポートされたビルドセッションごとに、一意のビルドIDが必要です。ただし、同じFortify Static Code Analyzerのインストール時にすべてのビルドIDがインポートされたら、-bオプションを使用して1回のスキャンで複数のビルドIDをスキャンできます("分析フェーズ" 下」を参照)。

分析フェーズ

分析フェーズでは、変換時に作成された中間ファイルがスキャンされ、脆弱性結果ファイル(FPR)が作成されます。

分析フェーズは、sourceanalyzerの1回の呼び出しで構成されます。ビルドIDを指定し、-scanディレクティブを他の必要な分析オプションまたは出力オプションとともに含めます("分析オプション" ページ 138」および"出力オプション" ページ142」を参照)。

次の例は、分析フェーズを実行し、結果をFPRファイルに保存するコマンドライン構文を示しています。

```
sourceanalyzer -b MyProject -scan -f MyResults.fpr
```

注: デフォルトでは、Fortify Static Code AnalyzerのFPRファイルのソースコードが含まれています。

複数のビルドを単一のスキャンコマンドと組み合わせるには、コマンドラインに追加のビルドを追加します。

```
sourceanalyzer -b MyProject1 -b MyProject2 -b MyProject3 -scan -f
MyResults.fpr
```

分析へのスキャンポリシーの適用

分析(スキャン)フェーズでは、最も深刻な脆弱性を特定してコードを迅速に修復できるようにするために、スキャンポリシーを指定できます。次の表で、使用可能な3つのスキャンポリシーについて説明します。

ポリシー名	説明
security	これはデフォルトのスキャンポリシーで、コード品質に関連する問題は分析結果から除外されます。セキュリティ問題のコード修正に重点を置く場合はこのポリシーを使用します。
classic	このスキャンポリシーでは、問題は除外されません。このスキャンポリシーを使用すると、コード品質に関連する問題を含むすべての問題を確認できます。
devops	このスキャンポリシーでは、securityポリシーによっても除外される問題が除外されて、優先度の低い問題の報告数が減ります。このスキャンポリシーは、スキャン速度を優先し

ポリシー名	説明
	て、開発者が結果を(中間監査なしで)直接レビューする場合に使用します。このスキャンポリシーを適用した後に残る問題は、修正が必要な重大なセキュリティ問題である可能性があります。 **注:** ローカルのsecurityスキャンポリシーに加えたカスタマイズが、このdevopsスキャンポリシーに自動的に組み込まれることはありません。

分析用のスキャンポリシーを指定するには、次の例に示すように、分析フェーズに`-scan-policy`(または`-sc`)オプションを含める必要があります。

```
sourceanalyzer -b MyProject -scan -scan-policy devops -f MyResults.fpr
```

または、`fortify-sca.properties`ファイルの`com.fortify.sca.ScanPolicy`プロパティを使用してスキャンポリシーを指定することもできます。例:

```
com.fortify.sca.ScanPolicy=devops
```

注: 分析用のスキャンポリシー設定を使用して、フィルタファイルを適用できます(「"フィルタファイルによる問題の除外" ページ180」を参照)。この場合、Fortify Static Code Analyzerによってスキャンポリシーとフィルタファイルの両方が分析に適用されます。

ポリシーファイルは、*<sca_install_dir>*`/Core/config/scales`ディレクトリにあります。スキャンポリシーごとに1つのファイルがあります。これらのポリシーファイルの設定を変更して、スキャンポリシーをカスタマイズできます。ポリシーファイルに使用される構文については、「"フィルタファイルによる問題の除外" ページ180」を参照してください。

参照情報

"変換と分析フェーズのプロパティ" ページ187

正規表現分析

正規表現(regex)分析では、正規表現ルールを使用して、ファイルコンテンツとファイル名の両方の脆弱性を検出できます。この分析では、プロジェクトファイル内の脆弱なシークレット(パスワード、キー、資格情報など)を検出できます。Configuration Analyzerには、正規表現分析機能が含まれています。

重要 Regex分析は言語に依存しないため、Fortify Static Code Analyzerが公式にはサポートしていないファイルタイプの脆弱性を検出する可能性があります。

正規表現分析では、変換フェーズに含まれるすべてのファイルパスとパスパターンが再帰的に検査されます。特に変換から除外されていなければ、検出された各ディレクトリのファイルはすべて分析されます。正規表現分析に含まれているファイルを管理するには、次のオプションを使用します。

- 変換フェーズで`-exclude`オプションを使用してファイルまたはディレクトリを除外します。このオプションの詳細については、"変換オプション" ページ136を参照してください。
- デフォルトでは、正規表現分析から検出可能なバイナリファイルがすべて除外されます。分析にバイナリファイルを含めるには、次のプロパティを`fortify-sca.properties`ファイルに追加します(または、`-D`オプションを使用して、このプロパティをコマンドラインに含めます)。

```
com.fortify.sca.regex.ExcludeBinaries = false
```

- デフォルトでは、スキャン時間が許容可能になるように、10 MBを超えるファイルが正規表現分析から除外されます。次のプロパティを使用して、最大ファイルサイズ(メガバイト単位)を変更できます。

```
com.fortify.sca.regex.MaxSize = <max_file_size_mb>
```

正規表現分析を無効にするには、次のプロパティを`fortify-sca.properties`ファイルに追加するか、コマンドラインに含める必要があります。

```
com.fortify.sca.regex.Enable = false
```

参照情報

"モバイルビルドセッション" ページ46

"正規表現分析のプロパティ" ページ195

高次分析

高次分析(HOA)では、高次コードを使用してデータフローを追跡する機能が改善されます。高次コードでは、関数が値として操作され、匿名関数式(ラムダ式)を使用して関数が生成され、引数として渡され、値として返され、オブジェクトの変数およびフィールドに割り当てられます。これらのコードパターンは、JavaScript、TypeScript、Python、Ruby、Swiftなどの最新の動的言語で共通です。

デフォルトでは、JavaScript、TypeScript、Python、Ruby、Swiftの各コードをスキャンすると、Fortify Static Code Analyzerで高次分析が実行されます。高次分析のプロパティについては、「"変換と分析フェーズのプロパティ" ページ187」を参照してください。

モジュラー分析

このリリースには、モジュラー分析の技術プレビューが含まれています。モジュラー分析では、コアプロジェクトとは別にライブラリとサブライブラリを事前にスキャンできます。その後、コアプロジェクトをスキャンするときに、これらの事前にスキャンされたライブラリを含めることができます。これにより、コアプロジェクトをスキャンするたびにライブラリを再スキャンしないため、コアプロジェクト分析のパフォーマンスが改善される可能性があります。モジュラー分析では、ライブラリのソースコード、Fortify Static Code Analyzerで変換されたファイル、またはライブラリのスキャンに使用されるカスタムルールを必要とせずに、ライブラリが参照されるプロジェクトをスキャンすることもできます。これにより、必要なことはコアアプリケーションで問題を

監査することのみであるという追加の利点が得られます。分析結果は、直接制御するコードに合わせてより合理化されるため、所有していないコードの問題を心配する必要がありません。

現在、モジュラー分析はJavaおよびJakarta EE (Java EE)で開発されたライブラリおよびアプリケーションで使用できます。

注: このリリースでは、モジュラー分析によるパフォーマンスの改善が見られない可能性もあります。Fortifyでは、今後のリリースでモジュラー分析のパフォーマンスが最適化される予定です。

次の操作のたびに、ライブラリを再スキャンする必要があります。

- 新しいバージョンのFortify Static Code Analyzerに更新する
- Fortify Security Contentを更新する
- ライブラリを変更する

モジュラーコマンドラインの例

ライブラリを個別に変換およびスキャンするには、次のコマンドを入力します。

```
sourceanalyzer -b LibA MyLibs/A/*.java

sourceanalyzer -b LibA -scan-module
```

コアプロジェクトを変換およびスキャンし、事前にスキャンされた複数のライブラリを含めるには、次のコマンドを入力します。

```
sourceanalyzer -b MyProj MyProj/*.java

sourceanalyzer -b MyProj -scan -include-modules LibA,LibB
```

この例で示したオプションについては、「"分析オプション" ページ138」を参照してください。

変換フェーズと分析フェーズの検証

Fortify Audit Workbenchの結果証明書には、スキャンによるコード分析が完了し、有効であるどうかが示されます。Fortify Audit Workbenchのプロジェクト概要には、Fortify Static Code Analyzerでスキャンされたコードに関する次の特定の情報が表示されます。

- スキャンされたファイルのリスト(ファイルサイズとタイムスタンプ付き)
- 変換に使用されるJavaクラスパス(該当する場合)
- 分析に使用されるRulepack
- Fortify Static Code Analyzerのランタイム設定とコマンドラインオプション
- 変換時または分析時に発生したエラーまたは警告
- コンピュータおよびプラットフォームの情報

注: 結果証明書を取得するには、分析フェーズの出力形式にFPRを指定する必要があります。

結果証明書情報を表示するには、Fortify Audit WorkbenchでFPRファイルを開き、 **[Tools] > [Project Summary] > [Certification]** の順に選択します。詳細については、*OpenText™ Fortify Audit Workbenchユーザガイド*を参照してください。

第4章: Javaコードの変換

このセクションでは、Javaコードを変換する方法について説明します。

Fortify Static Code Analyzerでは、Jakarta EE (Java EE)アプリケーション(JSPファイル、環境設定ファイル、展開デスクリプタを含む)、Javaバイトコード、およびLombokアノテーション付きJavaコードの分析をサポートしています。

このセクションでは、次のトピックについて説明します。

Java変換コマンドライン構文

Javaコードを変換するには、ライブラリで定義され、コード内で参照されるすべてのタイプが、ソースコード、クラスファイル、またはJARファイルに対応する定義を持っている必要があります。すべてのソースファイルをFortify Static Code Analyzerコマンドラインに含めてください。

プロジェクトにKotlinコードを参照するJavaコードが含まれている場合、JavaコードとKotlinコードが同じFortify Static Code Analyzerインスタンスに変換され、Kotlin要素へのJava参照が正しく解決されるようにしてください。KotlinからJavaへの相互運用性では、`-sourcepath`オプションで提供されるKotlinファイルをサポートしません。`-sourcepath`オプションの詳細については、「"Javaコマンドラインオプション" 次のページ」を参照してください。

Javaコードを変換するための基本的なコマンドライン構文を次の例に示します。

```
sourceanalyzer -b <build_id> -cp <classpath> <files>
```

Javaコードを使用すると、Fortify Static Code Analyzerではのいずれかを実行できます。

- コンパイラをエミュレートする。ビルドの統合に便利な場合があります
- ソースファイルを直接受け入れる。コマンドラインスキャンで便利です

AntとFortify Static Code Analyzerの統合の詳細については、「"Antとの統合" ページ126」を参照してください。

Fortify Static Code Analyzerでコンパイラをエミュレートするには、次のコマンドを入力します。

```
sourceanalyzer -b <build_id> javac [<translation_options>]
```

ファイルをFortify Static Code Analyzerに直接渡すには、次のコマンドを入力します。

```
sourceanalyzer -b <build_id> -cp <classpath> [<translation_options>]
<files> | <file_specifiers>
```

ここで:

- *<translation_options>*は、コンパイラに渡されるオプションです。
- `-cp` *<classpath>*は、Javaソースコードに使用するクラスパスを指定します。
  プロジェクトをビルドするために通常使用されるJAR依存関係を含めます。複数のパスはセミコロン(Windows)またはコロン(Windows以外)で区切ります。
  javacと同様に、Fortify Static Code Analyzerではクラスをクラスパスに出現する順序でロードします。リスト内に同じ名前のクラスが複数ある場合、Fortify Static Code Analyzerでは最初にロードされたクラスを使用します。次の例では、`A.jar`と`B.jar`の両方に`MyData.class`という名前のクラスが含まれる場合、Fortify Static Code Analyzerでは`A.jar`からの`MyData.class`を使用します。

  ```
  sourceanalyzer -cp A.jar:B.jar myfile.java
  ```

  Fortifyでは、`-cp`オプションで重複するクラスを使用しないように強く推奨します。
  Fortify Static Code AnalyzerではJARファイルを次の順序でロードします。

  a. `-cp`オプションから
  b. `jre/lib`から
  c. *<sca_install_dir>*`/Core/default_jars`から

  これにより、`-cp`オプションで指定したJARに同様の名前のクラスを含めることで、ライブラリクラスを上書きできます。

使用可能なすべてのJava固有のコマンドラインオプションの詳細については、"Javaコマンドラインオプション" 下を参照してください。

Javaコマンドラインオプション

次の表では、Javaコマンドラインオプション(Java SEおよびJakarta EE用)について説明します。

JavaまたはJakarta EEのオプション	説明
`-appserver weblogic|websphere`	JSPファイルを処理するアプリケーションサーバを指定します。 **同等のプロパティ名:** `com.fortify.sca.AppServer`
`-appserver-home` *<dir>*	アプリケーションサーバのホームを指定します。 • WebLogicの場合、これは`server/lib`ディレクトリを含むディレクトリへのパスです。 • WebSphereの場合、これは`JspBatchCompiler`スクリプト

JavaまたはJakarta EEのオプション	説明
	を含むディレクトリへのパスです。 **同等のプロパティ名:** `com.fortify.sca.AppServerHome`
`-appserver-version <version>`	アプリケーションサーバのバージョンを指定します。 **同等のプロパティ名:** `com.fortify.sca.AppServerVersion`

`-cp <dirs>`|
`-classpath <dirs>`
	Javaソースコードの分析時に使用するクラスパスを指定します。形式はjavacと同じです。ディレクトリのセミコロン区切りまたはコロン区切りリストです。次の例に示すように、Fortify Static Code Analyzerファイル指定子を使用することもできます。
	
	`-cp "build/classes:lib/*.jar"`
	
	ファイル指定子の詳細については、"ファイルとディレクトリの指定" ページ149を参照してください。
	
	同等のプロパティ名:
	`com.fortify.sca.JavaClasspath`

`-extdirs <dirs>`	javac `extdirs`オプションと同様に、ディレクトリのセミコロン区切りまたはコロン区切りリストを使用できます。これらのディレクトリにあるJARファイルは、クラスパスに暗黙的に含まれます。 **同等のプロパティ名:** `com.fortify.sca.JavaExtdirs`
`-java-build-dir <dirs>`	コンパイル済みJavaソースを含む1つ以上のディレクトリを指定します。
`-source <version>`| `-jdk <version>`	Javaコードを記述するJDKバージョンを示します。サポートされているバージョンについては、『*Fortifyソフトウェアシステム要件*』ドキュメントを参照してください。デフォルトはバージョン11です。 **同等のプロパティ名:** `com.fortify.sca.JdkVersion`
`-custom-jdk-dir`	JDKを含むディレクトリを指定します。Fortify Static Code

JavaまたはJakarta EEのオプション	説明
	Analyzerインストール(*<sca_install_dir>*/Core/bootcp/)に含まれていないバージョンを指定するには、このオプションを使用します。サポートされているバージョンについては、『*Fortify ソフトウェアシステム要件*』ドキュメントを参照してください。 **同等のプロパティ名:** com.fortify.sca.CustomJdkDir
-show-unresolved-symbols	変換されたJavaソースファイルで参照されている未解決のタイプ、フィールド、および関数が、変換の最後に表示されます。リストされるのは、レシーバータイプが解決済みJavaタイプであるフィールドおよび関数参照のみです。各クラス、フィールド、および関数が、コード内で最初に変換された出現箇所のソース情報とともに表示されます。この情報はログファイルにも書き込まれます。 **同等のプロパティ名:** com.fortify.sca.ShowUnresolvedSymbols
-sourcepath *<dirs>*	スキャンに含まれていないが名前解決に使用されるソースコードを含むディレクトリのリストをセミコロン区切りまたはコロン区切りで指定します。ソースパスはクラスパスに似ていますが、解決にはクラスファイルではなくソースファイルが使用されます。ターゲットファイルリストによって参照されるソースファイルのみが変換されます。 **同等のプロパティ名:** com.fortify.sca.JavaSourcePath

参照情報

"JavaおよびKotlinのプロパティ" ページ198

Javaのコマンドライン例

クラスパスとしてjavaee.jarが指定された1つのファイルMyServlet.javaを変換するには、次のコマンドを入力します。

```
sourceanalyzer -b MyServlet -cp lib/javaee.jar MyServlet.java
```

libディレクトリ内のすべてのJARファイルをクラスパスとして使用してsrcディレクトリ内のすべての.javaファイルを変換するには、次のコマンドを入力します。

```
sourceanalyzer -b MyProject -cp "lib/*.jar" "src/**/*.java"
```

javacコンパイラを使用してMyCode.javaファイルを変換およびコンパイルするには、次のコマンドを入力します。

```
sourceanalyzer -b MyProject javac -classpath libs.jar MyCode.java
```

Javaの警告の処理

変換中に生成されたすべての警告を表示するには、スキャンフェーズを開始する前に次のコマンドを入力します。

```
sourceanalyzer -b <build_id> -show-build-warnings
```

Java変換の警告

Javaコードを変換すると、次の警告が表示される場合があります。

警告	説明/解決策
Unable to resolve type... Unable to resolve function... Unable to resolve field... Unable to locate import... Unable to resolve symbol...	これらの警告は通常、リソースが不足している場合に発生します。たとえば、アプリケーションのビルドに必要な一部の.jarおよび.classファイルが指定されていない可能性があります。 これらの警告を解決するには、アプリケーションが使用する必要なファイルをすべて含める必要があります。
Multiple definitions found for class...	この警告は通常、Javaファイル内でクラスが重複している場合に発生します。

警告	説明/解決策
	これらの警告を解決するには、警告に表示されているソースファイルは同一のファイルが重複して変換対象のソースファイルに複数回含められているものではないことを(たとえば、同じプロジェクトの2つのバージョンが含まれていることを)確認します。重複が存在する場合は、変換するファイルからその1つを削除します。そうすると、Fortify Static Code Analyzerで使用するクラスのバージョンを決定できるようになります。

Jakarta EE (Java EE)アプリケーションの変換

Jakarta EEアプリケーションを変換するには、Fortify Static Code AnalyzerでJavaソースファイルおよびJakarta EEコンポーネント(JSPファイル、展開デスクリプタ、および環境設定ファイルなど)を処理します。Jakarta EEアプリケーション内のすべての関連ファイルを1ステップで処理することができますが、プロジェクトでは、ビルドプロセスに統合したり、組織内のさまざまな利害関係者のニーズを満たしたりするために、手順をコンポーネントに分割する必要が生じることがあります。

Javaファイルの変換

Jakarta EEアプリケーションを変換するには、Javaファイルの変換に使用したものと同じ手順を使用します。たとえば、"Javaのコマンドライン例" 前のページを参照してください。

JSPプロジェクト、環境設定ファイル、および展開ディスクリプタの変換

Jakarta EE (Java EE)アプリケーションのJavaファイルを変換するほかに、JSPファイル、環境設定ファイル、および展開デスクリプタの変換が必要になることがあります。JSPファイルは、Webアプリケーションアーカイブ(WAR)の一部である必要があります。ソースディレクトリがすでにWARファイル形式で構成されている場合は、ソースディレクトリからJSPファイルを直接変換できます。そうでない場合は、アプリケーションを展開し、展開ディレクトリからJSPファイルを変換する必要があります。

例:

```
sourceanalyzer -b MyJavaApp "/**/*.jsp" "/**/*.xml"
```

ここで、`/**/*.jsp`はJSPプロジェクトファイルの場所を参照し、`/**/*.xml`は環境設定ファイルおよび展開ディスクリプタファイルの場所を参照します。

Jakarta EE (Java EE)変換の警告

Jakarta EEアプリケーションの変換で次の警告が表示される場合があります。

```
Could not locate the root (WEB-INF) of the web application. Please build
your web application and try again. Failed to parse the following jsp
files:

<list_of_jsp_files>
```

この警告は、Webアプリケーションが標準のWARディレクトリ形式で展開されていないか、必要なライブラリの完全なセットが含まれていないことを示します。この警告を解決するには、Webアプリケーションが開かれたWARディレクトリ形式であり、アプリケーションに必要なすべての.jarファイルおよび.classファイルが正しいWEB-INF/libおよびWEB-INF/classesディレクトリに格納されていることを確認してください。また、すべてのタグのすべてのTLDファイル、およびそれらのタグの実装に対応するJARファイルが存在することも確認してください。

Javaバイトコードの変換

Fortifyでは、Javaバイトコードと JSP/Javaコードを同じsourceanalyzer呼び出しで変換しないことをお勧めします。同じビルドIDを持つ複数のsourceanalyzer呼び出しを使用して、バイトコードと JSP/Javaコードの両方を含むプロジェクトを変換します。

バイトコードを変換するには

1. fortify-sca.propertiesファイルに次のプロパティを追加します(または、-Dオプションを使用してコマンドラインにこれらのプロパティを含めます)。

   ```
   com.fortify.sca.fileextensions.class=BYTECODE
   com.fortify.sca.fileextensions.jar=ARCHIVE
   ```

   これは、Fortify Static Code Analyzerで.classおよび.jarファイルを処理する方法を指定します。
2. 次のいずれかを実行します。
   - Fortify Static Code Analyzerでバイトコードクラスを通常のJavaファイルに逆コンパイルして変換に含めることを要求します。

     fortify-sca.propertiesファイルに次のプロパティを追加します。

     ```
     com.fortify.sca.DecompileBytecode=true
     ```

     または、-Dオプションを使用して、変換フェーズのコマンドラインにこのプロパティを含めます。

     ```
     sourceanalyzer -b MyProject -Dcom.fortify.sca.DecompileBytecode=true
     -cp "lib/*.jar" "src/**/*.class"
     ```
   - Fortify Static Code Analyzerで逆コンパイルせずにバイトコードを変換することを要求します。

最善の結果を得るため、Fortifyでは完全なデバッグ情報(`javac -g`)を使用してバイトコードをコンパイルすることを推奨しています。

変換するJavaバイトコードファイルを指定することで、Fortify Static Code Analyzerの変換フェーズにバイトコードを含めます。最善のパフォーマンスを得るため、スキャンが必要な`.jar`または`.class`ファイルだけを指定します。次の例では、`.class`ファイルが変換されています。

```
sourceanalyzer -b MyProject -cp "lib/*.jar" "src/**/*.class"
```

JSP変換および分析に関する問題のトラブルシューティング

次のセクションでは、JSPの変換とスキャンに関するトラブルシューティング情報について説明します。

一部のJSPを変換できない

Fortify Static Code Analyzerでは、組み込みコンパイラまたは特定のアプリケーションサーバJSPコンパイラを使用して、分析のためにJSPファイルをJavaファイルに変換します。Fortify Static Code AnalyzerでJSPファイルをJavaファイルに変換する際にJSPパーサで問題が発生すると、次のようなメッセージが表示されます。

```
Failed to translate the following jsps into analysis model. Please see the
log file for any errors from the jsp parser and the user manual for hints
on fixing those
<list_of_jsp_files>
```

これは通常、次の1つ以上の理由で発生します。

- Webアプリケーションが適切な展開可能WARディレクトリ形式でレイアウトされていない
- アプリケーションに必要なJARファイルまたはクラスの一部が見つからない
- アプリケーションのタグライブラリまたはそれらの定義(TLD)の一部が見つからない

問題の詳細を取得するには、次の手順を実行します。

1. エディタでFortify Static Code Analyzerログファイルを開きます。
2. 次の文字列を検索します。
   - `Jsp parser stdout:`
   - `Jsp parser stderr:`

JSPパーサではこれらのエラーを生成します。エラーを解決してFortify Static Code Analyzerを再実行します。

Jakarta EEアプリケーションのスキャンの詳細については、「"Jakarta EE (Java EE)アプリケーションの変換" ページ58」を参照してください。

JSP関連カテゴリで問題の数が増加した

分析結果に含まれるクロスサイトスクリプティングなどのJSP関連カテゴリの脆弱性数が以前のFortify Static Code Analyzerバージョンと比較して大幅に増加している場合は、分析フェーズで`-legacy-jsp-dataflow`オプションを指定できます(`-scan`オプションも指定します)。このオプションを使用すると、JSP関連のデータフローに対する追加のフィルタリングが有効になり、検出される偽陽性が減少します。

`fortify-sca.properties`ファイルで指定できるこのオプションと同等のプロパティは`com.fortify.sca.jsp.LegacyDataflow`です。

第5章: Kotlinコードの変換

このセクションでは、Kotlinコードを変換する方法について説明します。

このセクションでは、次のトピックについて説明します。

Kotlinコマンドライン構文

Kotlinコードの変換は、Javaコードの変換と同様です。Kotlinコードを変換するには、ライブラリで定義され、コード内で参照されるすべてのタイプが、ソースコード、クラスファイル、またはJARファイルに対応する定義を持っている必要があります。すべてのソースファイルをFortify Static Code Analyzerコマンドラインに含めてください。

Kotlinコードを変換するための基本的なコマンドライン構文を次の例に示します。

```
sourceanalyzer -b <build_id> -cp <classpath> [<translation_options>]
<files>
```

ここで

- `-cp <classpath>`は、Kotlinソースコードに使用するクラスパスを指定します。プロジェクトをビルドするために通常使用されるJAR依存関係を含めます。複数のパスはセミコロン(Windows)またはコロン(Windows以外)で区切ります。

  Fortify Static Code Analyzerではクラスをクラスパスに出現する順序でロードします。リスト内に同じ名前のクラスが複数ある場合、Fortify Static Code Analyzerでは最初にロードされたクラスを使用します。次の例では、`A.jar`と`B.jar`の両方に`MyData.class`という名前のクラスが含まれる場合、Fortify Static Code Analyzerでは`A.jar`からの`MyData.class`を使用します。

  ```
  sourceanalyzer -cp "A.jar:B.jar" myfile.kt
  ```

  `-cp`オプションで重複するクラスを使用しないように強く推奨します。

使用可能なすべてのJava固有のコマンドラインオプションの詳細については、「"Kotlinコマンドラインオプション" 次のページ」を参照してください。

Kotlinコマンドラインオプション

次の表では、Kotlin固有のコマンドラインオプションについて説明します。

Kotlinオプション	説明

`-cp <paths>`|
`-classpath <dirs>`
	Kotlinソースコードの変換に使用するクラスパスを指定します。これは、セミコロン区切りまたはコロン区切りのディレクトリのリストです。次の例に示すように、Fortify Static Code Analyzerファイル指定子を使用することもできます。
	`-cp "build/classes:lib/*.jar"`
	ファイル指定子の詳細については、"ファイルとディレクトリの指定" ページ149を参照してください。
	同等のプロパティ名:
	`com.fortify.sca.JavaClasspath`

`-source <version>`| `-jdk <version>`	Kotlinコードを記述するJDKバージョンを示します。サポートされているバージョンについては、『*Fortifyソフトウェアシステム要件*』ドキュメントを参照してください。デフォルトはバージョン11です。 **同等のプロパティ名:** `com.fortify.sca.JdkVersion`
`-sourcepath <dirs>`	スキャンに含まれていないが名前解決に使用されるJavaソースコードを含むディレクトリのリストをセミコロン区切りまたはコロン区切りで指定します。ソースパスはクラスパスに似ていますが、解決にはクラスファイルではなくソースファイルが使用されます。ターゲットファイルリストによって参照されるソースファイルのみが変換されます。 **同等のプロパティ名:** `com.fortify.sca.JavaSourcePath`
`-jvm-default <mode>`	本体がKotlinインターフェイスに入っているメソッドの`DefaultImpls`クラスの生成を指定します。*<mode>*の有効な値は • `disable` — 本体を持つメソッドが含まれている各インターフェイスに対して、`DefaultImpls`クラスを生成するように指定します。

Kotlinオプション	説明
	• `all` — インターフェイスに`@JvmDefaultWithCompatibility`の注釈が付く場合に、`DefaultImpls`クラスを生成するように指定します。 • `all-compatibility` — インターフェイスに`@JvmDefaultWithoutCompatibility`の注釈が付かない限り、`DefaultImpls`クラスを生成するように指定します。 **同等のプロパティ名:** `com.fortify.sca.KotlinJvmDefault`

参照情報

"JavaおよびKotlinのプロパティ" ページ198

Kotlinコマンドラインの例

クラスパスとして`A.jar`が指定された1つのファイル`MyKotlin.kt`を変換するには、次のコマンドを入力します。

```
sourceanalyzer -b MyProject -cp lib/A.jar MyKotlin.kt
```

`lib`ディレクトリ内のすべてのJARファイルをクラスパスとして使用して`src`ディレクトリ内のすべての`.kt`ファイルを変換するには、次のコマンドを入力します。

```
sourceanalyzer -b MyProject -cp "lib/**/*.jar" "src/**/*.kt"
```

gradlewを使用してgradleプロジェクトを変換するには、次のコマンドを入力します。

```
sourceanalyzer -b MyProject gradlew clean assemble
```

`lib`ディレクトリおよびサブディレクトリ内の`src/java`およびすべてのJARファイルのJava依存関係をクラスパスとして使用して`src`ディレクトリ内のすべてのファイルを変換するには、次のコマンドを入力します。

```
sourceanalyzer -b MyProject -cp "lib/**/*.jar" -sourcepath "src/java" "src"
```

KotlinとJavaの変換相互運用性

プロジェクトにJavaコードを参照するKotlinコードが含まれている場合、別のKotlinファイルを参照するKotlinファイルの場合と同じ方法で、Javaファイルを変換ツールに渡すことができます。変換されたプロジェクトソースの一部として、または`-sourcepath`パラメータとして渡すことができます。

プロジェクトにKotlinコードを参照するJavaコードが含まれている場合、JavaコードとKotlinコードが同じFortify Static Code Analyzerインスタンスに変換され、Kotlin要素へのJava参照が正しく解決されるようにしてください。KotlinからJavaへの相互運用性では、`-sourcepath`オプションで提供されるKotlinファイルをサポートしません。`-sourcepath`オプションの詳細については、「"Kotlinコマンドラインオプション" ページ63」を参照してください。

Kotlinスクリプトの変換

Fortify Static Code Analyzerでは、試験的なスクリプトのカスタマイズを除くKotlinスクリプトの変換をサポートしています。スクリプトのカスタマイズには、外部プロパティの追加、静的または動的な依存関係の提供などがあります。スクリプト定義(テンプレート)はカスタムスクリプトの作成に使用され、テンプレートは`*.kts`拡張子に基づいてスクリプトに適用されます。Fortify Static Code Analyzerでは`*.kts`ファイルを変換しますが、これらのテンプレートは適用されません。

第6章: Visual Studioプロジェクトの変換

Fortify Static Code Analyzerでは、次のタイプのVisual Studioプロジェクトの変換をサポートするためのビルド統合を提供しています。

- C/C++プロジェクト
- .NET FrameworkとNET Coreをターゲットとするc#プロジェクト
- ASP.NETフレームワークとASP.NET Coreをターゲットとするasp.netアプリケーション
- AndroidおよびiOSプラットフォームをターゲットとするXamarinアプリケーション

関連するプログラミング言語およびフレームワークと、Visual StudioおよびMSBuildのサポートされているバージョンの一覧については、『*Fortify ソフトウェアシステム要件*』を参照してください。

このセクションでは、次のトピックについて説明します。

Visual Studioプロジェクト変換の前提条件

Fortifyでは、変換する各プロジェクトを完成させ、エラーなしでビルドできる環境で変換を実行することをお勧めします。ソフトウェア環境要件のリストについては、『*Fortify ソフトウェアシステム要件*』を参照してください。完全なプロジェクトには、次の要素が含まれます。

- 必要なすべてのソースコードファイル(C/C++、C#、またはVB.NET)。
- 必要なすべての参照ライブラリ。

  これには、関連するフレームワークの参照ライブラリ、NuGetパッケージ、およびサードパーティ製ライブラリが含まれます。
- C/C++プロジェクトの場合は、Visual StudioまたはMSBuildインストールに属していない必要なすべてのヘッダファイルを含めます。
- ASP.NETおよびASP.NET Coreプロジェクトの場合は、必要なすべてのASP.NETページファイルを含めます。

  サポートされているASP.NETページのタイプは、ASAX、ASCX、ASHX、ASMX、ASPX、AXML、BAML、CSHTML、Master、RAZOR、VBHTML、およびXAMLです。

Visual Studioプロジェクトのコマンドライン構文

Visual Studioソリューションまたはプロジェクトを変換する基本的な構文では、Fortify Static Code Analyzer変換コマンドの一部として、プロジェクトの対応するビルドオプションを指定します。そのようにすると、ソリューションおよびプロジェクトファイルを分析し、適切な変換ステップを自動的に実行するビルド統合が開始されます。

重要 適切なプロジェクト依存関係とリソースのすべてをビルド統合が間違いなく取得できるようにするため、Fortify Static Code Analyzerコマンドはビルドの環境設定にアクセスできるコマンドプロンプトから実行する必要があります。Fortifyでは、変換に最適な環境を確保するために、Visual Studioの開発者コマンドプロンプトからこのコマンドを実行するよう強く推奨します。

次の例では、Visual Studioソリューション`Sample.sln`に含まれるすべてのプロジェクトがFortify Static Code Analyzerによって変換されます。セミコロンで区切ったプロジェクトのリストを指定して、1つ以上の特定のプロジェクトを変換することもできます。

- WindowsまたはLinux上の.NET 6.0以降のソリューションの場合は、次のコマンドを使用してソリューションを変換します。
  a. 必要に応じて次のコマンドを実行し、以前のプロジェクトビルドから中間ファイルを削除します。

     ```
     dotnet clean Sample.sln
     ```

  b. 必要に応じて次のコマンドを実行し、必要なすべての参照ライブラリがダウンロードされ、プロジェクトにインストールされるようにします。次のコマンドをプロジェクトの最上位フォルダから実行します。

     ```
     dotnet restore Sample.sln
     ```

  c. プロジェクトのビルドの実装方法に応じて、次のFortify Static Code Analyzerコマンドのいずれかを実行します。このコマンドには、追加のビルドパラメータを含めることができます。

     ```
     sourceanalyzer -b MyProject dotnet msbuild Sample.sln
     ```

     または

     ```
     sourceanalyzer -b MyProject dotnet build Sample.sln
     ```

- Windows上のC、C++、および.NET Frameworkソリューション(4.8.x以前)の場合は、次のコマンドを使用してソリューションを変換します。

  ```
  sourceanalyzer -b MyProject msbuild /t:rebuild Sample.sln
  ```

  注: Visual Studioの開発者コマンドプロンプトではなくWindowsのコマンドプロンプトからFortify Static Code Analyzerを実行する場合は、環境を設定し、プロジェクトのビルドに必要なMSBuild実行可能ファイルへのパスをPATH環境変数に含める必要があります。

変換が完了したら、次の例に示すように、分析フェーズを実行して結果をFPRファイルに保存できます。

```
sourceanalyzer –b MyProject -scan -f MyResults.fpr
```

Visual Studioプロジェクトの変換に関する特殊なケースの処理

スクリプトからの変換の実行

スクリプトなどの非対話型モードで変換を実行するには、Fortify Static Code Analyzer変換を実行する前に次のコマンドを実行して、変換に最適な環境を確立してください。

```
cmd.exe /k <vs_install_dir>/Common7/Tools/VSDevCmd.bat
```

ここで、*<vs_install_dir>*はVisual Studioをインストールしたディレクトリです。

プレーンな.NETおよびASP.NETプロジェクトの変換

プレーンな.NETおよびASP.NETプロジェクトは、Windowsコマンドプロンプトおよび Visual Studio環境から変換できます。Windowsコマンドプロンプトから変換する場合は、プロジェクトのビルドに必要なMSBuild実行可能ファイルへのパスがPATH環境変数に含まれていることを確認してください。

C/C++およびXamarinプロジェクトの変換

C/C++およびXamarinプロジェクトは、Visual Studioの開発者コマンドプロンプトまたはFortify Extension for Visual Studioから変換する必要があります。

注: Xamarinプロジェクトでは、対応するネイティブAndroid APIのルールがFortify Secure Coding Rulepacksに存在する場合、Xamarin.Android APIのカスタムルールを使用する必要はありません。使用すると、重複した問題が報告される可能性があります。

空白を含む設定を持つプロジェクトの変換

空白を含む環境設定またはその他の設定ファイルを使用してプロジェクトがビルドされている場合は、設定値を引用符で囲む必要があります。たとえば、環境設定`My Configuration`を使用してビルドされたVisual Studioソリューション`Sample.sln`を変換するには、次のコマンドを使用します。

```
sourceanalyzer –b MySampleProj msbuild /t:rebuild /p:Configuration="My
Configuration" Sample.sln
```

Visual Studioソリューションの単一プロジェクトの変換

Visual Studioソリューションに複数のプロジェクトが含まれている場合は、ソリューション全体ではなく1つのプロジェクトを変換することもできます。プロジェクトファイルには、projで終わるファイル名拡張子(.vcxprojや.csprojなど)が付いています。1つのプロジェクトを変換するには、MSBuildコマンドのパラメータとしてソリューションの代わりにプロジェクトファイルを指定します。

次の例では、Sample.vcxprojプロジェクトファイルを変換します。

```
sourceanalyzer -b MySampleProj msbuild /t:rebuild Sample.vcxproj
```

複数の実行可能ファイルをビルドするプロジェクトの分析

Visual StudioまたはMSBuildプロジェクトで複数の実行可能ファイル(ファイル名の拡張子*.exeを持つファイルなど)をビルドする場合は、分析結果の誤検知の問題を避けるため、実行ファイルごとに個別に分析フェーズを実行することを強くお勧めします。Fortify これを行うには、分析フェーズの実行時に-binary-nameオプションを使用して、実行可能ファイル名またはNETアセンブリ名をパラメータとして指定します。

次の例は、Sample1 (.NETアセンブリ名が関連付けられていないC++プロジェクト)とSample2 (.NETアセンブリ名Sample2を持つ.NETプロジェクト)という2つのプロジェクトで構成されるVisual StudioソリューションSample.slnを変換して分析する方法を示しています。各プロジェクトは、それぞれ別個の実行可能ファイル(Sample1.exeとSample2.exe)をビルドします。分析結果はSample1.fprファイルとSample2.fprファイルに保存されます。

```
sourceanalyzer -b MySampleProj msbuild /t:rebuild Sample.sln sourceanalyzer
-b MySampleProj -scan -binary-name Sample1.exe -f Sample1.fpr
sourceanalyzer -b MySampleProj -scan -binary-name Sample2.exe -f
Sample2.fpr
```

-binary-nameオプションについて詳しくは、"分析オプション" ページ138を参照してください。

Visual Studioプロジェクトを変換する別の方法

このセクションでは、Visual Studioプロジェクトを変換する別の方法について説明します。

Visual Studioソリューション用の別の変換オプション

Visual Studioソリューションでのみ使用できる変換方法には、次の2種類があります。

- Fortify Extension for Visual Studioを使用する
  Fortify Extension for Visual Studioは、変換および分析(スキャン)フェーズを1ステップで実行します。
- Fortify Static Code Analyzerコマンドにdevenvコマンドを追加する

次のコマンドは、Visual StudioソリューションSample.slnを変換します。

```
sourceanalyzer –b MySampleProj devenv Sample.sln /rebuild
```

Fortify Static Code Analyzerはdevenvの呼び出しを同等のMSBuildの呼び出しに変換するため、この場合、このコマンドを使用するソリューションはdevenvツールではなくMSBuildによってビルドされます。

Fortify Static Code Analyzerを明示的に実行せずに変換する

Fortify Static Code Analyzerを直接起動せずにVisual Studioプロジェクトを変換するオプションがあります。これには、DotNetおよびFrameworkディレクトリの*<sca_install_dir>*\Core\private-bin\sca\MSBuildPluginにあるFortify.targetsファイルが必要です。プロジェクトをビルドするビルドコマンドラインで絶対パスまたは相対パスを使用してファイルを指定できます。使用しているビルドコマンド(dotnet.exeまたはMSBuild.exe)に応じて、それぞれDotnetディレクトリまたはFrameworkディレクトリを含むパスを使用します。例:

```
dotnet.exe msbuild /t:rebuild /p:CustomAfterMicrosoftCommonTargets=<sca_
install_dir>\Core\private-bin\sca\MSBuildPlugin\Dotnet\Fortify.targets
Sample.sln
```

または

```
msbuild.exe /t:rebuild /p:CustomAfterMicrosoftCommonTargets=<sca_install_
dir>\Core\private-bin\sca\MSBuildPlugin\Framework\Fortify.targets Sample.sln
```

プロジェクトの変換を設定するために設定できる環境変数は複数あります。これらの変数の多くはデフォルト値を持ち、Fortify Static Code Analyzerでは変数が設定されていない場合に使用されます。これらの変数を次の表に示します。

環境変数	説明	デフォルト値
FORTIFY_MSBUILD_BUILDID	変換のFortify Static Code Analyzerビルド IDを指定します。この値は必ず設定してください。 これは、Fortify Static Code Analyzer -bオプションと同じです。	なし
FORTIFY_MSBUILD_DEBUG	デバッグモードを有効にします。これは、Fortify Static Code Analyzer –debugオプションと同じです。	False

環境変数	説明	デフォルト値
FORTIFY_MSBUILD_DEBUG_VERBOSE	冗長デバッグモードを有効にします。これは、Fortify Static Code Analyzer `–debug-verbose`オプションと同じです。両方がtrueに設定されている場合は、FORTIFY_MSBUILD_DEBUGよりも優先されます。	False
FORTIFY_MSBUILD_MEM	変換のメモリ要件をJVM `-Xmx`オプションの形式で指定します。たとえば、`-Xmx2G`になります。	システムで使用可能な物理メモリに基づいて自動割り当て
FORTIFY_MSBUILD_SCALOG	Fortify Static Code Analyzerログファイルの場所(絶対パス)を指定します。 これは、Fortify Static Code Analyzer `-logfile`オプションと同じです。	`%LOCALAPPDATA%/Fortify/sca/log/sca.log`

第7章: CおよびC++コードの変換

このセクションでは、CおよびC++コードを変換する方法について説明します。

重要 この章では、Visual StudioまたはMSBuildプロジェクトの一部 ***ではない*** CおよびC++コードを変換する方法について説明します。Visual StudioまたはMSBuildプロジェクトの変換方法については、"Visual Studioプロジェクトの変換" ページ66を参照してください。

このセクションでは、次のトピックについて説明します。

CおよびC++コード変換の前提条件

プロジェクトをビルドするために必要な依存関係(サードパーティ製ライブラリのヘッダを含む)があることを確認してください。Fortify Static Code Analyzer変換では、オブジェクトファイルと静的/動的ライブラリのファイルは必要ありません。

注: Fortify Static Code Analyzerは、必ずしもすべての標準以外のC++構造体をサポートしていません。

Gradleを使用してC++プロジェクトをビルドする場合は、C++アプリケーションプラグインが次のいずれかの形式でGradleファイルに追加されていることを確認してください。

```
apply plugin: 'cpp'
```

```
plugins { id 'cpp-application' }
```

Gradleとの統合について詳しくは、"ビルド統合" ページ124を参照してください。

CおよびC++のコマンドライン構文

コンパイラに渡されるコマンドラインオプションは、プリプロセッサの実行に影響を与え、言語の機能や拡張機能を有効または無効にすることができます。Fortify Static Code AnalyzerでコンパイラとおなじようにソースコードをCおよびC++ソースコードの変換フェーズで完全なコンパイラコマンドラインが必要です。元のコンパイラコマンドの前に`sourceanalyzer`コマンドとオプションを指定します。

1つのファイルを変換する基本的なコマンドライン構文は次のとおりです。

```
sourceanalyzer -b <build_id> [<sca_options>] <compiler> [<compiler_
options>] <file>.c
```

ここで:

- *<sca_options>*は、Fortify Static Code Analyzerに渡されるオプションです。
- *<compiler>*は、使用するC/C++コンパイラの名前(`gcc`、`g++`、`cl`など)です。サポートされているC/C++コンパイラのリストについては、*Fortifyソフトウェアシステム要件*のドキュメントを参照してください。
- *<compiler_options>*は、C/C++コンパイラに渡されるオプションです。
- *<file>*`.c`は、ASCIIまたはUTF-8エンコーディング形式である必要があります。

注: Fortify Static Code Analyzerのすべてのオプションをコンパイラオプションの前に指定する必要があります。

コンパイラコマンドは、単独で実行したときに正常に完了する必要があります。コンパイラコマンドが失敗する場合は、コンパイラコマンドの前に指定したFortify Static Code Analyzerコマンドも失敗します。

たとえば、次のコマンドを使用してファイルをコンパイルするとします。

```
gcc -I. -o hello.o -c helloworld.c
```

この場合、次のコマンドを使用してこのファイルを変換できます。

```
sourceanalyzer -b MyProject gcc -I. -o hello.o -c helloworld.c
```

Fortify Static Code Analyzerは、変換フェーズの一部として元のコンパイラコマンドを実行します。前のコマンド例では、スキャンに適した変換済みのソースと、`gcc`を実行して得られるオブジェクトファイル`hello.o`の両方が生成されます。Fortify Static Code Analyzer `-nc`オプションを使用して、コンパイラの実行を無効にすることができます。

前処理されたCおよびC++コードのスキャン

コンパイルの前にC/C++ビルドでFortify Static Code Analyzerがサポートしていないサードパーティ製のCプリプロセッサを実行する場合は、中間ファイルでFortify Static Code Analyzer変換を開始する必要があります。Fortify Static Code Analyzerのタッチレスビルド統合では、ビルドでサポートされていないプリプロセッサとサポートされているコンパイラをパイプチェーンではなく一時ファイルで接続された2つのコマンドとして実行した場合に、中間ファイルが自動的に変換されます。

C/C++のプリコンパイル済みヘッダファイル

一部のC/C++コンパイラは、コンパイルのパフォーマンスを向上させるプリコンパイル済みヘッダファイルをサポートしています。コンパイラによっては、この機能の実装によって微妙な副作用が発生します。この

機能を有効にすると、コンパイラが警告やエラーなしで誤ったソースコードを受け入れる可能性があります。その結果、Fortify Static Code Analyzerで変換エラーが報告されてもコンパイラでは報告されないという矛盾が発生する可能性があります。

コンパイラのプリコンパイル済みヘッダ機能を使用する場合は、プリコンパイル済みヘッダを無効にしてから、完全ビルドを実行してソースコードが正しくコンパイルされるのを確認してください。

第8章: JavaScriptおよびTypeScriptコードの変換

JavaScript、TypeScript、JSX、およびTSXソースファイルが含まれるJavaScriptプロジェクト、およびHTMLファイルに埋め込まれているJavaScriptを分析できます。

一部のJavaScriptフレームワークは、平文のJavaScriptにトランスパイルされ(ソースからソースへのコンパイル)、これが生成されるコードになります。このタイプのコードを除外するには、`-exclude`コマンドラインオプションを使用してください。

JavaScriptコードおよびTypeScriptコードを変換する場合は、すべてのソースファイルを1回の呼び出しで一緒に指定してください。Fortify Static Code Analyzerでは、その後の呼び出し時にビルドIDに関連付けられたファイルリストに新しいファイルを追加する機能はサポートされていません。

Fortify Static Code Analyzerでは、圧縮されたJavaScript (*.min.js)を変換しません。

注: 変換から除外する圧縮されたJavaScriptファイルには、Fortify Static Code Analyzerで自動的に検出できないタイプがあります。これらのファイルは、`-exclude`コマンドラインオプションを使用して直接除外してください。

このセクションでは、次のトピックについて説明します。

ピュアJavaScriptプロジェクトの変換

JavaScriptを変換する基本的なコマンドライン構文は次のとおりです。

```
sourceanalyzer -b <build_id> <js_file_or_dir>
```

ここで、*<js_file_or_dir>*は変換するJavaScriptファイルの名前、または複数のJavaScriptファイルを含むディレクトリのいずれかです。*<js_file_or_dir>*に`*.js`を指定して、複数のファイルを変換することもできます。

依存関係の除外

特定の依存関係の変換を回避するには、fortify-sca.propertiesファイル内の適切なプロパティ設定に追加します。次のプロパティで指定されたファイルは変換されません。

- com.fortify.sca.skip.libraries.ES6
- com.fortify.sca.skip.libraries.jQuery
- com.fortify.sca.skip.libraries.javascript
- com.fortify.sca.skip.libraries.typescript

各プロパティには、ファイル名(パス情報なし)のカンマまたはコロン区切りリストを指定します。

これらのプロパティで指定されたファイルは、ローカルファイルとインターネット上のファイルの両方に適用されます。たとえば、JavaScriptコードに次のローカルファイル参照が含まれるとします。

```
<script src="js/jquery-ui.js" type="text/javascript" charset="utf-
8"></script>
```

デフォルトでは、fortify-sca.propertiesファイル内のcom.fortify.sca.skip.libraries.jQueryプロパティにjquery-us.jsが含まれるため、Fortify Static Code Analyzerでは前の例に示したファイルを変換しません。

ファイル名には正規表現を使用できます。Fortify Static Code Analyzerではcom.fortify.sca.skip.libraries.jQueryプロパティ値に含まれる各ファイル名の.min.jsまたは.jsの前に正規表現'(-?\d+\.\d+\.\d+)?'を自動的に挿入します。

注: -excludeコマンドラインオプションを使用して、ローカルファイルまたはディレクトリ全体を除外することもできます。このオプションの詳細については、"変換オプション" ページ136を参照してください。

詳細な分析を行うために、依存関係の言語が-disable-languageオプションで無効になっている場合でも、依存関係ファイルが変換に含まれます。言語を無効にするオプションの詳細については、「"変換オプション" ページ136」を参照してください。

NPM依存関係の変換の管理

デフォルトで、Fortify Static Code Analyzerではコードにインポートされたnpm依存関係のみを変換します。npm依存関係の変換を管理するには、次の3つのオプションがあります。

- com.fortify.sca.follow.importsプロパティはデフォルトで有効になっており、プロジェクトで使用されているすべてのインポートされたファイル(npm依存関係を含む)を解決し、変換に含めるようにFortify Static Code Analyzerに指示します。このプロジェクト内のインポートされたファイルを見つけるために、Fortify Static Code AnalyzerはNode.jsと似たアルゴリズムを使用します(詳細については、Node.jsのWebサイトを参照してください)。

このプロパティをfalseに設定すると、コマンドラインに明示的に含まれていないインポートされたNPM依存関係は変換に含まれなくなります。

- `com.fortify.sca.exclude.unimported.node.modules`プロパティはデフォルトで有効になっており、プロジェクトで参照されていないnode_modulesディレクトリを除外するようにFortify Static Code Analyzerに指示します。このプロパティはデフォルトで有効になっています。これは、ビルドシステムにのみ必要な依存関係など、最終プロジェクトには必要ない依存関係の変換を避けるためです。

  このプロパティをfalseに設定すると、Fortify Static Code Analyzerでは、プロジェクトによって参照されていない、(`com.fortify.sca.follow.imports`プロパティが有効の解決中に検出されたすべてのモジュールが変換に含められます。
- `-exclude`オプションは、モジュールを明示的に除外するために、上記の2つのプロパティと一緒に使用できます。

  このオプションを使用すると、上記のプロパティ設定よりも優先されます。

参照情報

"NPM依存関係の除外の例" 下

NPM依存関係の除外の例

次の例は、NPM依存関係を除外する3つの異なるシナリオを示しています。これらの例では、次のディレクトリ構造を使用します。

```
./ RootProjectDir innerSrcDir node_modules innerProjectReferencedModule index.ts
moduleNotReferencedByProject index.ts innerProject.ts (innerProjectReferencedModule
からのインポートを含む) node_modules projectReferencedModule index.ts
moduleNotReferencedByProject index.ts projectMain.ts (projectReferencedModuleからのイ
ンポートを含む)
```

例1

この例は、ファイルがデフォルトの動作で変換される場合を示しています。この場合、`com.fortify.sca.follow.imports`と`com.fortify.sca.exclude.unimported.node.modules`が両方とも`true`に設定されています。

```
sourceanalyzer RootProjectDir/
```

例1の変換には、次のファイルが含まれます。

```
./RootProjectDir/innerSrcDir/innerProject.ts
./RootProjectDir/innerSrcDir/node_
modules/innerProjectReferencedModule/index.ts ./RootProjectDir/projectMain.ts
./RootProjectDir/node_modules/projectReferencedModule/index.ts
```

例2

この例は、プロジェクトによって参照されているモジュールに加えて、解決中に見つかったがプロジェクトによって参照されていないモジュールも変換に含める場合を示しています。

```
sourceanalyzer RootProjectDir/ -
Dcom.fortify.sca.exclude.unimported.node.modules=false
```

例2の変換には、次のファイルが含まれます。

```
./RootProjectDir/innerSrcDir/innerProject.ts
./RootProjectDir/innerSrcDir/node_
modules/innerProjectReferencedModule/index.ts
./RootProjectDir/innerSrcDir/node_
modules/moduleNotReferencedByProject/index.ts ./RootProjectDir/projectMain.ts
./RootProjectDir/node_modules/projectReferencedModule/index.ts
./RootProjectDir/node_modules/moduleNotReferencedByProject/index.ts
```

例3

この例は、-excludeオプションを使用して、任意のnode_modulesディレクトリ内のすべてのファイルを除外する場合を示しています。-excludeオプションは、 com.fortify.sca.follow.importsおよびcom.fortify.sca.exclude.unimported.node.modulesプロパティの設定に基づくモジュールの解決策を上書きします。

```
sourceanalyzer RootProjectDir/ -exclude "**/node_modules/*.*"
```

例3の変換には、次のファイルが含まれます。

```
./RootProjectDir/innerSrcDir/innerProject.ts ./RootProjectDir/projectMain.ts
```

HTMLファイルを使用したJavaScriptプロジェクトの変換

プロジェクトにJavaScriptファイルに加えてHTMLファイルが含まれている場合は、次の例に示すように、fortify-sca.propertiesファイルまたはコマンドラインでcom.fortify.sca.EnableDOMModelingプロパティをtrueに設定します。

```
sourceanalyzer –b MyProject <js_file_or_dir> -
Dcom.fortify.sca.EnableDOMModeling=true
```

com.fortify.sca.EnableDOMModelingプロパティをtrueに設定すると、DOM関連のクロスサイトスクリプティングの問題など、DOM関連の攻撃に関する偽陰性レポートを減らすことができます。

注: このオプションを有効にすると、Fortify Static Code AnalyzerではHTMLファイル内のDOMツリー構造をモデル化するJavaScriptコードが生成されます。分析フェーズの時間が長くなる場合があります(分析する変換済みコードが多くなるため)。

com.fortify.sca.EnableDOMModelingプロパティをtrueに設定した場合は、Fortify Static Code AnalyzerがDOMモデリングに含める追加のHTMLタグをcom.fortify.sca.DOMModeling.tagsプロパ

ティで指定することもできます。デフォルトで、Fortify Static Code AnalyzerではHTMLタグbody、button、div、form、iframe、input、head、html、およびpを含めます。

たとえば、HTMLタグulおよびliをDOMモデルに含めるには、次のコマンドを使用します。

```
sourceanalyzer –b MyProject <js_file_or_dir> -
Dcom.fortify.sca.DOMModeling.tags=ul,li
```

外部JavaScriptまたはHTMLを変換に含める

src属性で指定された外部JavaScriptまたはHTMLファイルを含めるため、Fortify Static Code Analyzerでダウンロードして変換フェーズに含めることができるドメインを指定できます。これを行うには、com.fortify.sca.JavaScript.src.domain.whitelistプロパティで1つ以上のドメインを指定します。

注: このプロパティは、fortify-sca.propertiesファイル内でグローバルに設定できます。

たとえば、HTMLファイルに次のステートメントが含まれているとします。

```
<script src='http://xyzdomain.com/foo/bar.js' language='text/javascript'/>
</script>
```

xyzdomain.comドメインがファイルをダウンロードするのに安全な場所であると確信している場合は、次のプロパティ指定をコマンドラインに追加して、変換フェーズに含めることができます。

```
-Dcom.fortify.sca.JavaScript.src.domain.whitelist="xyzdomain.com/foo"
```

注: プロパティ値では、ドメインからwww.プレフィクスを省略できます。たとえば、元のHTMLファイルのsrcタグでwww.google.comからファイルをダウンロードするように指定している場合は、google.comドメインを指定するだけです。

複数のドメインを信頼するには、次の例に示すように、縦棒文字(|)区切りで各ドメインを含める必要があります。

```
-Dcom.fortify.sca.JavaScript.src.domain.whitelist=
"xyzdomain.com/foo|abcdomain.com|123.456domain.com”
```

プロキシサーバを使用している場合は、次の例に示すように、プロキシサーバ情報をコマンドラインに含める必要があります。

```
-Dhttp.proxyHost=example.proxy.com -Dhttp.proxyPort=8080
```

プロキシサーバオプションの完全なリストについては、ネットワーキングプロパティに関するJavaのドキュメントを参照してください。

第9章: Pythonコードの変換

Fortify Static Code AnalyzerではPythonアプリケーションを変換し、.py拡張子を持つファイルをPythonソースコードとして処理します。

このセクションでは、次のトピックについて説明します。

Python変換コマンドライン構文

Pythonコードを変換する基本的なコマンドライン構文は次のとおりです。

```
sourceanalyzer -b <build_id> -python-version <python_version> -python-
path <dirs> <files>
```

注: Pythonコードを変換する場合は、すべてのソースファイルを1回の呼び出しで一緒に指定してください。Fortify Static Code Analyzerでは、その後の呼び出し時にビルドIDに関連付けられたファイルリストに新しいファイルを追加する機能はサポートされていません。

Pythonコマンドラインオプション

次の表では、Pythonオプションについて説明します。

Pythonオプション	説明
`-python-version <version>`	スキャンするPythonソースコードのバージョンを指定します。*<version>*の有効な値は、2および3です。デフォルト値は3です。 **同等のプロパティ名:** `com.fortify.sca.PythonVersion`
`-python-no-auto-root-calculation`	モジュールおよびパッケージのインポートに使用する、すべてのプロジェクトソースファイルの共通ルートディレクトリの自動計算を無効にします。

Pythonオプション	説明
	同等のプロパティ名: `com.fortify.sca.PythonNoAutoRootCalculation`
`-python-path <dirs>`	追加のインポートディレクトリのセミコロン区切り(Windows)またはコロン区切り(Windows以外)リストを指定します。`-python-path`オプションを使用して、パッケージまたはモジュールのインポートに使用するパスを指定できます。このオプションを使用してネームスペースパッケージディレクトリのすべてのパスを含めてください。Fortify Static Code Analyzerでは、インポートされるファイルごとに指定されたパスを順番に検索し、最初に検出されたファイルを使用します。 **同等のプロパティ名:** `com.fortify.sca.PythonPath`
`-django-disable-autodiscover`	Fortify Static Code AnalyzerでDjangoテンプレートを自動検出しないことを指定します。 **同等のプロパティ名:** `com.fortify.sca.DjangoDisableAutodiscover`
`-django-template-dirs <dirs>`	Djangoテンプレートを含むディレクトリのセミコロン区切り(Windows)またはコロン区切り(Windows以外)リストを指定します。Fortify Static Code Analyzerでは、Djangoテンプレートファイルごとに指定されたパスを順番に検索し、最初に検出されたテンプレートファイルを使用します。 **同等のプロパティ名:** `com.fortify.sca.DjangoTemplateDirs`

参照情報

"Pythonのプロパティ" ページ204

Pythonのコマンドライン例

WindowsでPython 3コードを変換する場合:

```
sourceanalyzer -b Python3Proj -python-path
"C:\Python312\Lib;C:\Python312\Lib\site-packages" src/*.py
```

WindowsでPython 2コードを変換する場合:

```
sourceanalyzer -b MyPython2 -python-version 2 -python-path
"C:\Python27\Lib;C:\Python27\Lib\site-packages" src/*.py
```

Windows以外でPython 3コードを変換する場合:

```
sourceanalyzer -b Python3Proj -python-path
/usr/lib/python3.12:/usr/local/lib/python3.12/site-packages src/*.py
```

Windows以外でPython 2コードを変換する場合:

```
sourceanalyzer -b MyPython2 -python-version 2 -python-path
/usr/lib/python2.7:/usr/local/lib/python2.7/site-packages src/*.py
```

仮想環境でのPythonの変換

このセクションでは、仮想環境でPythonプロジェクトを変換する方法について説明します。プロジェクトのすべての依存関係が仮想環境にインストールされていることを確認してください。仮想環境でPythonプロジェクトを変換するには、プロジェクトの依存関係を指定する`-python-path`オプションを含めます。

Python仮想環境の例

仮想環境名が`myenv`で、プロジェクトの依存関係が`myenv/lib/python<version>/site-packages`ディレクトリにインストールされているPythonプロジェクトを変換するには、次のように入力します。

```
sourceanalyzer –b mybuild -python-path "myenv/lib/python<version>/site-
packages/" myproject/
```

conda環境の例

conda環境名が`myenv`で、プロジェクトの依存関係が`<conda_install_dir>/envs/myenv/lib/python<version>/site-packages`ディレクトリにインストールされているPythonプロジェクトを変換するには、次のように入力します。

```
sourceanalyzer –b mybuild -python-path "<conda_install_
dir>/envs/myenv/lib/python<version>/site-packages/" myproject/
```

インポート済みのモジュールとパッケージを含める

Pythonアプリケーションを変換してスキャンに備えるため、Fortify Static Code Analyzerではアプリケーションが使用するインポート済みのモジュールおよびパッケージを検索します。Fortify Static Code Analyzerでは、Pythonランタイムシステムがインポート済みのモジュールとパッケージを検索するために使用する`PYTHONPATH`環境変数を考慮しません。

Fortify Static Code Analyzerでは、次の順序でディレクトリのリストを使用して、インポート済みのモジュールとパッケージを検索します。

1. すべてのプロジェクトソースファイルの共通ルートディレクトリ。Fortify Static Code Analyzerによって自動的に計算されます。たとえば、2つのプロジェクトディレクトリ`PrimaryDir/project1/*`および`PrimaryDir/project2/*`が存在する場合、共通ルートディレクトリは`PrimaryDir`です。インポート済みモジュールおよびパッケージの検索ターゲットである共通ルートディレクトリを削除するには、変換コマンドに`-python-no-auto-root-calculation`オプションを含める必要があります。
2. `-python-path`オプションで指定されたディレクトリ。
   Fortify Static Code Analyzerでは、標準Pythonライブラリのモジュールのサブセットを変換に含めます(「builtins」モジュール、もともとCで記述されたすべてのモジュールなど)。Fortify Static Code Analyzerでは最初にFortify Static Code Analyzerに含まれるセットで標準Pythonライブラリモジュールを検索し、次に`-python-path`オプションで指定されたパスで検索します。Fortify Static Code Analyzerで見つからないモジュールをPythonコードでインポートすると、警告が表示されます。標準Pythonライブラリのすべてのモジュールが見つかるようにするには、`-python-path`リストで標準Pythonライブラリへのパスを追加します。
3. Fortify Static Code Analyzerが変換中のファイルを含む現在のディレクトリ。たとえば、Fortify Static Code Analyzerで`PrimaryDir/project1/a.py`を変換すると、インポート済みモジュールおよびパッケージを検索する最後のディレクトリとしてディレクトリ`PrimaryDir/project1`が追加されます。

ネームスペースパッケージを含める

ネームスペースパッケージを変換するには、`-python-path`オプションを使用してネームスペースパッケージディレクトリへのすべてのパスを含める必要があります。たとえば、複数のフォルダにネームスペースパッケージ`package_name`の2つのサブパッケージが含まれている場合は:

```
/path_1/package_name/subpackageA
/path_2/package_name/subpackageB
```

Sourceanalyzerコマンドラインに、`-python-path`オプションとともに`/path_1;/path_2`を含めます。

Djangoの変換

Fortify Static Code Analyzerでは、Djangoフレームワークをサポートしています。Djangoフレームワークを使用して作成されたコードを変換するには、*<sca_install_dir>*`/Core/config/fortify-sca.properties`設定ファイルに次のプロパティを追加します。

```
com.fortify.sca.limiters.MaxPassthroughChainDepth=8

com.fortify.sca.limiters.MaxChainDepth=8
```

デフォルトでは、Fortify Static Code Analyzerはプロジェクトのルートディレクトリ内のDjangoテンプレートの検出を試みます。検出されたDjangoテンプレートは、自動的に変換に追加されます。Fortify Static Code AnalyzerでDjangoテンプレートを自動的に検出しないようにする場合は、`-django-disable-`

autodiscoverオプションを使用します。プロジェクトにDjangoテンプレートが必要なのに、Djangoテンプレートが予期しない場所にあるようにプロジェクトが設定されている場合は、-django-disable-autodiscoverオプションに加えて-django-template-dirsオプションを使用してテンプレートを含むディレクトリを指定します。

sourceanalyzerコマンドに-django-template-dirsオプションを追加して、Djangoテンプレートファイルの追加の場所を指定できます。

```
-django-template-dirs <dirs>
```

第10章: モバイルプラットフォームのコードの変換

Fortify Static Code Analyzerでは、次のモバイルアプリケーションソース言語の分析をサポートしています。

- Xcodeを使用して開発されるiOSアプリケーション用のSwift、Objective-C、およびObjective-C++
- Androidアプリケーション用のJava

Xamarinアプリケーションの変換の詳細については、"Visual Studioプロジェクトの変換" ページ66を参照してください。

このセクションでは、次のトピックについて説明します。

Apple iOSプロジェクトの変換

このセクションでは、iOSアプリケーションのSwift、Objective-C、およびObjective-C++ソースコードを変換する方法について説明します。プロジェクトのソースファイルを識別するために、Fortify Static Code Analyzerは自動的にXcodeコマンドラインツールのXcodebuildに統合されます。

iOSプロジェクト変換の前提条件

iOSプロジェクトを変換するための前提条件は次のとおりです。

- Objective-C++プロジェクトは、非脆弱なObjective-Cランタイム(ABIバージョン2または3)を使用する必要があります。
- Appleの`xcode-select`コマンドラインツールを使用して、Xcodeパスを設定します。Fortify Static Code Analyzerでは、システムグローバルなXcode設定を使用して、Xcodeツールチェーンとヘッダを検索します。
- 正常なXcodeビルドに必要なすべてのソースファイルが提供されていることを確認します。

  `-exclude`オプションを使用して、分析からファイルを除外できます("iOSコード分析のコマンドライン構文" 次のページを参照)。
- プロジェクトをビルドするために必要な依存関係が用意されている必要があります。
- Swiftコードを変換するには、CocoaPodsを含むすべてのサードパーティモジュールを使用できることを確認します。ブリッジヘッダも使用可能である必要があります。ただしXcodeでは、通常、ビルド中にこれらのファイルを自動的に生成します。

- プロジェクトにバイナリ形式のプロパティリストファイルが含まれる場合は、先にファイルをXML形式に変換する必要があります。これを行うには、Xcodeの`putil`コマンドを使用できます。
- Objective-Cプロジェクトを変換するには、サードパーティライブラリのヘッダが使用可能な必要があります。
- WatchKitアプリケーションを変換するには、iPhoneアプリケーションターゲットとWatchKit Extensionターゲットの両方を変換してください。

iOSコード分析のコマンドライン構文

次のコマンドライン構文を使用して、iOSコードを変換できます。

```
sourceanalyzer -b <build_id> xcodebuild [<compiler_options>]
```

ここで、*<compiler_options>*はXcodeコンパイラに渡される、サポートされているオプションです。何らかの*<compiler_options>*を指定した`build`オプションを含める必要があります。Fortify Static Code Analyzer Xcodebuildの統合では、`xcodebuild archive`などの代替ビルドコマンドの出力形式はサポートされていません。

注: このコマンドを実行すると、xcodebuildによってソースコードがコンパイルされます。

分析からファイルを除外するには、`-exclude`オプションを使用します("変換オプション" ページ136を参照)。ファイルがXcodeのビルドに含まれている場合でも、除外指定に一致するすべてのソースファイルが変換されません。次に例を示します。

```
sourceanalyzer -b MyProject -exclude "**/TestFile.swift" xcodebuild clean
build
```

アプリケーションがプロパティリストファイル(たとえば*<file>*`.plist`)を使用している場合は、これらのファイルを別の`sourceanalyzer`コマンドで変換します。プロジェクトファイルの変換に使用したものと同じビルドIDを使用します。次に例を示します。

```
sourceanalyzer -b MyProject <path_to_plist_files>
```

プロジェクトでCocoaPodsを使用している場合は、`-workspace`を含めてプロジェクトをビルドします。例:

```
sourceanalyzer -b DemoAppSwift xcodebuild clean build -workspace
DemoAppSwift.xcworkspace -scheme DemoAppSwift -sdk iphonesimulator
```

変換が完了したら、次の例に示すように、分析フェーズを実行して結果をFPRファイルに保存できます。

```
sourceanalyzer -b DemoAppSwift -scan -f MyResults.fpr
```

Androidプロジェクトの変換

このセクションでは、Androidアプリケーションの Javaソースコードを変換する方法について説明します。Fortify Static Code Analyzerを使用して、次のいずれからでもGradleでコードをスキャンできます。

- オペレーティングシステムのコマンドライン
- Android Studioで実行中のターミナルウィンドウ

Gradleの使い方はどちらの方法でも同じです。

注: Fortify Analysis Plugin for IntelliJ IDEA and Android Studioを使用すると、Android Studioから Androidコードを直接スキャンすることもできます。詳細については、*OpenText™ Fortify Analysis Plugin for IntelliJ IDEA and Android Studioユーザガイド*を参照してください。

Androidプロジェクト変換の前提条件

Androidプロジェクトを変換するための前提条件は次のとおりです。

- Android Studioと関連するAndroid SDKは、スキャンを実行するシステムにインストールされます。
- Androidプロジェクトでは、ビルドにGradleを使用します。

  Gradleを使用しない古いプロジェクトがある場合は、関連付けられているAndroid Studioプロジェクトに Gradleサポートを追加する必要があります。

  Androidプロジェクトの作成に使用するバージョンのAndroid Studioに付属するGradleと同じバージョンを使用します。
- アプリケーションのプロジェクトのAndroidコードをビルドするために必要なすべての依存関係が用意されていることを確認します。
- Android Studio内に表示されないコマンドウィンドウからAndroidコードを変換するには、Gradle Wrapper (`gradlew`)がシステムパスで定義されている必要があります。

Androidコード分析のコマンドライン構文

Androidプロジェクトをスキャンするには、gradlewを使用します。これは、Gradle Wrapperを使用する点以外はGradleを使用することに似ています。Gradle Wrapperを使用してAndroidプロジェクトを変換する方法については、"Gradle統合の使用" ページ128を参照してください。

Androidレイアウトファイルで検出されたフィルタリングの問題

Androidプロジェクトにレイアウトファイル(ユーザインタフェースの設計に使用)が含まれている場合、プロジェクトファイルにはAndroid Studioによって自動的に生成される`R.java`ソースファイルが含まれている可能性があります。プロジェクトをスキャンすると、Fortify Static Code Analyzerでこれらのレイアウトファイルに関連する問題を検出できます。

Fortifyでは、レイアウトファイルでレポートされた問題を標準的な監査に含めて、これらの問題が偽陽性かどうかを慎重に判断できるようにすることをお勧めします。関心がないレイアウトファイルで問題を特定した後は、"分析のフィルタリング" ページ180の説明に従って、問題をフィルタできます。インスタンスIDに基づいて問題をフィルタできます。

第11章: Goコードの変換

このセクションでは、Goコードを変換する方法について説明します。Fortify Static Code Analyzerでは、Windows、Linux、およびmacOS上のGoコードの分析をサポートしています。

このセクションでは、次のトピックについて説明します。

Goコマンドライン構文

最善の結果を得るには、プロジェクトがコンパイル可能で、必要なすべての依存関係を利用できる必要があります。

次のエンティティは、変換(およびスキャン)から除外されます。

- ベンダフォルダ
- サブフォルダ内の任意の`go.mod`ファイルによって定義されたプロジェクト(%PROJECT_ROOT%の下にある`go.mod`ファイルによって定義されたプロジェクトを除く)
- `_test.go`サフィックスが付くすべてのファイル(ユニットテスト)

Goコードを変換する基本的なコマンドライン構文は次のとおりです。

```
sourceanalyzer -b <build_id> [-gopath <dir>] [-goroot <dir>] <files>
```

Goコマンドラインオプション

次の表では、Goコード変換専用のコマンドラインオプションについて説明します。

Goオプション	説明
`-gopath <dir>`	Goプロジェクトの変換に使用するGOPATH環境変数の値を指定します。このオプションを指定しない場合、Fortify Static Code AnalyzerではGOPATHシステム環境変数の既存の値が使用されます。 gopathディレクトリは絶対パスとして指定する必要があります。次の例は、*<dir>*に有効な値です。

Goオプション	説明

	`/home/projects/go_workspace/my_proj`
	`C:\projects\go_workspace\my_proj`
	
	次の例は、*<dir>*に無効な値です。
	
	`go_workspace/my_proj`
	
	このオプションとGOPATHシステム環境変数が設定されていない場合、gopathのデフォルトは、ユーザのホームディレクトリにあるgoという名前のサブディレクトリです(Linuxの場合は`$HOME/go`、Windowsの場合は`%USERPROFILE%\go`)。ただし、そのディレクトリにGoディストリビューションが含まれている場合を除きます。
	
	モジュールを使用する場合、goコンパイラコマンドのドキュメントで説明されているように、GOPATH環境変数でパッケージのインポートを解決する必要はありません。ただし、不足しているモジュールの依存関係をダウンロードするときに使用する出力ディレクトリは、引き続きGOPATHによって決定されます。
	
	注: Fortify Static Code Analyzerでは、パッケージのインポートを解決するためにGOPATH環境変数のみに依存する古いGoプロジェクトを完全にはサポートしていません。
	
	同等のプロパティ名
	`com.fortify.sca.GOPATH`

`-goroot <dir>`	Goインストールの場所を指定します。このオプションを指定しない場合は、GOROOTシステム環境変数が使用されます。 このオプションが指定されず、GOROOTシステム環境変数が設定されていない場合、Fortify Static Code AnalyzerではFortify Static Code Analyzerインストールに含まれるGoコンパイラを使用します。 **同等のプロパティ名** `com.fortify.sca.GOROOT`
`-goproxy <url>`	1つ以上のプロキシURLをカンマ区切りで指定します。また、`direct`または`off`(ネットワークの使用を無効化)を指定することもできます。 このオプションが指定されず、GOPROXYシステム環境変数が設定されていない場合、Fortify Static Code Analyzerでは`https://proxy.golang.org,direct`を使用します。

Goオプション	説明
	同等のプロパティ名 `com.fortify.sca.GOPROXY`

参照情報

"Goのプロパティ" ページ205

依存関係の解決

Fortify Static Code Analyzerでは、Goに組み込む2つの依存関係管理システムをサポートしています。

- モジュール

  Fortify Static Code Analyzerでは、ネイティブのGoツールチェーンを使用して必要なすべての依存関係をダウンロードします。Fortify Static Code Analyzerを実行するコンピュータでインターネットへのアクセスが制限されている場合は、次のいずれかを実行します。

  - Artifactoryなどのアーティファクト管理システムを使用している場合は、GOPROXY環境変数を設定するか、"Goコマンドラインオプション" ページ89で説明されている`-goproxy`オプションを使用します。
  - モジュールとベンダを使用して、必要なすべての依存関係をダウンロードします。

- GOPATHの依存関係の解決

  depなどのサードパーティの依存関係管理システムを使用している場合は、変換を開始する前に、すべての依存関係をダウンロードする必要があります。

第12章: DartおよびFlutterコードの変換

このセクションでは、DartおよびFlutterコードを変換する方法について説明します。Fortify Static Code Analyzerでは、WindowsおよびLinux上のDartおよびFlutterコードの分析をサポートしています。

このセクションでは、次のトピックについて説明します。

DartおよびFlutter変換の前提条件

Dartプロジェクトと Flutterプロジェクトを変換するための前提条件は次のとおりです。

- サポートされているDart SDK (Dart専用プロジェクトの場合)およびFlutter SDK (Flutterプロジェクトの場合)がシステムにインストールされている必要があります。サポートされているDartおよびFlutter SDKのバージョンについては、『*Fortify ソフトウェアシステム要件*』を参照してください。
- 次のいずれかのコマンドを実行して、プロジェクトの依存関係をダウンロードします。
  - Flutterプロジェクトの場合は、`flutter pub get`を使用します。
  - Dart専用プロジェクトの場合は、`dart pub get`を使用します。

  たとえば、プロジェクトルートが`myproject`のFlutterプロジェクトの依存関係をダウンロードするには、次のコマンドを実行します。

```
cd myproject

flutter pub get
```

重要 プロジェクトに、`pubspec.yaml`ファイルが異なるネストされたパッケージが含まれている場合は、パッケージルートごとに`dart pub get`または`flutter pub get`を実行する必要があります。

重要 プロジェクトディレクトリに次のものが含まれている必要があります。

- `pubspec.yaml`ファイル。このファイルは依存関係を指定します。
- `.dart_tool`ディレクトリ。このディレクトリには、`pub`ツールによって自動的に生成される`package_config.json`ファイルが含まれます。

DartおよびFlutterのコマンドライン構文

DartおよびFlutterのコードを変換する基本的なコマンドライン構文は次のとおりです。

```
sourceanalyzer -b <build_id> <translation_options> <dirs>

sourceanalyzer -b <build_id> <translation_options> <files>
```

DartおよびFlutterのコマンドライン例

my_appプロジェクトルートディレクトリを使用してDartプロジェクトまたはFlutterプロジェクトを変換するには

```
sourceanalyzer -b myProject my_app/
```

my_appプロジェクトルートディレクトリ内のa_widget.dartファイルを変換するには、次のコマンドを使用します。

```
sourceanalyzer -b myProject my_app/a_widget.dart
```

my_dart_projディレクトリ内のすべてのdartソースファイルを変換するには

```
sourceanalyzer -b myProject "my_dart_proj/**/*.dart"
```

第13章: Rubyコードの変換

このセクションでは、次のトピックについて説明します。

Rubyコマンドライン構文

Rubyコードを変換する基本的なコマンドライン構文は次のとおりです。

```
sourceanalyzer -b <build_id> <file>
```

ここで、*<file>*はスキャンするRubyファイルの名前です。複数のRubyファイルを含めるには、次の例に示すように、ファイルをスペースで区切ります。

```
sourceanalyzer -b <build_id> file1.rb file2.rb file3.rb
```

個々のRubyファイルをリストするだけでなく、アスタリスク(*)のワイルドカードを使用して、指定したディレクトリ内のすべてのRubyファイルを選択できます。たとえば、`src`ディレクトリ内のすべてのRubyファイルを検索するには、次の`sourceanalyzer`コマンドを使用します。

```
sourceanalyzer -b <build_id> src/*.rb
```

注: Rubyコードを変換する場合は、すべてのソースファイルを1回の呼び出しで一緒に指定してください。Fortify Static Code Analyzerでは、その後の呼び出し時にビルドIDに関連付けられたファイルリストに新しいファイルを追加する機能はサポートされていません。

Rubyコマンドラインオプション

次の表は、Rubyの変換オプションについて説明しています。

Rubyオプション	説明
`-ruby-path <dirs>`	Rubyライブラリを含むディレクトリへのパスを1つ以上指定します("ライブラリの追加" 次のページを参照)。 **同等のプロパティ名:** `com.fortify.sca.RubyLibraryPaths`

Rubyオプション	説明
`-rubygem-path <dirs>`	RubyGemsの場所へのパスを指定します("Gemパスの追加" 下を参照)。 **同等のプロパティ名:** `com.fortify.sca.RubyGemPaths`

参照情報

"Rubyのプロパティ" ページ205

ライブラリの追加

Rubyソースコードに特定のライブラリが必要な場合は、Rubyライブラリを`sourceanalyzer`コマンドに追加します。Rubyのgemとともにインストールされるすべての Rubyライブラリを含めます。たとえば、`/usr/share/ruby/myPersonalLibrary`ディレクトリ内に`utils.rb`ファイルがある場合は、次のオプションを`sourceanalyzer`コマンドに追加します。

```
-ruby-path /usr/share/ruby/myPersonalLibrary
```

複数のライブラリはセミコロン(Windows)またはコロン(Windows以外)で区切ります。次に、Windows以外のシステムでのオプションの例を示します。

```
-ruby-path /path/one:/path/two:/path/three
```

Gemパスの追加

Rubyのすべてのgemとその依存関係パスを追加するには、Rubyのすべてのgemをインポートします。Rubyのgemパスを取得するには、`gem env`コマンドを実行します。**GEM PATHS**の下で次のようなディレクトリを探します。

```
/home/myUser/gems/ruby-version
```

このディレクトリには、システムにインストールされているすべてのgemファイルのディレクトリが含まれる`gems`という名前の別のディレクトリがあります。この例では、コマンドラインで次のコマンドを使用します。

```
-rubygem-path /home/myUser/gems/ruby-version/gems
```

複数の`gems`ディレクトリがある場合は、次のようにセミコロン(Windows)またはコロン(Windows以外)で区切ります。

```
-rubygem-path /path/to/gems:/another/path/to/more/gems
```

注: Windowsシステムの場合は、`gems`ディレクトリをセミコロンで区切ります。

第14章: COBOLコードの変換

COBOL変換はWindowsシステムでのみ実行され、最新のCOBOL方言をサポートします。または、レガシーCOBOL変換を使用できます(「"レガシーCOBOL変換の使用" ページ99」を参照)。

COBOLコードの変換でサポートされている技術のリストについては、*Fortifyソフトウェアシステム要件*ドキュメントを参照してください。Fortify Static Code Analyzerは、現時点ではCOBOLアプリケーションのカスタムルールをサポートしていません。

注: Fortify Static Code AnalyzerでCOBOLをスキャンするには、COBOLスキャン機能を明確に含むFortify Static Code Analyzerライセンスファイルが必要です。COBOLのスキャンと必要なライセンスファイルについて詳しくは、カスタマサポートにお問い合わせください。

このセクションでは、次のトピックについて説明します。

COBOLソースファイルとコピーブックファイルの変換準備

COBOLプログラムを分析する前に、Fortify Static Code Analyzerを実行するWindowsシステムに次のプログラムコンポーネントをコピーする必要があります。

- COBOLソースコード

  Fortifyでは、COBOLソースコードファイルに拡張子.CBL、.COB、.cbl、または.cobを使用することを推奨します。ソースコードファイルに拡張子がないか、拡張子が標準以外のものである場合は、「"ファイル拡張子を持たないCOBOLソースファイルの変換" 次のページ」および「"任意のファイル拡張子を持つCOBOLソースファイルの変換" 次のページ」の手順に従う必要があります。
- COBOLソースコードで使用しているすべてのコピーブックファイル

  これには、COBOLソースコードで参照するすべてのSQL INCLUDEファイル(SQL INCLUDEファイルは技術的にはコピーブックファイル)が含まれます。

  重要 コピーブックファイルには、拡張子.CPYまたは.cpyを使用する必要があります。

  COBOLソースコードに次が含まれている場合:

  ```
  COPY FOO
  ```

  または

  ```
  EXEC SQL INCLUDE FOO END-EXEC
  ```

  FOOは、COBOLコピーブックの名前で、対応するコピーブックファイルは名前がFOO.CPYまたはFOO.cpyになります。

Fortifyでは、COBOLソースコードファイルをsourcesという名前のディレクトリに、コピーブックファイルをcopybooksという名前のディレクトリに配置することを推奨します。これらのディレクトリは同じレベルで作成してください。

COBOLのコマンドライン構文

1つのCOBOLソースコードファイルを変換するために使用される基本的な構文は次のとおりです。

```
sourceanalyzer -b <build_id> <path>
```

変換されたCOBOLプログラムをスキャンして、分析結果をFPRファイルに保存するために使用される基本的な構文は次のとおりです。

```
sourceanalyzer -b <build_id> -scan -f <results>.fpr
```

参照情報

"ファイルとディレクトリの指定" ページ149

ファイル拡張子を持たないCOBOLソースファイルの変換

メインフレームからCOBOLソースファイル(コピーブックファイルではなく)を取得し、これに.COBや.CBL_のファイル拡張子がない場合(COBOLファイル名としては一般的です)、変換コマンドラインに次の項目を含める必要があります。

```
-noextension-type COBOL
```

次のコマンドの例では、ファイル拡張子を持たないCOBOLソースコードを変換します。

```
sourceanalyzer -b MyProject -noextension-type COBOL -copydirs copybooks
sources
```

任意のファイル拡張子を持つCOBOLソースファイルの変換

任意の拡張子(.xyz)を持つCOBOLソースファイルがある場合は、変換コマンドラインに次の項目を含める必要があります。

```
-Dcom.fortify.sca.fileextensions.xyz=COBOL
```

ファイルまたはディレクトリ指定子を使用する場合は、式*.xyzも含める必要があります("ファイルとディレクトリの指定" ページ149を参照)。

COBOLコマンドラインオプション

次の表では、COBOLコマンドラインオプションについて説明します。レガシーCOBOL変換を使用するには、「"レガシーCOBOL変換コマンドラインオプション" 次のページ」を参照してください。

COBOLオプション	説明
-copydirs *<dirs>*	Fortify Static Code Analyzerでコピーブックファイルを検索する、1つ以上のディレクトリをセミコロン区切りで指定します。 **同等のプロパティ名:** com.fortify.sca.CobolCopyDirs
-dialect *<dialect>*	COBOLの方言を指定します。*<dialect>*の有効な値は、COBOL390およびMICROFOCUSです。dialect値では、大文字と小文字を区別しません。デフォルト値はCOBOL390です。 **同等のプロパティ名:** com.fortify.sca.CobolDialect
-checker-	1つ以上のCOBOLチェッカディレクティブをセミコロン区切りで指定します。

COBOLオプション	説明
`directives <directives>`	**注:** このオプションは、OpenText™ Server Expressの上級ユーザ向けです。 **同等のプロパティ名:** `com.fortify.sca.CobolCheckerDirectives`

レガシーCOBOL変換の使用

次のいずれかの条件に当てはまる場合は、レガシーCOBOL変換を使用してください。

- Windows以外のオペレーティングシステムでFortify Static Code Analyzerを実行している。
  サポートされているWindows以外のプラットフォームおよびアーキテクチャについては、『*Fortifyソフトウェアシステム要件*』を参照してください。
- 使用するCOBOL方言が、デフォルトのCOBOL変換でサポートされているものとは異なる(「"COBOLコマンドラインオプション" 前のページ」の`-dialect`オプションを参照)。

「"COBOLソースファイルとコピーブックファイルの変換準備" ページ97」の説明に従ってCOBOLソースコードファイルとコピーブックファイルを準備し、「"COBOLのコマンドライン構文" ページ97」で説明されているコマンドライン構文を使用します。レガシーCOBOL変換では、ファイル拡張子の有無に関係なく、コピーブックファイルを受け入れることにご注意ください。コピーブックファイルにファイル拡張子がある場合は、`-copy-extensions`コマンドラインオプションを使用します(「"レガシーCOBOL変換コマンドラインオプション" 下」を参照)。

レガシーCOBOL変換コマンドラインオプション

次の表では、レガシーCOBOL変換のコマンドラインオプションについて説明します。

レガシーCOBOLオプション	説明
`-cobol-legacy`	レガシーCOBOL変換を使用したCOBOLコードの変換を指定します。このオプションは、レガシーCOBOL変換を有効にする場合に必要です。 **同等のプロパティ名:** `com.fortify.sca.CobolLegacy`
`-copydirs <dirs>`	Fortify Static Code Analyzerでコピーブックファイルを検索する、1つ以上のディレクトリをセミコロン区切りまたはコロン区切りで指定します。 **同等のプロパティ名:** `com.fortify.sca.CobolCopyDirs`

レガシーCOBOLオプション	説明
`-copy-extensions <ext>`	1つ以上のコピーブックファイル拡張子をセミコロン区切りまたはコロン区切りで指定します。 **同等のプロパティ名:** `com.fortify.sca.CobolCopyExtensions`
`-fixed-format`	すべてのコード行のカラム8 ~ 72の間のソースコードのみを検索するようFortify Static Code Analyzerに指示する、固定形式のCOBOLを指定します。デフォルトはフリーフォーマットです。 IBM Enterprise COBOLコードは通常、固定形式です。以下の場合は、`-fixed-format`オプションが必要になる可能性があります。 • COBOL変換が無期限にハングする可能性がある • Fortify Static Code AnalyzerによりCOBOL変換で多数の解析エラーが報告される **同等のプロパティ名:** `com.fortify.sca.CobolFixedFormat`

第15章: Salesforce ApexおよびVisualforceコードの変換

このセクションでは、次のトピックについて説明します。

ApexおよびVisualforce変換の前提条件

Apexプロジェクトや Visualforceプロジェクトを変換するには、スキャンするすべてのソースコードが、Fortify Static Code Analyzerをインストールしたのと同じマシン上で使用可能である必要があります。

カスタムSalesforceアプリをスキャンするには、そのアプリを開発して展開したSalesforce組織(org)からローカルコンピュータにダウンロードします。ダウンロードしたアプリは次の要素で構成されています。

- `.cls`拡張子を持つファイル内のApexクラス
- `.page`拡張子を持つファイル内のVisualforce Webページ
- `.trigger`拡張子を持つファイルに含まれている、データベースの「trigger」関数と呼ばれるApexコードファイル
- `.component`拡張子を持つVisualforceコンポーネントファイル
- `.object`拡張子を持つオブジェクト

Salesforce Webサイトで入手できるAnt移行ツールを使用して、Salesforceクラウド内の組織からローカルコンピュータにアプリをダウンロードします。プロジェクトマニフェストファイルが、`build.xml`ファイル内の指定したターゲットに対して正しく設定されていることを確認します。たとえば、次の`package.xml`マニフェストファイルは、Fortify Static Code Analyzerに、すべてのクラス、カスタムオブジェクト、ページ、およびコンポーネントを提供します。

```
<?xml version="1.0" encoding="UTF-8"?> <Package
xmlns
=
http://soap.sforce.com/2006/04/metadata>

<types>

<members>
*
</members>

<name>

</name>

</types>

<types>
<members>

</members>
```

```
*
</members>

<name>
ApexPage
</name>

</types>

<types>

<members>
*
</members>

<name>
ApexComponent
</name>

</types>
 <types> <members>*</members> <name>CustomObject</name> </types> <version>55.0</version> </Package>
```

Ant移行ツールのドキュメントを使用して、取得ターゲットを設定します。AppExchangeのアプリを組織で使用している場合、それらのアプリはパッケージ化されたターゲットとしてダウンロードされる必要があります。

ApexおよびVisualforceのコマンドライン構文

ApexおよびVisualforceのコードを変換する基本的なコマンドライン構文は次のとおりです。

```
sourceanalyzer -b <build_id> <files>
```

ここで、*<files>*はApexまたはVisualforceのファイルまたはソースファイルへのパスです。

重要 ソースファイルでサポートされているファイル拡張子は、.cls、.component、.trigger、.object、および.pageです。

すべてのApexおよびVisualforce固有のコマンドラインオプションについて詳しくは、「ApexおよびVisualforceのコマンドラインオプション」を参照してください。

第16章: その他の言語および設定の変換

このセクションでは、次のトピックについて説明します。

Solidityコードの分析

次の例に、Solidityコードを分析するための基本的なコマンドライン構文を示します。

```
sourceanalyzer -b <build_id> <files>

sourceanalyzer -b <build_id> -scan -f <results>.fpr
```

依存関係のインポート

Fortify Static Code Analyzer変換では、相対パスと絶対パスで指定したファイルのインポートステートメントのみがサポートされます。ライブラリのインポートステートメントはサポートされません。

コンパイラのバージョンの管理

Fortify Static Code Analyzerは、pragmaステートメントを使用してコード内で参照されているコンパイラを、Solidityコンパイラリポジトリからダウンロードします。デフォルトでは、Fortify Static Code Analyzerは

Solidityコンパイラを${flight.workdir}/solidityにダウンロードします。

ファイルにpragmaステートメントが含まれていない場合は、^0.8.0の既定値が使用されます。分析で使用する別のデフォルトコンパイラバージョンを指定するには、コマンドラインにflight.solidity.defaultCompilerVersionプロパティを含めることができます。指定するバージョンは、Solidityコンパイラリポジトリに存在する必要があります。例:

```
sourceanalyzer -b MyProject ./

sourceanalyzer -b MyProject -scan -
Dflight.solidity.defaultCompilerVersion=0.8.16 -f MyResults.fpr
```

Solidityコンパイラをダウンロードするための接続にプロキシが必要な場合は、-Dhttps.proxyHostおよび-Dhttps.proxyPortを使用してプロキシ情報を含めてください。例:

```
sourceanalyzer -b MyProject ./

sourceanalyzer -b MyProject -scan -Dhttps.proxyHost=MyProxyHost -
Dhttps.proxyPort=1234 -f MyResults.fpr
```

flight.solidity.defaultCompilerVersionは、fortify-sca.propertiesファイルに追加できます。

参照情報

"Fortify Static Code Analyzerのプロパティファイル" ページ185

PHPコードの変換

MyPHP.phpという名前の単一のPHPファイルを変換する構文を次の例に示します。

```
sourceanalyzer -b <build_id> MyPHP.php
```

ソースまたはphp.iniファイルエントリに相対パス名が含まれる(./または../で始まる)ファイルを変換するには、次の例に示すようにPHPソースルートを設定します。

```
sourceanalyzer -php-source-root <path> -b <build_id> MyPHP.php
```

-php-source-rootオプションの詳細については、「"PHPコマンドラインオプション" 次のページ」の説明を参照してください。

注: PHPコードを変換する場合は、すべてのソースファイルを1回の呼び出しで一緒に指定してください。Fortify Static Code Analyzerでは、その後の呼び出し時にビルドIDに関連付けられたファイルリストに新しいファイルを追加する機能はサポートされていません。

PHPコマンドラインオプション

次の表では、PHP固有のコマンドラインオプションについて説明します。

PHPオプション	説明
`-php-source-root <path>`	プロジェクトルートディレクトリへの絶対パスを指定します。相対パス名は最初にカレントディレクトリから展開されます。ファイルが見つからない場合は、指定したPHPソースルートディレクトリからパスが展開されます。 **同等のプロパティ名:** `com.fortify.sca.PHPSourceRoot`
`-php-version <version>`	PHPのバージョンを指定します。デフォルトのバージョンは8.2です。有効なバージョンのリストについては、*Fortify ソフトウェアシステム要件*のドキュメントを参照してください。 **同等のプロパティ名:** `com.fortify.sca.PHPVersion`

参照情報

"PHPのプロパティ" ページ207

ABAPコードの変換

ABAPコードの変換は、その他のオペレーティング言語コードの変換に似ています。ただし、SAPデータベースからコードを抽出してスキャン用のコードを作成する追加の手順が必要です。詳しくは、"移送依頼のインポート" 次のページを参照してください。このセクションは、SAPとABAPの基本を理解していることを前提としています。

ABAPコードを変換するために、Fortify ABAP Extractorプログラムはソースファイルをプレゼンテーションサーバにダウンロードし、必要に応じてFortify Static Code Analyzerを開始します。ファイルをローカルシステムにダウンロードしてオペレーティングシステムのコマンドを実行するには、アクセス許可を持つアカウントを使用する必要があります。

Extractorプログラムはオンラインで実行されるため、抽出するために選択されたソースファイルのボリュームが許容されるプロセス実行時間を超えた場合は、`max dialog work process time reached`例外を受信することがあります。これを回避するには、大規模なプロジェクトを連続する小さいExtractorタスクとしてダウンロードします。たとえば、プロジェクトが4つの異なるパッケージで構成されている場合、各パッケージを同じプロジェクトディレクトリに個別にダウンロードします。例外が頻繁に発生する場合は、SAP Basis管理者と協力して最大時間制限(`rdisp/max_wprun_time`)を増加させます。

ABAPからパッケージが抽出されたら、Fortify ABAP ExtractorはTDEVCからパッケージ名と一致する`parentcl`フィールドを持つすべてを抽出します。次に、TDEVCから抽出したものと同じ`parentcl`フィー

ルドを持つ残りのすべてをTDEVCから再帰的に抽出します TDEVCから抽出されるフィールドはdevclassです。

devclassの値はプログラム名のセットとして扱われ、指定できるプログラム名と同様に処理されます。

TRDIRからプログラムを抽出するには、名前フィールドを次のいずれかと比較します。

- 選択画面で指定したプログラム名
- TDEVCから抽出された値のリスト(パッケージが提供された場合)

TRDIRの行は名前フィールドに指定したプログラム名を含む行であり、式LIKEprogramnameを使用して行が抽出されます。

この最終的な名前のリストをREAD REPORTで使用して、SAPシステムからコードを取得します。このメソッドは、レコードに対して単にREPORTSだけでなく、クラスとメソッドも読み込みます。

READ REPORTを呼び出すたびに、ローカルシステムの一時フォルダにファイルが作成されます。このファイルセットに対してFortify Static Code Analyzerが変換とスキャンを行い、Fortify Audit Workbenchで開くことができるFPRファイルを生成します。

参照情報

"ABAPのプロパティ" ページ207

INCLUDEの処理

ソースコードがダウンロードされると、Fortify ABAP Extractorはソース内のINCLUDEステートメントを検出します。検出すると、分析のためにincludeのターゲットをローカルコンピュータにダウンロードします。

移送依頼のインポート

ABAPコードをスキャンするには、Fortify ABAP Extractorの移送依頼をSAPサーバにインポートする必要があります。Fortifyの移送依頼は*<sca_install_dir>*/Tools/SAP_Extractor.zipにあります。

Fortify ABAP Extractorのパッケージ(SAP_Extractor.zip)には次のファイルが含まれています。

- K900XXX.S9S(「XXX」はリリース番号)
- R900XXX.S9S(「XXX」はリリース番号)

これらのファイルが、ローカル移送ドメインの外部からSAPシステムにインポートする必要があるSAP移送依頼の構成要素です。SAP管理者または移送依頼をシステムにインストールする権限を持つ個人に、移送依頼をインポートしてもらってください。

S95ファイルには、プログラム、トランザクション(YSCA)、およびプログラムユーザインタフェースが含まれています。これらをシステムにインポートした後、SAPデータベースからコードを抽出してFortify Static Code Analyzerスキャン用のコードを作成できます。

インストールに関する注意事項

Fortify ABAP Extractorの移送依頼は、SAPリリース7.02、SPレベル0006が実行されているシステムでサポートされています。別のSAPバージョンを実行していて、移送依頼のインポートエラー(Install

release does not match the current version)が発生した場合は、移送依頼のインストールに失敗しています。

この問題を解決するには、次の手順を実行します。

1. 移送依頼のインポートを再実行します。
   [Import Transport Request]ダイアログボックスが開きます。
2. [**Options**]タブを選択します。
3. [**Ignore Invalid Component Version**]チェックボックスをオンにます。
4. インポート手順を完了します。

それでも問題が解決しない場合や、異なるテーブル構造を持つSAPバージョンがシステムで実行されている場合、Fortifyでは、Fortify Static Code AnalyzerがABAPコードをスキャンできるように、独自の技術を使用してABAPファイル構造をエクスポートすることをお勧めします。

Fortify Static Code Analyzerのお気に入りリストへの追加

Fortify Static Code Analyzerのお気に入りリストへの追加はオプションですが、追加すると、Fortify Static Code Analyzerスキャンにすばやくアクセスしてスキャンを開始できます。次の手順では、日常業務でユーザメニューを使用すると仮定しています。別のメニューから作業を行う場合は、使用するメニューにお気に入りリンクを追加します。Fortify Static Code Analyzerのエントリを作成する前に、SAPサーバが実行中であり、Webベースのクライアントの SAP Easy Accessエリアにアクセスしていることを確認してください。

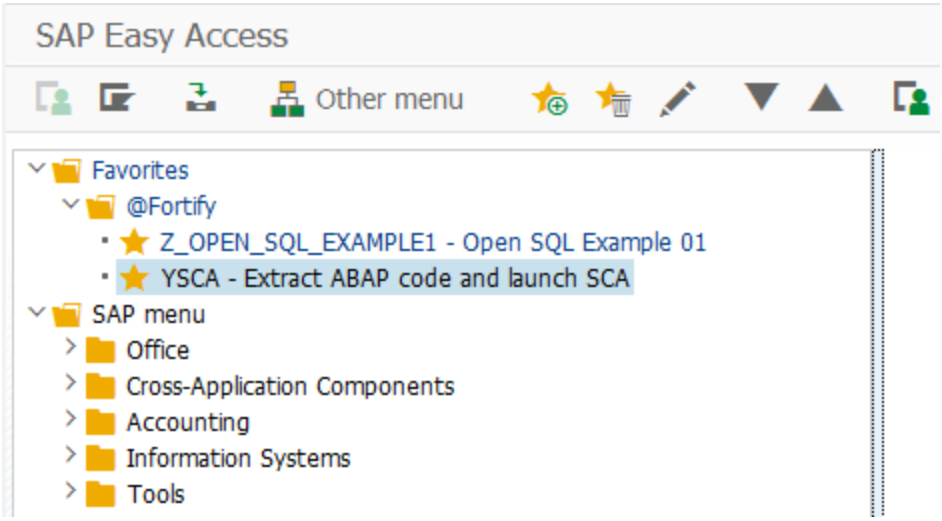
Fortify Static Code Analyzerをお気に入りリストに追加するには、次の手順を実行します。

1. [**SAP Easy Access**]メニューから、トランザクションボックスに「S000」と入力します。
   [**SAP Menu**]が開きます。
2. [**Favorites**]フォルダを右クリックし、[**Insert transaction**]を選択します。
   [**Manual entry of a transaction**]ダイアログボックスが開きます。
3. [**Transaction Code**]ボックスに「YSCA」と入力します。
4. 緑色のチェックマークアイコンをクリックします。
   [お気に入り(Favorites)]リストに [ABAPコードを抽出してSCAを起動する(Extract ABAP code and launch SCA)]という項目が表示されます。
5. [ABAPコードを抽出してSCAを起動する(Extract ABAP code and launch SCA)]リンクをクリックしてFortify ABAP Extractorを開始します。

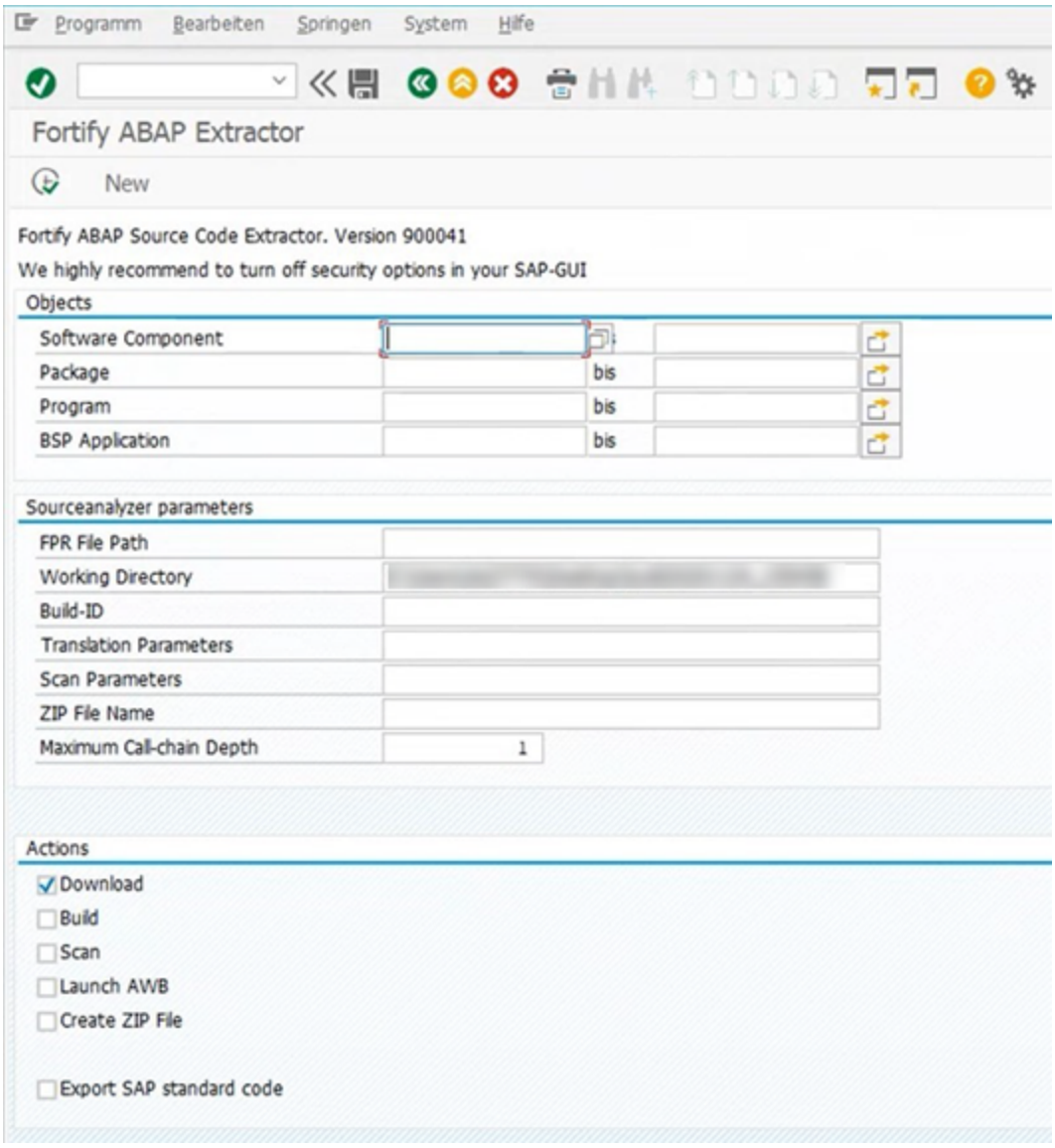
Fortify ABAP Extractorの実行

Fortify ABAP Extractorを実行するには、次の手順を実行します。

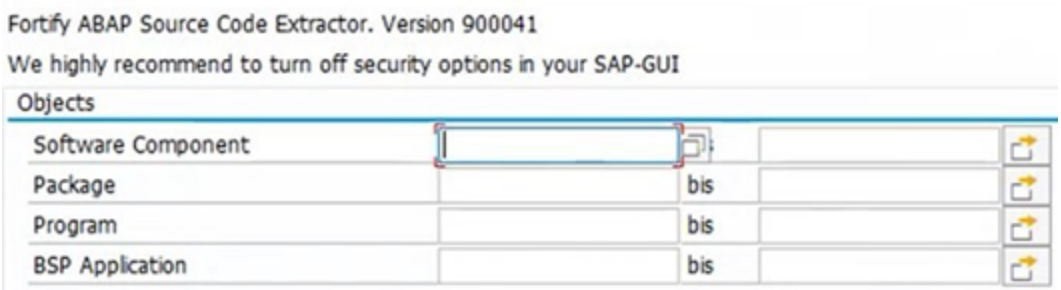
1. [お気に入り(Favorites)]リンクまたはトランザクションコードからFortify ABAP Extractorを起動するか、手動でExtractorオブジェクトを起動します。



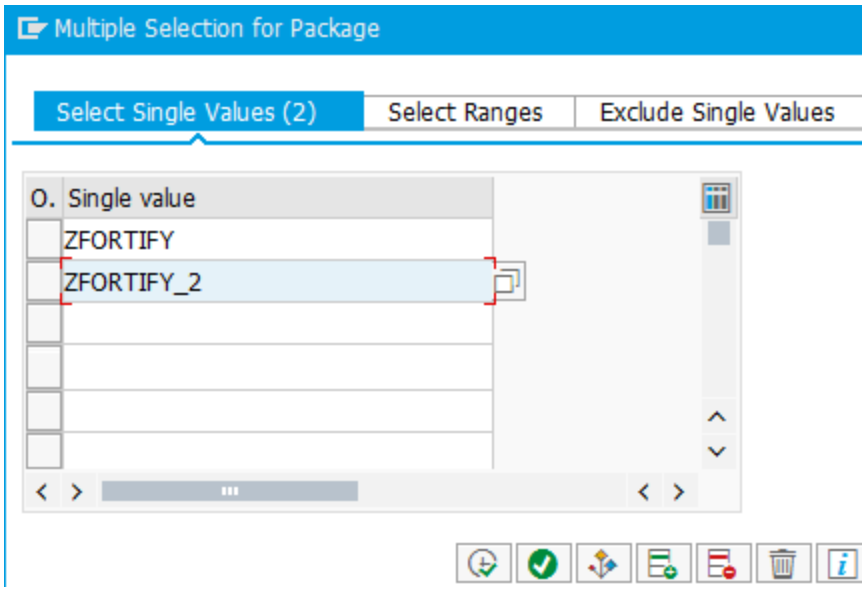
これにより、Fortify ABAP Extractorが開きます。



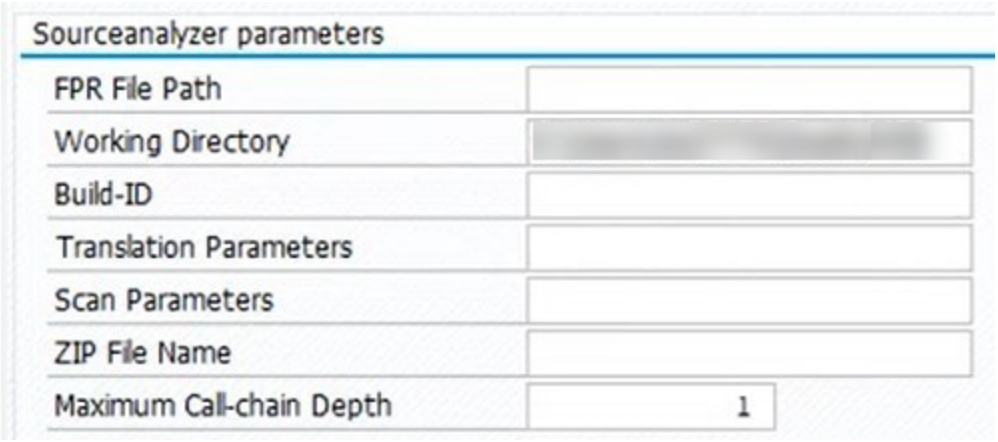
2. ダウンロードするコードを選択します。
   スキャンするソフトウェアコンポーネント、パッケージ、プログラム、またはBSPアプリケーションの範囲を開始名と終了名で指定します。



注: 複数のオブジェクトまたは範囲を指定できます。



3. 次の表に示すFortify Static Code Analyzer固有の情報を指定します。



フィールド	説明
FPR File Path	(オプション)スキャン結果ファイル(FPR)を保存するディレクトリを入力または選択します。FPRファイルの名前をパス名に含めてください。抽出プロセスを実行している同じコンピュータにダウンロードしたコードを自動的にスキャンするには、FPRファイルパスを指定する必要があります。
作業ディレクトリ(Working Directory)	抽出したソースコードを保存するディレクトリを入力または選択します。
Build-ID	(オプション)スキャンのビルドIDを入力します。Fortify Static Code Analyzerでは変換されたソースコードを識別するためにビルドIDを使用します。これはコード

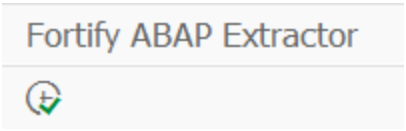
フィールド	説明
	をスキャンするために必要です。抽出プロセスを実行している同じコンピュータにダウンロードしたコードを自動的に変換するには、ビルドIDを指定する必要があります。
変換パラメータ(Translation Parameters)	(オプション)追加のFortify Static Code Analyzerコマンドライン変換オプションを入力します。抽出プロセスを実行している同じコンピュータにダウンロードしたコードを自動的に変換したり、変換オプションをカスタマイズしたりするには、変換オプションを指定する必要があります。
スキャンパラメータ(Scan Parameters)	(オプション)Fortify Static Code Analyzerコマンドラインスキャンオプションを入力します。抽出プロセスを実行している同じコンピュータにダウンロードしたコードを自動的にスキャンしたり、スキャンオプションをカスタマイズしたりするには、スキャンオプションを指定する必要があります。
ZIPファイル名(ZIP File Name)	(オプション)圧縮パッケージで出力する場合は、ZIPファイル名を入力します。
Maximum Call-chain Depth	グローバルSAP関数Fは、Fが明示的に選択されていない限り、または明示的に選択されたコードで始まる関数呼び出しのチェーンを介してFに到達可能で、そのチェーンの長さがこの数以下である場合を除き、ダウンロードされません。Fortifyでは、カスタマサポートの指示がない限り、2より大きい値を指定しないことを推奨します。

4. 次の表に示すアクション情報を指定します。

フィールド	説明
Download	SAPデータベースから抽出したソースコードをFortify Static Code Analyzerでダウンロードするには、 [ダウンロード(Download)]チェックボックスをオンにします。
ビルド(Build)	Fortify Static Code Analyzerでダウンロード済みのすべてのABAPコードを変換し、指定したビルドIDを使用してそのコードを保存するには、 [ビルド(Build)]チェックボックスをオンにします。このアクションを実行するには、Fortify ABAP Extractorを実行しているコンピュータにFortify Static Code Analyzerのインストール済みバージョンが必要です。多くの場合、Fortify Static Code Analyzerがインストールされているシステムにダウンロードしたソースコードを移動する方が簡単です。

フィールド	説明
スキャン(Scan)	指定したビルドIDのスキャンをFortify Static Code Analyzerで実行するには、[スキャン(Scan)]チェックボックスをオンにします。このアクションでは、変換(ビルド)アクションが以前に実行されている必要があります。このアクションを実行するには、Fortify ABAP Extractorを実行しているコンピュータにFortify Static Code Analyzerのインストール済みバージョンが必要です。多くの場合、ダウンロードしたソースコードを事前定義されたFortify Static Code Analyzerコンピュータに移動する方が簡単です。
Launch AWB	Fortify Audit Workbenchを起動して指定したFPRファイルを開くには、[AWBの起動(Launch AWB)]チェックボックスをオンにします。
ZIPファイルの作成(Create ZIP File)	出力を圧縮するには、[ZIPファイルの作成(Create ZIP File)]チェックボックスをオンにします。また、ソースコードをSAPデータベースから抽出した後で、出力を手動で圧縮することもできます。
Export SAP standard code	カスタムコードに加えてSAP標準コードもエクスポートするには、[SAP標準コードのエクスポート(Export SAP standard code)]チェックボックスをオンにします。

5. [実行(Execute)]をクリックします。



Fortify ABAP Extractorのアンインストール

ABAP Extractorをアンインストールするには、次の手順を実行します。

1. ABAP Workbenchで、Object Navigatorを開きます。
2. Y_FORTIFY_ABAPパッケージを選択します。
3. [**Programs**]タブを展開します。
4. 次の要素を右クリックして、[**Delete**]を選択します。
   - プログラム: Y_FORTIFY_SCA

FlexおよびActionScriptの変換

ActionScriptを変換するための基本的なコマンドライン構文は次のとおりです。

```
sourceanalyzer -b <build_id> -flex-libraries <libs> <files>
```

ここで:

*<libs>*は、セミコロンで区切られた(Windows)またはコロンで区切られた(Windows以外のシステム)「リンク」するライブラリ名のリストで、*<files>*は変換するファイルです。

FlexおよびActionScriptコマンドラインオプション

Flexファイルを変換するには、次のコマンドラインオプションを使用します。この情報は、各説明に記載されているproperties環境設定ファイル(`fortify-sca.properties`)でも指定できます。

FlexおよびActionScriptオプション	説明
`-flex-sdk-root` *<dir>*	有効なFlex SDKのルートの場所を指定します。このディレクトリには、`flex-config.xml`ファイルを含むフレームワークフォルダが含まれている必要があります。MXMLC実行可能ファイルを含む`bin`フォルダも含まれている必要があります。 **同等のプロパティ名:** `com.fortify.sca.FlexSdkRoot`
`-flex-libraries` *<libs>*	リンクするライブラリ名をセミコロン区切り(Windows)またはコロン区切り(Windows以外)のリストで指定します。ほとんどの場合、このリストには`flex.swc`、`framework.swc`、`playerglobal.swc`(通常、Flex SDKルートの`frameworks/libs/`にある)が含まれています。 **注:** SWCファイルまたはSWCファイルをFlexライブラリとして指定できます(SWZは現在サポートされていません)。 **同等のプロパティ名:** `com.fortify.sca.FlexLibraries`
`-flex-source-roots` *<dirs>*	MXMLソースが保存されているルートディレクトリをセミコロン区切り(Windows)またはコロン区切り(Windows以外)のリストで指定します。通常、これらには`com`という名前のサブフォルダが含まれます。 たとえば、指定されたFlexソースルートが`foo/bar/src`である場合、`foo/bar/src/com/fortify/manager/util/Foo.mxml`は`com.fortify.manager.util.Foo`という名前のオブジェクト(パッケージ`com.fortify.manager.util`内の`Foo`という名前のオブジェクト)に変換されます。

Flexおよび ActionScriptオプション	説明
	同等のプロパティ名: `com.fortify.sca.FlexSourceRoots`

注: `-flex-sdk-root`および`–flex-source-roots`は主にMXML変換用であり、純粋なActionScriptをスキャンする場合はオプションです。`–flex-libraries`はすべてのActionScriptリンク済みライブラリを解決するために使用します。

Fortify Static Code AnalyzerではMXMLファイルをActionScriptに変換し、ActionScriptパーサを介して実行します。生成されたActionScriptは簡単に分析でき、Flexランタイムモデルのように厳密な正解を示すものではありません。その結果、MXMLファイルで解析エラーが発生する可能性があります。たとえば、XML解析が失敗し、ActionScriptへの変換が失敗し、結果のActionScriptの解析も失敗する可能性があります。元のソースコードに対して明確な接続がないというエラーが発生した場合は、カスタマサポートまでお知らせください。

参照情報

"FlexおよびActionScriptのプロパティ" ページ208

ActionScriptのコマンドライン例

次の例は、ActionScriptの変換に関する一般的なシナリオのコマンドライン構文を示しています。

例1

次の例は、1つのMXMLファイルと1つのMXMLライブラリ(`MyLib.swf`)のみを含む単純なアプリケーションの例です。

```
sourceanalyzer -b MyFlexApp -flex-libraries lib/MyLib.swf -flex-sdk-root
/home/myself/flex-sdk/ -flex-source-roots . my/app/FlexApp.mxml
```

これにより、含めるライブラリの場所、Flex SDK、およびFlexソースルートの場所が識別されます。`/my/app/FlexApp.mxml`にある1つのMXMLファイルは、`my.app`パッケージ内に位置する`FlexApp`という1つのActionScriptクラスとしてMXMLアプリケーションを変換します。

例2

次の例は、ソースファイルがsrcディレクトリに対して相対的であるアプリケーションの例です。1つのSWFライブラリMyLib.swfと、Flex SDKのFlexライブラリとフレームワークライブラリを使用します。

```
sourceanalyzer -b MyFlexProject -flex-sdk-root /home/myself/flex-sdk/
-flex-source-roots src/ -flex-libraries lib/MyLib.swf "src/**/*.mxml"
"src/**/*.as"
```

この例では、Flex SDKを見つけ、Fortify Static Code Analyzerファイル指定子を使用してsrcフォルダ内の.asファイルと.mxmlファイルを含めます。この例では、-flex-sdk-rootにある.SWCファイルを明示的に指定する必要はありません。ただし、この例では例示のために明示的に指定しています。Fortify Static Code Analyzerにより、指定したFlex SDKルート内にあるすべての.SWCファイルが自動的に検索され、ActionScriptまたはMXMLファイルの変換に使用されるライブラリであると見なされます。

例3

この例では、Flex SDKルートとFlexライブラリがプロパティファイルで指定されています。これは、sourceanalyzerを実行するたびに情報を入力するのは時間がかかり、データが頻繁には変わらないためです。アプリケーションを2つのセクションに分割し、mainセクションフォルダとmodulesフォルダに保存します。各フォルダには、パスの起点になるsrcフォルダが含まれています。ファイル指定子には、両方のsrcフォルダ内にあるすべての.mxmlおよび.asファイルを選択するワイルドカードが含まれています。main/src/com/foo/util/Foo.mxml内のMXMLファイルは、たとえば次のようにソースルートを指定して、パッケージcom.foo.util内のFooという名前のActionScriptクラスとして変換されます。

```
sourceanalyzer -b MyFlexProject -flex-source-roots main/src:modules/src
"./main/src/**/*.mxml" "./main/src/**/*.as" "./modules/src/**/*.mxml"
"./modules/src/**/*.as"
```

解決の警告の処理

変換中に生成されたすべての警告を表示するには、スキャンフェーズを開始する前に次のコマンドを入力します。

```
sourceanalyzer -b <build_id> -show-build-warnings
```

ActionScriptの警告

次のようなメッセージが表示される場合があります。

```
The ActionScript front end was unable to resolve the following imports: a.b
at y.as:2. foo.bar at somewhere.as:5. a.b at foo.mxml:8.
```

このエラーは、Fortify Static Code Analyzerで必要なすべてのライブラリが見つからないときに発生します。Fortify Static Code Analyzerで分析を完了するために、(-flex-librariesオプションまたはcom.fortify.sca.FlexLibrariesプロパティを使用して追加のSWCライブラリまたはSWC Flexライブラリを指定する必要がある場合があります。

ColdFusionコードの変換

CFMLページ内の未定義の変数を汚染されているものとして扱う場合は、*<sca_install_dir>*/Core/config/fortify-sca.propertiesの次の行をコメント解除します。

```
#com.fortify.sca.CfmlUndefinedVariablesAreTainted=true
```

これは、register-globalsスタイルの脆弱性に注意するようにDataflow Analyzerに指示します。ただし、このプロパティを有効にすると、インクルードページ内の変数がそれ以前に発生したインクルードページの汚染された値に初期化されることがDataflow Analyzerで判明した場合に支障があります。

ColdFusionコマンドライン構文

ColdFusionソースコードを変換するには、次のコマンドを入力します。

```
sourceanalyzer -b <build_id> -source-base-dir <dir> <files> | <file_specifiers>
```

ここで:

- *<build_id>*はプロジェクトのビルドIDを指定します
- *<dir>*はWebアプリケーションのルートディレクトリを指定します
- *<files>* | *<file_specifiers>* はCFMLソースコードファイルを指定します
  *<file_specifiers>*の使用方法については、"ファイルとディレクトリの指定" ページ149を参照してください。

注: Fortify Static Code Analyzerでは、各CFMLソースファイルへの相対パスを計算するために`-source-base-dir` ディレクトリを開始点として使用します。Fortify Static Code Analyzerでは、インスタンスIDを生成するときに、これらの相対パスを使用します。アプリケーションソースツリー全体を別のディレクトリに移動する場合、`-source-base-dir`オプションに適切なパラメータを指定すると、Fortify Static Code Analyzerで生成されるインスタンスIDは同じままです。

ColdFusion (CFML)コマンドラインオプション

次の表では、CFMLオプションについて説明します。

ColdFusionのオプション	説明
`-source-base-dir` *<web_app_root_dir>* *<files>* | *<file_specifiers>*	Webアプリケーションのルートディレクトリ。 **同等のプロパティ名:** `com.fortify.sca.SourceBaseDir`

参照情報

"ColdFusion (CFML)のプロパティ" ページ208

SQLの変換

Windows (および.NETプロジェクトの場合に限り、Linuxも)のFortify Static Code Analyzerでは、.sql拡張子を持つファイルはPL/SQLではなくT-SQLであることを前提とします。Windowsで.sql拡張子を持つPL/SQLファイルを使用する場合は、PL/SQLとして扱われるようにFortify Static Code Analyzerを設定する必要があります。

Windowsプラットフォームでの変換にSQLタイプを指定するには、次のいずれかの変換コマンドを入力します。

```
sourceanalyzer -b <build_id> -sql-language TSQL <files>
```

または

```
sourceanalyzer -b <build_id> -sql-language PL/SQL <files>
```

または、.sql拡張子を持つファイルのデフォルトの動作を変更できます。fortify-sca.propertiesファイルで、com.fortify.sca.fileextensions.sqlプロパティをPLSQLまたはTSQLに設定します。

参照情報

"SQLのプロパティ" ページ209

PL/SQLのコマンドライン例

次の例では、2つのPL/SQLファイルを変換します。

```
sourceanalyzer -b MyProject x.pks y.pks
```

次の例では、sourcesディレクトリ内のすべてのPL/SQLファイルを変換します。

```
sourceanalyzer -b MyProject "sources/**/*.pks"
```

T-SQLのコマンドライン例

次の例では、2つのT-SQLファイルを変換します。

```
sourceanalyzer -b MyProject x.sql y.sql
```

次の例では、sourcesディレクトリ内のすべてのT-SQLファイルを変換します。

```
sourceanalyzer -b MyProject "sources\**\*.sql"
```

注: この例では、fortify-sca.properties内のcom.fortify.sca.fileextensions.sqlプロパティがTSQLに設定されている場合を想定しています。

Scalaコードの変換

Scalaコードの変換に必要なものは次のとおりです。

- Scalaコンパイラプラグイン
  このプラグインは、Maven Central Repositoryからダウンロードできます。
- Lightbendライセンスファイル
  このライセンスファイルは、Fortify Static Code Analyzerのインストールに含まれ、*<sca_install_dir>*/plugins/lightbendディレクトリにあります。

ライセンスの設定方法およびScalaコードの変換方法については、https://developer.lightbend.com/guides/fortifyにあるLightbendのドキュメントを参照してください。

重要 プロジェクトに、Scala以外のソースコードが含まれている場合は、分析フェーズを実行する前にScala Fortifyコンパイラプラグインを使用してScalaコードを変換してから、同じビルドIDでsourceanalyzerを使用してその他のソースコードを変換する必要があります。

コードとしてのインフラストラクチャ(IaC)の変換

Fortify Static Code Analyzerは、Azure Resource Manager (ARM)、Bicep、AWS CloudFormation、およびHCLテンプレートを変換します。

注: HCLの分析サポートは、Terraformおよびサポートされるクラウドプロバイダのコードとしてのインフラストラクチャ(IaC)構成に固有のものです。

最適な結果を得るには、テンプレートファイルの展開が有効であることを確認してください。テンプレートには次が含まれていてはなりません。

- 静的であり、ローカルで検出可能な検証エラー(タイプエラーや未定義の変数または関数への参照など)。

- テンプレートの解釈中、ただしリソースが展開または変更される前に発生する、展開前エラー(無効な配列のインデックス付け操作など)。
- クラウドで発生する展開エラー(存在しないリソースの動的参照など)。

Fortifyでは、AWS CloudFormationのファイル名拡張子を.json、.yaml、.template、または.txtにすることを推奨しています。Fortify Static Code Analyzerでは、他の言語やファイルタイプでそれらの拡張子が一般的に使用されていない場合にのみ、他の拡張子をサポートしています(.javaや.htmlなど)。

デフォルトで、Fortify Static Code Analyzerでは、HCL拡張子.hclおよび.tfを持つファイルを変換します。

ARM変換コマンドラインの例

ARMテンプレートを変換する場合:

```
sourceanalyzer -b MyProject ArmTemplate.json
```

ディレクトリ内のすべてのARMテンプレートを変換する場合:

```
sourceanalyzer -b MyProject "src/**/*.json"
```

Bicep変換コマンドラインの例

1つのBicepテンプレートを変換する場合:

```
sourceanalyzer -b MyProject BicepTemplate.bicep
```

ディレクトリ内のすべてのBicepテンプレートを変換する場合:

```
sourceanalyzer -b MyProject "src/**/*.bicep"
```

AWS CloudFormation変換コマンドラインの例

異なる拡張子を持つAWS CloudFormationテンプレートを変換する場合:

```
sourceanalyzer -b MyProject CFTemplateA.template CFTemplateB.yaml
CFTemplateC.json CFTemplateD.customext
```

.template拡張子を持つディレクトリ内のすべてのAWS CloudFormationテンプレートを変換する場合:

```
sourceanalyzer -b MyProject "src/**/*.template"
```

.jsonまたは.yamlの拡張子を持つディレクトリ内のすべてのAWS CloudFormationテンプレートを変換する場合:

```
sourceanalyzer -b MyProject "src/**/*.json" "src/**/*.yaml"
```

HCL変換コマンドラインの例

異なる拡張子を持つ2つのHCLテンプレートを変換する場合:

```
sourceanalyzer -b MyProject HCLTemplateA.hcl HCLTemplateB.tf
```

ディレクトリ内のすべてのHCLテンプレートを変換する場合:

```
sourceanalyzer -b MyProject "src/**/*.tf" "src/**/*.hcl"
```

参照情報

"JSONの変換" 下

"YAMLの変換" 下

JSONの変換

デフォルトで、Fortify Static Code Analyzerでは、JSON拡張子.jsonを持つファイルをJSONとして変換します。次の例では、1つのJSONファイルを変換します。

```
sourceanalyzer -b MyProject x.json
```

次の例では、sourcesディレクトリ内のすべてのJSONファイルを変換します。

```
sourceanalyzer -b MyProject "sources/**/*.json"
```

YAMLの変換

デフォルトで、Fortify Static Code Analyzerでは、YAML拡張子.yamlおよび.ymlを持つファイルを変換します。次の例では、異なるファイル拡張子を持つ2つのYAMLファイルを変換します。

```
sourceanalyzer -b MyProject x.yaml y.yml
```

次の例では、sourcesディレクトリ内のすべてのYAMLファイルを変換します。

```
sourceanalyzer -b MyProject "sources/**/*.yaml" "sources/**/*.yml"
```

Dockerfileの変換

デフォルトで、Fortify Static Code Analyzerでは`Dockerfile*`、`dockerfile*`、`*.Dockerfile`、および`*.dockerfile`ファイルがDockerfileとして返還されます。

注: `com.fortify.sca.fileextensions`プロパティを使用して、Dockerfileを検出するために使用されるファイル名拡張子を変更できます。「"変換と分析フェーズのプロパティ" ページ187」を参照してください。

Fortify Static Code Analyzerでは、Dockerfile内のエスケープ文字としてバックスラッシュ(\)とバッククォート(`)を受け付けます。Dockerfileでエスケープ文字が設定されていない場合、Fortify Static Code Analyzerではバックスラッシュがエスケープ文字と見なされます。

Dockerfileを含むディレクトリを変換する構文を次の例に示します。

```
sourceanalyzer -b <build_id> <dir>
```

Dockerfileの形式が正しくない場合、Fortify Static Code Analyzerはログファイルにエラーを書き込んでファイルを解析できないことを示し、そのDockerfileの分析をスキップします。ログに書き込まれるエラーの例を次に示します。

```
Unable to parse dockerfile ProjA.Dockerfile, error on Line 1:20: mismatched
input '\n' expecting {LINE_EXTEND, WHITESPACE}

Unable to parse config file
C:/Users/jsmith/MyProj/docker/dockerfile/ProjA.Dockerfile
```

ASP/VBScript仮想ルートの変換

Fortify Static Code Analyzerでは、ASP仮想ルートを処理できます。物理ディレクトリにマップするエイリアスとして仮想ディレクトリを使用するWebサーバの場合、Fortify Static Code Analyzerを使用するとエイリアスを使用できます。

たとえば、`Include`および`Library`という名前の仮想ディレクトリがあり、それぞれ物理ディレクトリ`C:\WebServer\CustomerOne\inc`および`C:\WebServer\CustomerTwo\Stuff`を参照しているとします。

次の例では、*virtual*インクルードを使用するアプリケーションのASP/VBScriptコードを示しています。

```
<!--#include virtual="Include/Task1/foo.inc"-->
```

この例で、前述のASPコードは次の物理的な場所にあるファイルを参照します。

```
C:\Webserver\CustomerOne\inc\Task1\foo.inc
```

この例では、実際のディレクトリが仮想ディレクトリ名`Include`に置き換わります。

仮想ルートへの対応

各仮想ディレクトリのマッピングをFortify Static Code Analyzerに提供するには、次の例に示すように、Fortify Static Code Analyzerコマンドライン呼び出しで`com.fortify.sca.ASPVirtualRoots.name_of_virtual_directory`プロパティを設定する必要があります。

```
sourceanalyzer -Dcom.fortify.sca.ASPVirtualRoots.<virtual_directory>=<full_path_to_corresponding_physical_directory>
```

注: Windowsでは、物理パスにスペースが含まれている場合は、プロパティ設定を引用符で囲む必要があります。

```
sourceanalyzer "-Dcom.fortify.sca.ASPVirtualRoots.<virtual_directory>=<full_path_to_corresponding_physical_directory>"
```

前のセクションの例で展開するには、次のプロパティ値をFortify Static Code Analyzerに渡します。

```
-Dcom.fortify.sca.ASPVirtualRoots.Include="C:\WebServer\CustomerOne\inc"

-Dcom.fortify.sca.ASPVirtualRoots.Library="C:\WebServer\CustomerTwo\Stuff"
```

これにより、`Include`は`C:\WebServer\CustomerOne\inc`に、`Library`は`C:\WebServer\CustomerTwo\Stuff`にマップされます。

Fortify Static Code Analyzerで`#include`ディレクティブが出現する場合:

```
<!-- #include virtual="Include/Task1/foo.inc" -->
```

Fortify Static Code Analyzerでは、プロジェクトに`Include`という名前の物理ディレクトリが含まれているかどうかを特定します。このような物理ディレクトリがない場合、Fortify Static Code Analyzerではそのランタイムプロパティを調べて、`-Dcom.fortify.sca.ASPVirtualRoots.Include="C:\WebServer\CustomerOne\inc"`設定を見つけます。その後、Fortify Static Code Analyzerでは、`C:\WebServer\CustomerOne\inc\Task1\foo.inc`ファイルを探します。

または、*`<sca_install_dir>`*`\Core\config`にある`fortify-sca.properties`ファイルでこのプロパティを設定できます。次の例に示すように、物理ディレクトリのパス内のバックスラッシュ文字(\)をエスケープする必要があります。

```
com.fortify.sca.ASPVirtualRoots.Library=C:\\WebServer\\CustomerTwo\\Stuff

com.fortify.sca.ASPVirtualRoots.Include=C:\\WebServer\\CustomerOne\\inc
```

注: 以前のバージョンのASPVirtualRootプロパティは引き続き有効です。Fortify Static Code Analyzerコマンドラインで次のように使用できます。

```
-Dcom.fortify.sca.ASPVirtualRoots=C:\WebServer\CustomerTwo\Stuff;C:\WebServer\CustomerOne\inc
```

これはFortify Static Code Analyzerに対し、virtualインクルードディレクティブの解決時に、指定した順序でリストされているディレクトリを検索するように促します。

仮想ルートの使用例

次のファイルがあります。

```
C:\files\foo\bar.asp
```

このファイルを指定するには、次のインクルードを使用します。

```
<!-- #include virtual="/foo/bar.asp">
```

次に、sourceanalyzerコマンドで仮想ルートを次のように設定します。

```
-Dcom.fortify.sca.ASPVirtualRoots=C:\files\foo
```

これにより、仮想ルートの前から/fooがストライプされます。fooを com.fortify.sca.ASPVirtualRootsプロパティで指定しない場合、Fortify Static Code AnalyzerではC:\files\bar.aspを探して失敗します。

仮想ルートを指定するシーケンスは次のとおりです。

1. ソースのパスの最初の部分を削除します。
2. パスの最初の部分を、コマンドラインで指定した仮想ルートに置き換えます。

Classic ASPのコマンドライン例

VBScriptで記述された、MyASP.aspという名前の単一ファイルのClassic ASPを変換するには、次のコマンドを入力します。

```
sourceanalyzer -b mybuild "MyASP.asp"
```

VBScriptのコマンドライン例

myApp.vbという名前のVBScriptファイルを変換するには、次のコマンドを入力します。

```
sourceanalyzer -b mybuild "myApp.vb"
```

第17章: ビルドへのFortify Static Code Analyzer統合

サポートされているビルドツールに分析を統合できます。

このセクションでは、次のトピックについて説明します。

ビルド統合

プロジェクト全体を1回の操作で変換できます。元のビルド操作の前に`sourceanalyzer`コマンドをFortify Static Code Analyzerオプション付きで追加します。

完全なプロジェクトを変換するための基本的なコマンドライン構文は、次のとおりです。

```
sourceanalyzer -b <build_id> [<sca_options>] <build_tool> [<build_tool_options>]
```

ここで、*<build_tool>*はビルドツール(make、gmake、msbuild、devenv、gbuildなど)の名前です。サポートされているビルドツールのリストについては、ドキュメント『*Fortifyソフトウェアシステム要件*』を参照してください。Fortify Static Code Analyzerでは、ビルドツールが実行され、すべてのコンパイラ操作を傍受して各入力に使用される特定のコマンドラインが収集されます。

注: Fortify Static Code Analyzerでは、ビルドツールで実行されるコンパイラコマンドのみが処理されます。ビルドを実行する前にプロジェクトをクリーンアップしない場合は、ビルドツールで再コンパイルされるファイルのみがFortify Static Code Analyzerで処理されます。

Xcodebuildとの統合については、「"iOSコード分析のコマンドライン構文" ページ86」を参照してください。
MSBuildとの統合については、「"Visual Studioプロジェクトの変換" ページ66」を参照してください。

ビルドを正常に統合するには、ビルドツールで次が実行される必要があります。

- Fortify Static Code Analyzerでサポートされているコンパイラを実行する
- ハードコードされたパスではなく、オペレーティングシステムパスの検索でコンパイラを実行する (xcodebuildの統合には、この要件が必ずしも適用されません。)
- 後でコンパイラが実行されるサブプロセスを実行するのではなく、コンパイラを実行する

ご使用の環境でこれらの要件を満たせない場合は、「"Fortify Static Code Analyzerを開始するビルドスクリプトの変更" 下」を参照してください。

makeの例

次のビルドコマンドを使用してプロジェクトを構築する場合:

```
make clean make make install
```

次のサンプルコマンドを使用して、プロジェクト全体を同時に変換およびコンパイルできます。

```
make clean sourceanalyzer -b MyProject make make install
```

Fortify Static Code Analyzerを開始するビルドスクリプトの変更

ビルド統合の代替方法として、ビルドスクリプトを変更して、各コンパイラ、リンカ、およびアーカイバ操作の前に`sourceanalyzer`コマンドを追加することもできます。たとえば、makefileでは、これらのツールの名前に変数が定義されることが多いです。

```
CC=gcc
CXX=g++
LD=ld AR=ar
```

makefile内のツール参照の前に、`sourceanalyzer`コマンドと適切なFortify Static Code Analyzerオプションを追加できます。

```
CC=sourceanalyzer -b mybuild gcc CXX=sourceanalyzer -b mybuild g++
LD=sourceanalyzer -b mybuild ld AR=sourceanalyzer -b mybuild ar
```

各操作に同じビルドIDを使用する場合は、Fortify Static Code Analyzerで自動的に、個別に変換された各ファイルが変換された単一のプロジェクトに結合されます。

タッチレスビルド統合の使用

Fortify Static Code Analyzerには、Fortify Static Code Analyzerで直接サポートされていないビルドシステムを使用してプロジェクトを変換できる`touchless`という汎用ビルドツールが含まれています。

touchlessビルド統合のコマンドライン構文は次のとおりです。

```
sourceanalyzer -b <build_id> touchless <build_command>
```

たとえば、build.pyというPythonスクリプトを使用して依存関係を計算し、適切に順序付けられたCコンパイラ操作を実行する場合があります。ビルドを実行するには、次のコマンドを実行します。

```
python build.py
```

Fortify Static Code Analyzerでは、このようなビルド設計がネイティブサポートされていません。ただし、touchlessビルドツールを使用して、単一のコマンドでプロジェクト全体を変換およびビルドできます。

```
sourceanalyzer -b <build_id> touchless python build.py
```

この章の前半で説明したサポートされているビルドシステムにビルドを正常に統合するための同じ要件(「"ビルド統合" ページ124」を参照)は、サポートされていないビルドシステムとのtouchlessの統合にも適用されます。

Antとの統合

Antビルドファイルを使用するプロジェクト用に、Javaソースファイルを変換できます。この統合は、Antのbuild.xmlファイルを変更せずにコマンドラインに適用できます。ビルドが実行されると、Fortify Static Code Analyzerでjavacタスクの呼び出しがすべて傍受され、Javaソースファイルがコンパイル時に変換されます。プロパティはすべて、ANT_OPTS環境変数に追加して、Antに渡すようにします。sourceanalyzerコマンドには含めないでください。

注: アプリケーションの一部であるJSPファイル、環境設定ファイル、またはJava以外のソースファイルを個別のステップで変換する必要があります。

Antの統合を使用するには、sourceanalyzer実行可能ファイルがPATHシステムにインストールされていることを確認します。

次のように、Antコマンドラインの前にsourceanalyzerコマンドを追加します。

```
sourceanalyzer -b <build_id> ant [<ant_options>]
```

Bazelとの統合

JavaまたはPythonで記述され、Bazelを使用してビルドされたプロジェクトを変換できます。ビルドが実行されると、Fortify Static Code Analyzerでソースファイルがコンパイル時に変換されます。サポートされているBazelのバージョンについては、*Fortifyソフトウェアシステム要件*のドキュメントを参照してください。

Fortify Static Code AnalyzerとBazelの統合を実行する前に、次の要件を満たしていることを確認します。

- Bazelのビルドがエラーなく実行されている。
- `sourceanalyzer`実行可能ファイルがシステムパス上にある。
  ビルドが完了するたびに、システムパス上にあるFortify Static Code Analyzerと同じバージョンを指定して、Fortify Static Code Analyzerの分析フェーズを実行します。

設定済みのJavaまたはPythonの変換フェーズを実行するには、Bazelワークスペースディレクトリに移動し、ビルドするターゲットを指定してFortify Static Code Analyzerコマンドを実行します。Bazelのビルドコマンドラインの前に、次のように`sourceanalyzer`コマンドを付加します。

```
sourceanalyzer -b <build_id> <sca_options> bazel build <target>
```

注: Fortify Static Code Analyzerが複数インストールされている場合、Bazelプロジェクトに使用するバージョンは、システムパス上の他のすべてのバージョンのFortify Static Code Analyzerより前に定義されている必要があります。

例

特定のターゲットのプロジェクトを変換する場合:

```
sourceanalyzer -b MyProjectA bazel build //proja:my-prj
```

パッケージ`proja/abc`内のターゲット`abc`を変換する場合:

```
sourceanalyzer -b MyProjectA bazel build //proja/abc
```

または

```
sourceanalyzer -b MyProjectA bazel build //proja/abc:abc
```

パッケージ`proja/abc`内のすべてのターゲットを変換する場合:

```
sourceanalyzer -b MyProjectA bazel build //proja/abc:all
```

`projb/`ディレクトリ内のすべてのターゲットを変換する場合:

```
sourceanalyzer -b MyProjectB bazel build //projb/...
```

変換に特定のJDKバージョンを指定する場合:

```
sourceanalyzer -b MyProjectC -jdk 17 bazel build //projc:my-java-prj
```

変換にPythonプロジェクトの依存関係を指定する場合:

```
sourceanalyzer -b MyProjectD -python-path /usr/local/lib/python3.6/ bazel
build //projd:my-python-app
```

Fortify Static Code AnalyzerとBazelの統合では、複数のターゲットおよび関連アクション(ターゲットの除外など)はサポートされていません。

参照情報

"Javaコマンドラインオプション" ページ54

"Pythonコマンドラインオプション" ページ80

CMakeとの統合

Windows以外のシステムでは、Fortify Static Code AnalyzerコマンドにJSONコンパイルデータベースを組み込むことによって、CMakeを使用してビルドされたプロジェクトを変換できます。これは、MakefileジェネレータとNinjaジェネレータでのみサポートされています(詳細については、『CMake Reference Documentation』を参照してください)。

Fortify Static Code AnalyzerをCMakeビルドと統合するには:

1. CMakeプロジェクト用の`compile_commands.json`ファイルを生成します。
   `cmake`設定コマンドに`-DCMAKE_EXPORT_COMPILE_COMMANDS=yes`を追加します。例:

   ```
   cmake -G Ninja -DCMAKE_EXPORT_COMPILE_COMMANDS=yes
   ```

2. 次のようにして、`sourceanalyzer`コマンドにJSONコンパイルデータベースを含めます。

   ```
   sourceanalyzer -b <build_id> compile_commands.json
   ```

Gradleとの統合

Fortify Static Code Analyzerは、Gradleでビルドされたプロジェクトとの変換統合を提供します。ビルドスクリプトを変更せずに統合するか、Fortify Static Code Analyzer Gradleプラグインを使用することができます。

Gradle統合の使用

`build.gradle`ファイルを変更せずに、Gradleを使用してビルドされたプロジェクトを変換できます。ビルドが実行されると、Fortify Static Code Analyzerでソースファイルがコンパイル時に変換されます。代わりに、Fortify Static Code Analyzer Gradle Pluginを使用してGradleビルドスクリプト内から分析を実行できます(「"Fortify Static Code Analyzer Gradleプラグインの使用" ページ130」を参照)。

特にGradle統合でサポートされているプラットフォームおよび言語については、ドキュメント『*Fortifyソフトウェアシステム要件*』を参照してください。プロジェクト内のファイルは、Gradleの統合でサポートされていない言語では変換されません(エラーもレポートされません)。したがって、これらのファイルは分析されず、既存の潜在的な脆弱性は検出されない可能性があります。

Fortify Static Code AnalyzerをGradleビルドに統合するには、sourceanalyzer実行可能ファイルがシステムのパス上にあることを確認します。プロジェクトをビルドするすべてのGradleコマンドに対して、システムパスのsourceanalyzer実行可能ファイルを常に使用します。

注: Fortify Static Code Analyzerが複数インストールされている場合、Gradleプロジェクトに使用するバージョンは、システムパス上の他のすべてのバージョンのFortify Static Code Analyzerより前に定義されている必要があります。

次のように、Gradleコマンドラインの前にsourceanalyzerコマンドを追加します。

```
sourceanalyzer -b <build_id> <sca_options> gradle [<gradle_options>]
<gradle_tasks>
```

例

```
sourceanalyzer -b MyProject gradle clean build sourceanalyzer -b MyProject
gradle --info assemble
```

ビルドファイル名がbuild.gradleと異なる場合は、次の例に示すように、ビルドファイル名に--build-fileオプションを付けます。

```
sourceanalyzer -b MyProject gradle --build-file sample.gradle clean
assemble
```

次の例に示すように、Gradle Wrapper (gradlew)を使用することもできます。

```
sourceanalyzer -b MyProject gradlew [<gradle_options>]
```

アプリケーションでXMLまたはプロパティ環境設定ファイルが使用されている場合は、これらのファイルを個別のsourceanalyzerコマンドを使用して変換します。プロジェクトファイルで使用したときと同じビルドIDを使用します。次に例を示します。

```
sourceanalyzer -b MyProject <path_to_xml_files> sourceanalyzer -b MyProject
<path_to_properties_files>
```

gradleまたはgradlewでプロジェクトが変換されたら、次の例に示すように分析フェーズを実行して、結果をFPRファイルに保存できます。

```
sourceanalyzer -b MyProject -scan -f MyResults.fpr
```

詳細オプションとデバッグオプションを含める

Fortify Static Code Analyzerの-verboseオプションを使用する場合は、-gradleオプションも含める必要があります。このオプションの使用は、GradleとGradle Wrapperの両方に適用されます。例:

```
sourceanalyzer -b MyProject -gradle -verbose gradle assemble
```

Gradleの統合の一環として、Fortify Static Code Analyzerで元のビルドファイルbuild.gradleが一時的に更新されます。-debugオプションを含める場合は、Fortify Static Code Analyzerに元のビルドファイルのコピーがbuild.gradle.origとして保存されます。-debugオプションを使用した分析が完了したら、build.gradle.origファイルの名前をbuild.gradleに戻し、-debugオプションを付けずにsourceanalyzerを再度実行します。

参照情報

"Fortify Static Code Analyzer Gradleプラグインの使用" 下

Gradle統合のトラブルシューティング

Fortify Static Code AnalyzerGradle統合を使用したGradleビルドで設定キャッシング(--configuration-cacheオプション)を使用した場合、ビルドは次のメッセージをレポートします。

```
このビルドで設定キャッシュの問題が見つかりました。(Configuration cache problems found in this build.)
```

次のようなエラーメッセージが表示される場合もあります。

```
エラー: 例外を出してビルドが失敗しました... (FAILURE: Build failed with an exception...)
```

このエラーメッセージは、Fortify Static Code Analyzer変換に関しては無視して構いません。プロジェクトが変換されているからです。プロジェクトが変換されていることは、-show-filesオプションを使用して確認できます。例:

```
sourceanalyzer -b mybuild -show-files
```

Fortify Static Code Analyzer Gradleプラグインの使用

Fortify Static Code Analyzerのインストールには、*<sca_install_dir>*/plugins/gradleにGradleプラグインが含まれています。Fortify Static Code Analyzer Gradleプラグインを使用するには、まずJavaまたはKotlinプロジェクト用にプラグインを設定してから、そのプラグインを使用してプロジェクトを分析する必要があります。Gradleプラグインは、分析用に3つのFortify Static Code Analyzerタスク(sca.clean、sca.translate、およびsca.scan)を提供しています。特にFortify Static Code Analyzer Gradleプラグイン用にサポートされているプラットフォームと言語については、ドキュメント『*Fortifyソフトウェアシステム要件*』を参照してください。

注: Fortify Static Code Analyzerが複数インストールされている場合、Gradleプロジェクトに使用するバージョンは、システムパス上の他のすべてのバージョンのFortify Static Code Analyzerより前に定義されている必要があります。

Fortify Static Code Analyzer Gradleプラグインを設定するには:

1. Gradle設定ファイルを編集して、プラグインへのパスを指定します。
   - Groovy DSL (`settings.gradle`):

     ```
     pluginManagement { repositories { gradlePluginPortal() maven { url =
     uri("file://<sca_plugin_path>") } } }
     ```

   - Kotlin DSL (`settings.gradle.kts`):

     ```
     pluginManagement { repositories { maven(url = uri("file://<sca_
     plugin_path>")) gradlePluginPortal() } }
     ```

2. 以下の例に示すように、ビルドスクリプトにエントリを追加します。
   - Groovy DSL (`build.gradle`):

     ```
     id 'com.fortify.sca.plugins.gradlebuild' version '24.2'
     ```

     および

     ```
     SCAPluginExtension { buildId = "mybuild" options = ["-encoding",
     "utf-8", "-logfile", "mybuild.log", "-debug-verbose"] }
     ```

   - Kotlin DSL (`build.gradle.kts`):

     ```
     plugins { id ("com.fortify.sca.plugins.gradlebuild") version "24.2"
     ... }
     ```

     および

     ```
     SCAPluginExtension { buildId = "mybuild" options = listOf("-
     encoding", "utf-8", "-logfile", "mybuild.log", "-debug-verbose") }
     ```

3. Gradle設定ファイルとGradleビルドファイルを保存して閉じます。

次のコマンドシーケンスを使用して、JavaまたはKotlinプロジェクトを分析します。

- 既存のJavaまたはKotlinプロジェクトビルドの既存のFortify Static Code Analyzer一時ファイルをすべて削除するには、次のコマンドを実行します。

  ```
  gradlew sca.clean
  ```

- 設定済みのJavaまたはKotlinプロジェクトの変換フェーズを実行するには、次のコマンドを実行します。

  ```
  gradlew sca.translate
  ```

- 設定済みのJavaまたはKotlinプロジェクトを分析するには、次のコマンドを実行します。

```
gradlew sca.scan
```

このタスクは、Fortify Static Code Analyzerによってプロジェクトが Fortify Static Code Analyzer Gradle プラグインを使用してすでに変換されている場合に、正常に実行されます。

サブプロジェクトを持つJavaまたはKotlinプロジェクトの操作

JavaまたはKotlinマルチプロジェクトビルド(サブプロジェクトを含む)がある場合は、Fortify Static Code Analyzer Gradleプラグインを`allprojects`ブロックを使用して設定する必要があります。以下に例を示します。

Groovy DSL (`build.gradle`)

```
allprojects { apply plugin: "com.fortify.sca.plugins.gradlebuild"
SCAPluginExtension { buildId = "mybuild" options = ["-encoding", "utf-8",
"-logfile", "mybuild.log", "-debug-verbose"] ... } }
```

Kotlin DSL (`build.gradle.kts`):

```
allprojects { apply(plugin = "com.fortify.sca.plugins.gradlebuild")
SCAPluginExtension { buildId = "mybuild" options = listOf("-encoding",
"utf-8", "-logfile", "mybuild.log", "-debug-verbose") ... } }
```

参照情報

"Gradle統合の使用" ページ128

Mavenとの統合

Fortify Static Code Analyzerには、Mavenプロジェクトのビルドに次の機能を追加する方法を提供するMavenプラグインが含まれています。

- Fortify Static Code Analyzerで消去、変換、スキャンする
- Fortify Static Code Analyzerで変換されたプロジェクトのモバイルビルドセッション(MBS)をFortify Static Code Analyzerからエクスポートする
- 変換されたコードをFortify ScanCentral SASTに送信する
- 結果をFortify Software Security Centerにアップロードする

プラグインを直接使用することも、プラグインの機能をビルドプロセスに統合することもできます。

Fortify Mavenプラグインのインストールと更新

Fortify Mavenプラグインは、*<sca_install_dir>*`/plugins/maven`に配置されています。このディレクトリには、バイナリバージョンとソースバージョンのプラグインがzipアーカイブとtarballアーカイブの両方で含まれています。プラグインをインストールするには、使用するバージョン(バイナリまたはソース)を抽出し、

付属のREADME.TXTファイルの指示に従います。アーカイブを展開したディレクトリでインストールを実行します。

Mavenでサポートされているバージョンについては、ドキュメント『*Fortify ソフトウェアシステム要件*』を参照してください。

以前のバージョンのFortify Mavenプラグインがインストールされている場合は、最新バージョンをインストールします。

Fortify Mavenプラグインのアンインストール

Fortify Mavenプラグインをアンインストールするには、*<maven_local_repo>*/repository/com/fortify/ps/maven/pluginディレクトリからすべてのファイルを手動で削除します。

Fortify Mavenプラグインインストールのテスト

Fortify Mavenプラグインをインストールしたら、付属のサンプルファイルのいずれかを使用して正しくインストールされていることを確認します。

Eightballサンプルファイルを使用してFortify Mavenプラグインをテストするには、次の手順に従います。

1. sourceanalyzer実行可能ファイルを含むディレクトリをパス環境変数に追加します。
   例:

   ```
   export set PATH=$PATH:/<sca_install_dir>/bin
   ```

   または

   ```
   set PATH=%PATH%;<sca_install_dir>/bin
   ```

2. 「sourceanalyzer -version」と入力してパスの設定をテストします。
   パスの設定が正しい場合は、Fortify Static Code Analyzerにバージョン情報が表示されます。
3. サンプルのEightballディレクトリ(*<root_dir>*/samples/EightBall)に移動します。
4. 次のコマンドを入力します。

   ```
   mvn com.fortify.sca.plugins.maven:sca-maven-plugin:<ver>:clean
   ```

   ここで、*<ver>*は使用しているFortify Mavenプラグインのバージョンです。バージョンが指定されていない場合は、ローカルリポジトリにインストールされている最新バージョンのFortify MavenプラグインがMavenで使用されます。

   注: Fortify Mavenプラグインのバージョンを表示するには、テキストエディタで*<root_dir>*に抽出したpom.xmlファイルを開きます。Fortify Mavenプラグインのバージョンは、<version>要素で指定されます。

5. ステップ4のコマンドが正常に完了した場合は、Fortify Mavenプラグインが正しくインストールされています。次のエラーメッセージが表示された場合は、Fortify Mavenプラグインが正しくインストールされていません。

   ```
   [ERROR] Error resolving version for plugin
   'com.fortify.sca.plugins.maven:sca-maven-plugin' from the repositories
   ```

   Mavenのローカルリポジトリをチェックし、Fortify Mavenプラグインの再インストールを試みます。

Fortify Mavenプラグインの使用

Mavenプロジェクトでは、次の2つの方法でFortify分析を実行します。

- Mavenプラグインとして

  この方法では、`mvn`コマンドを使用してFortify分析タスクを目標として実行します。たとえば、次のコマンドを使用してソースコードを変換します。

  ```
  mvn com.fortify.sca.plugins.maven:sca-maven-plugin:<ver>:translate
  ```

  この方法でコードを分析するには、Fortify Mavenプラグインに付属のドキュメントを参照してください。次の表では、Fortify Mavenプラグインをインストールした後にドキュメントが置かれる場所について説明します。

パッケージタイプ	ドキュメントの場所
バイナリ	*<root_dir>*/docs/index.html
ソース	*<root_dir>*/sca-maven-plugin/target/site/index.html

- Fortify Static Code Analyzerビルド統合時

  この方法では、プロジェクトをビルドするために使用されるMavenコマンドの前に`sourceanalyzer`コマンドとFortify Static Code Analyzerオプションを追加します。Fortify Static Code Analyzerのビルド統合の一環としてファイルを分析するには、次の手順に従います。

  a. 以前のビルドを消去します。

     ```
     sourceanalyzer -b MyProject -clean
     ```

  b. コードを変換します。

     ```
     sourceanalyzer -b MyProject [<sca_options>] [<mvn_command_with_
     options>]
     ```

     例:

     ```
     sourceanalyzer -b MyProject mvn package
     ```

使用可能なFortify Static Code Analyzerオプションについては、「"コマンドラインインタフェース" ページ136」を参照してください。

c. 次の例に示すように、スキャンを実行して結果をFPRファイルに保存することにより、分析を完了します。

```
sourceanalyzer -b MyProject [<sca_scan_options>] -scan -f
MyResults.fpr
```

第18章: コマンドラインインタフェース

この章では、一般的なFortify Static Code Analyzerのコマンドラインオプションと分析対象のソースファイルを指定する方法について説明します。特定の言語に固有のコマンドラインオプションについては、その言語の章で説明します。

このセクションでは、次のトピックについて説明します。

変換オプション

次の表では、変換オプションについて説明します。

変換オプション	説明
`-b <build_id>`	ビルドIDを指定します。Fortify Static Code Analyzerでは、ビルドIDを使用して、ビルドの一環としてコンパイルおよび結合されたファイルが追跡され、後でそれらのファイルがスキャンされます。 **同等のプロパティ名:** `com.fortify.sca.BuildID`
`-disable-language <Languages>`	変換フェーズから除外する言語のコロン区切りリストを指定します。 有効な言語の値は、`abap`、`actionscript`、`apex`、`cfml`、`cobol`、`configuration`、`cpp`、`dart`、`dotnet`、`golang`、`java`、`javascript`、`jsp`、`kotlin`、`objc`、`php`、`plsql`、`python`、`ruby`、`scala`、`sql`、`swift`、`tsql`、`typescript`、および`vb`です。 **同等のプロパティ名:** `com.fortify.sca.DISabledLanguages`

変換オプション	説明
`-enable-language <Languages>`	変換する言語のコロン区切りリストを指定します。有効な言語の値は、`abap`、`actionscript`、`apex`、`cfml`、`cobol`、`configuration`、`cpp`、`dart`、`dotnet`、`golang`、`java`、`javascript`、`jsp`、`kotlin`、`objc`、`php`、`plsql`、`python`、`ruby`、`scala`、`sql`、`swift`、`tsql`、`typescript`、および`vb`です。 **同等のプロパティ名:** `com.fortify.sca.EnabledLanguages`

`-exclude <file_specifiers>`	変換から除外するファイルを指定します。変換から除外されたファイルはスキャンもされません。複数のファイルパスはセミコロン(Windows)またはコロン(Windows以外)で区切ります。例:
	`sourceanalyzer -cp "**/*.jar" "**/*" -exclude "**/Test/*.java"`
	この例では、任意の`Test`サブディレクトリ内のすべてのJavaファイルを除外します。ファイル指定子を使用する方法の詳細については、「"ファイルとディレクトリの指定" ページ149」を参照してください。
	注: ほとんどのコンパイラまたはビルドツールと変換を統合すると、このオプションで除外するように指定されている場合でも、コンパイラまたはビルドツールで処理されるソースファイルがすべてFortify Static Code Analyzerで変換されます。ただし、Fortify Static Code Analyzerの`xcodebuild`および`MSBuild`では、`-exclude`オプションがサポートされています。
	同等のプロパティ名:
	`com.fortify.sca.exclude`

`-encoding <encoding_name>`	ソースファイルのエンコーディングタイプを指定します。Fortify Static Code Analyzerでは、エンコードの異なるソースファイルを含むプロジェクトをスキャンできます。複数のエンコードを含むプロジェクトを処理するには、Fortify Static Code Analyzerで最初にソースコードファイルを読み込む際に、変換フェーズで`-encoding`オプションを指定する必要があります。Fortify Static Code Analyzerでは、このエンコーディングがビルドセッションで記憶され、FVDLファイルに反映されます。 有効なエンコーディング名は`java.nio.charset.Charset`です。 通常、エンコーディングタイプを指定しないと、Fortify Static Code Analyzerではエンコーディングパラメータなしで

変換オプション	説明
	java.io.InputStreamReaderコンストラクタからfile.encodingが使用されます。一部のケース(ActionScriptパーサの場合など)では、Fortify Static Code Analyzerのデフォルトは UTF-8エンコーディングです。 **同等のプロパティ名:** com.fortify.sca.InputFileEncoding
-nc	コンパイラのコマンドラインの前に指定すると、Fortify Static Code Analyzerでソースファイルが変換されますが、コンパイラは実行されません。
-noextension-type *<file_type>*	拡張子がないソースファイルにファイルタイプを指定します。有効なファイルタイプの値はABAP、ACTIONSCRIPT、APEX、APEX_OBJECT、APEX_TRIGGER、ARCHIVE、ASPNET、ASP、ASPX、BITCODE、BSP、BYTECODE、CFML、COBOL、CSHARP、DART、DOCKERFILE、FLIGHT、GENERIC、GO、HOCON、HTML、INI、JAVA、JAVA_PROPERTIES、JAVASCRIPT、JSP、JSPX、KOTLIN、MSIL、MXML、OBJECT、PHP、PLSQL、PYTHON、RUBY、RUBY_ERB、SCALA、SWIFT、SWC、SWF、TLD、SQL、TSQL、TYPESCRIPT、VB、VB6、VBSCRIPT、VISUAL_FORCE、VUE、およびXMLです。
-project-root	変換および分析フェーズで生成された中間ファイルを保存するディレクトリを指定します。Fortify Static Code Analyzerでは、このプロジェクトのルートディレクトリにある中間ファイルを広く活用します。場合によっては、このディレクトリをネットワークドライブではなくローカルストレージに配置すると、分析のパフォーマンスが向上します。 **同等のプロパティ名:** com.fortify.sca.ProjectRoot

分析オプション

次の表では、分析オプションについて説明します。

分析オプション	説明
-b *<build_id>*	前の変換コマンドで使用されるビルドIDを指定します。 **同等のプロパティ名:** com.fortify.sca.BuildID

分析オプション	説明
`-scan`	Fortify Static Code Analyzerで、指定したビルドIDのセキュリティ分析が実行されます。 **注:** このオプションは、同じsourceanalyzerコマンド内で`-scan-module`オプションと同時に使用しないでください。
`-scan-policy <policy_name>` | `-sc <policy_name>`	スキャンのスキャンポリシーを指定します。有効なポリシー名は、`classic`、`security`、および`devops`です。詳細については、「"分析へのスキャンポリシーの適用" ページ48」を参照してください。 **同等のプロパティ名:** `com.fortify.sca.ScanPolicy`
`-scan-module`	Fortify Static Code Analyzerで、指定したビルドIDのセキュリティ分析が個別のモジュールとして実行されます。 **注:** このオプションは、同じsourceanalyzerコマンド内で`-scan`オプションと同時に使用しないでください。 **同等のプロパティ名:** `com.fortify.sca.ScanScaModule`
`-include-modules`	プロジェクトスキャンに含めるビルドIDのカンマ区切りリストまたはコロン区切りリストで、以前に個別のモジュールとしてスキャンされたライブラリを指定します。 **同等のプロパティ名:** `com.fortify.sca.IncludeScaModules`
`-analyzers <analyzer_list>`	アナライザのコロン区切りリストまたはカンマ区切りリストで有効にするアナライザを指定します。有効なアナライザ名は、`buffer`、`content`、`configuration`、`controlflow`、`dataflow`、`nullptr`、`semantic`、および`structural`です。このオプションを使用して、セキュリティ要件に必要ないアナライザを無効にすることができます。 **同等のプロパティ名:** `com.fortify.sca.DefaultAnalyzers`
`-p <level>` | `-scan-precision <level>`	短縮ダイヤルを使用して、スキャン精度レベルを指定してプロジェクトをスキャンします。スキャン精度レベルが低いほど、スキャンのパフォーマンスは高速になります。有効な値は、1、2、3、および4です。詳細については、「"短縮ダイヤルを使用したスキャン速度の設定" ページ163」を参照してください。

分析オプション	説明
	同等のプロパティ名: `com.fortify.sca.PrecisionLevel`
`-project-root`	変換および分析フェーズで生成された中間ファイルを保存するディレクトリを指定します。Fortify Static Code Analyzerでは、このプロジェクトのルートディレクトリにある中間ファイルを広く活用します。場合によっては、このディレクトリをネットワークドライブではなくローカルストレージに配置すると、分析のパフォーマンスが向上します。 **同等のプロパティ名:** `com.fortify.sca.ProjectRoot`
`-project-template <file>`	スキャンに使用する問題テンプレートファイルを指定します。これにより、ローカルコンピュータのスキャンのみが影響を受けます。FPRをFortify Software Security Centerにアップロードする場合は、アプリケーションバージョンに割り当てられた問題テンプレートが使用されます。 **同等のプロパティ名:** `com.fortify.sca.ProjectTemplate`
`-quick`	`fortify-sca-quickscan.properties`ファイルを使用してプロジェクトのクイックスキャンを実行し、重大な問題や優先度の高い問題がないか調べます。この方法では、分析の詳細さは低下します。デフォルトでは、クイックスキャンの場合、Buffer AnalyzerとControl Flow Analyzerが無効になります。さらに、Quick Viewフィルタセットも適用されます。詳細については、「"クイックスキャン" ページ161」を参照してください。 **同等のプロパティ名:** `com.fortify.sca.QuickScanMode`
`-filter <file>`	結果フィルタファイルを指定します。詳細については、「"分析のフィルタリング" ページ180」を参照してください。 **同等のプロパティ名:** `com.fortify.sca.FilterFile`
`-bin <binary>` | `-binary-name <binary>`	スキャンするソースファイルのサブセットを指定します。ビルド時に名前付きバイナリにリンクされたソースファイルのみがスキャンに含められます。このオプションを複数回使用すると、スキャンに複数のバイナリが含まれるように指定できます。 **同等のプロパティ名:** `com.fortify.sca.BinaryName`

分析オプション	説明
`-disable-default-rule-type <type>`	デフォルトのRulepackで指定されたタイプのルールをすべて無効にします。このオプションを複数回使用すると、複数のルールタイプを指定できます。 `<type>`パラメータは、XMLタグからサフィックス`Rule`を除いた値です。たとえば、DataflowSourceRule要素の場合は`DataflowSource`を使用します。また、特性化ルールの特定のセクション (`Characterization:Control flow`、`Characterization:Issue`、`Characterization:Generic`など)を指定することもできます。 `<type>`パラメータでは、大文字と小文字が区別されません。
`-no-default-issue-rules`	問題が直接発生するデフォルトのRulepackでルールを無効にします。Fortify Static Code Analyzerでは、関数の動作を特徴付けるルールは引き続きロードされます。 **注:** これは、DataflowSink、Semantic、Controlflow、Structural、Configuration、Content、Statistical、Internal、Characterization:Issueルールタイプを無効にした場合と同等です。 **同等のプロパティ名:** `com.fortify.sca.NoDefaultIssueRules`
`-no-default-rules`	デフォルトのRulepackからルールがロードされないようにします。Fortify Static Code Analyzerでは、説明要素と言語ライブラリのRulepackが処理されますが、ルールは処理されません。 **同等のプロパティ名:** `com.fortify.sca.NoDefaultRules`
`-no-default-source-rules`	デフォルトのRulepackでソースルールを無効にします。 **注:** 特性化ソースルールは無効になりません。 **同等のプロパティ名:** `com.fortify.sca.NoDefaultSourceRules`
`-no-default-sink-rules`	デフォルトのRulepackでシンクルールを無効にします。 **注:** 特性化シンクルールは無効になりません。 **同等のプロパティ名:** `com.fortify.sca.NoDefaultSinkRules`
`-rules <file>`|	カスタムのRulepackまたはディレクトリを指定します。このオプションを複数

分析オプション	説明
<dir>	回使用すると、複数のRulepackファイルを指定できます。ディレクトリを指定すると、Fortify Static Code Analyzerでは、そのディレクトリ内の.binおよび.xmlの拡張子を持つすべてのファイルが含められます。 **同等のプロパティ名:** com.fortify.sca.RulesFile

出力オプション

次の表では、出力オプションについて説明します。これらのすべてのオプションを分析フェーズで適用します(-scanオプションも指定します)。build-label、build-project、およびbuild-versionオプションは、変換フェーズで指定できます。分析フェーズで再度指定すると、それらのオプションは上書きされます。

出力オプション	説明
-f *<file>*| -output-file *<file>*	分析結果が書き込まれるファイルを指定します。出力ファイルを指定しない場合は、Fortify Static Code Analyzerから端末に出力が書き込まれます。 **同等のプロパティ名:** com.fortify.sca.ResultsFile
-format *<format>*	出力形式を制御します。有効なオプションは、fpr、fvdl、fvdl.zip、text、およびautoです。デフォルトはautoであり、-fオプションで指定されたファイルのファイル名拡張子に基づいて出力形式が選択されます。 FVDLは、Fortify Static Code Analyzerの詳細な分析結果が含まれるXMLファイルです。これには、脆弱性の詳細、ルールの説明、コードスニペット、スキャン時に使用されるコマンドラインオプション、スキャンのエラーまたは警告が含まれています。 FPRは、FVDLファイル、およびスキャン時に使用されるソースコードのコピー、外部メタデータ、カスタムルール(該当する場合)などの追加情報が含まれる分析結果のパッケージです。Fortify Audit Workbenchは、自動的に.fpr拡張子に関連付けられます。 **注:** 結果証明書を使用する場合は、fpr形式を指定する必要があります。結果証明書については、『*OpenText™ Fortify Audit Workbenchユーザガイド*』を参照してください。

出力オプション	説明
	一部の情報がFPRファイルまたはNFDLファイルに含まれないようにすると、スキャン時間や出力ファイルサイズを改善できます。この表のその他のオプション、および「"FPRファイルの最適化" ページ166」を参照してください。 **同等のプロパティ名:** `com.fortify.sca.Renderer`
`-append`	`-f`オプションで指定されたファイルに結果を追加します。結果のFPRファイルには、以前のスキャンによる問題と現在のスキャンによる問題が含まれています。ビルド情報およびプログラムデータ(ソースとシンクのリスト)セクションもマージされます。このオプションを使用するには、出力ファイルの形式を`fpr`または`fvdl`にする必要があります。`-format`出力オプションについては、この表の説明を参照してください。 (分析中のプログラムに関する情報とは異なり)Fortify Software Security Contentの情報、コマンドラインオプション、システムプロパティ、警告、エラー、Fortify Static Code Analyzerの実行に関するその他の情報を含むエンジンデータはマージされません。エンジンデータは`-append`オプションを使用してマージされないため、Fortifyでは、`-append`生成された結果が証明されません。 このオプションを指定しない場合は、Fortify Static Code Analyzerで新しい結果がFPRファイルに追加され、`previous`の結果として古い結果にラベルが付けられます。 一般に、同時にアプリケーション全体を分析できない場合にのみ、`-append`オプションを使用します。 **同等のプロパティ名:** `com.fortify.sca.OutputAppend`
`-build-label` *<label>*	分析結果に含めるプロジェクトのラベルを指定します。このオプションは、変換フェーズや分析フェーズ中に含めることができます。Fortify Static Code Analyzerでは、このラベルはコード分析に使用されません。 **同等のプロパティ名:** `com.fortify.sca.BuildLabel`
`-build-project` *<project_name>*	分析結果に含めるプロジェクトの名前を指定します。このオプションは、変換フェーズや分析フェーズ中に含めることができます。Fortify Static Code Analyzerでは、この名前はコード分析に使用されません。 **同等のプロパティ名:** `com.fortify.sca.BuildProject`

出力オプション	説明
`-build-version <version>`	分析結果に含めるプロジェクトのバージョンを指定します。このオプションは、変換フェーズや分析フェーズ中に含めることができます。Fortify Static Code Analyzerでは、このバージョンはコード分析に使用されません。 **同等のプロパティ名:** `com.fortify.sca.BuildVersion`
`-disable-source-bundling`	ソースファイルを分析結果ファイルから除外します。 **同等のプロパティ名:** `com.fortify.sca.FPRDisableSourceBundling`
`-fvdl-no-descriptions`	Fortify Software Security Contentの説明を分析結果ファイルから除外します。 **同等のプロパティ名:** `com.fortify.sca.FVDLDisableDescriptions`
`-fvdl-no-enginedata`	エンジンデータを分析結果ファイルから除外します。エンジンデータには、Fortify Software Security Contentの情報、コマンドラインオプション、システムプロパティ、警告、エラー、およびFortify Static Code Analyzerの実行に関するその他の情報が含まれています。 **同等のプロパティ名:** `com.fortify.sca.FVDLDisableEngineData`
`-fvdl-no-progdata`	プログラムデータを分析結果ファイルから除外します。これにより、Fortify Audit Workbenchの関数ビューからテイントソース情報が削除されます。 **同等のプロパティ名:** `com.fortify.sca.FVDLDisableProgramData`
`-fvdl-no-snippets`	コードスニペットを分析結果ファイルから除外します。 **同等のプロパティ名:** `com.fortify.sca.FVDLDisableSnippets`

その他のオプション

次の表では、その他のオプションについて説明します。

その他のオプション	説明
`@<file>`	指定したファイルからコマンドラインオプションを読み込みます。 **注:** デフォルトでは、このファイルでJVMシステムエンコーディングが使用されます。`fortify-sca.properties`ファイルで指定された`com.fortify.sca.CmdlineOptionsFileEncoding`プロパティを使用して、エンコーディングを変更できます。このプロパティの詳細については、「"変換と分析フェーズのプロパティ" ページ187」を参照してください。
`-h`| `-?`| `-help`	コマンドラインオプションの概要を出力します。
`-debug`	Fortify Supportログファイルにデバッグ情報を含めます。この情報は、カスタマサポートのトラブルシューティングにのみ役立ちます。 **同等のプロパティ名:** `com.fortify.sca.Debug`
`-debug-verbose`	これは`-debug`オプションと同じですが、特に解析エラーに関する詳細が含まれています。 **同等のプロパティ名:** `com.fortify.sca.DebugVerbose`
`-debug-mem`	パフォーマンス情報をFortify Supportログの中に含めます。 **同等のプロパティ名:** `com.fortify.sca.DebugTrackMem`
`-verbose`	詳細ステータスメッセージをコンソールとFortify Supportログファイルに送信します。 **同等のプロパティ名:** `com.fortify.sca.Verbose`
`-logfile <file>`	Fortify Static Code Analyzerで作成されるログファイルを指定します。

その他のオプション	説明
	同等のプロパティ名: `com.fortify.sca.LogFile`
`-clobber-log`	sourceanalyzerが実行されるたびにログファイルを上書きするようにFortify Static Code Analyzerに指示します。このオプションを指定しない場合は、Fortify Static Code Analyzerでログファイルに情報が追加されます。 **同等のプロパティ名:** `com.fortify.sca.ClobberLogFile`
`-quiet`	コマンドラインの進行状況情報を無効にします。 **同等のプロパティ名:** `com.fortify.sca.Quiet`
`-version` | `-v`	Fortify Static Code Analyzerのバージョンと、Fortify Static Code Analyzerと一緒に含まれているさまざまな独立したモジュールのバージョンが表示されます(その他の機能はすべて、Fortify Static Code Analyzerに含まれています)。
`-autoheap`	システムで使用可能な物理メモリに基づいて、メモリの自動割り当てを有効にします。これはデフォルトのメモリ割り当て設定です。
`-Xmx<size>M|G`	Fortify Static Code Analyzerで使用されるメモリ使用量の最大値を指定します。 32 GB ~ 48 GBのヒープサイズは、内部JVMの実装上推奨されていません。この範囲のヒープサイズでは、32 GBよりもパフォーマンスが低下します。JVMでは、32 GB未満のヒープサイズが最適化されます。スキャンで32GBを超えるサイズが必要な場合は、64GB以上が必要になります。ガイドラインとして、メモリを大量に消費するその他のプロセスが実行されていないと仮定すると、使用可能なメモリの2/3を超えて割り当てないでください。 このオプションを指定する場合は、パフォーマンスが低下するため、物理的に使用可能なメモリよりも多く割り当てないでください。ガイドラインとして、メモリを大量に消費するその他のプロセスが実行されていないと仮定すると、使用可能なメモリの2/3を超えて割り当てないでください。 **注:** このオプションを指定すると、`-autoheap`オプションで取得したデフォルトのメモリ割り当てが上書きされます。

ディレクティブ

一度に1つのディレクティブのみを使用します。ディレクティブを変換コマンドや分析コマンドと組み合わせて使用しないでください。次の表で説明するディレクティブを使用して、以前の変換コマンドに関する情報のリストを表示します。

ディレクティブ	説明
`-clean`	すべてのFortify Static Code Analyzer中間ファイルとビルドレコードを削除します。ビルドIDを指定すると、そのビルドIDに関連するファイルとビルドレコードのみが削除されます。
`-show-binaries`	他のバイナリの生成時に使用されていない、作成済みのすべてのオブジェクトが表示されます。ビルドに完全に統合されている場合は、作成されたバイナリがすべて表示されます。
`-show-build-ids`	既知のすべてのビルドIDのリストが表示されます。
`-show-build-tree`	`-bin`オプションを使用してスキャンすると、バイナリの作成に使用されたファイルと、それらのファイルの作成に使用されたファイルをすべてツリーレイアウトで表示します。`-bin`オプションが存在しない場合は、バイナリごとにツリーが表示されます。 **注:** このオプションを使用すると、大量の情報が生成される可能性があります。
`-show-build-warnings`	`-b`オプションを指定して使用すると、変換フェーズで発生したエラーおよび警告がコンソールに表示されます。 **注:** Fortify Audit Workbenchでは、これらのエラーおよび警告が結果証明書タブにも表示されます。
`-show-files`	指定したビルドIDに含まれているファイルが表示されます。`-bin`オプションが存在する場合は、バイナリに渡されたソースファイルのみが表示されます。
`-show-loc`	`-b`オプションを指定して使用すると、変換されたコードの行数が表示されます。

LIMライセンスディレクティブ

Fortify Static Code Analyzerでは、LIMライセンスの使用状況を管理するためのディレクティブが提供されています。LIMのライセンスプール資格情報を保存またはクリアできます。指定したライセンスプールでデタッチされたリースが許可されている場合は、オフライン分析用にデタッチされたリースを要求(およびリリース)することもできます。

注: デフォルトでFortify Static Code AnalyzerはLIMサーバにHTTPS接続する必要があり、信頼された証明書が必要です。詳細については、「"信頼された証明書の追加" ページ42」を参照してください。

次の表で説明するディレクティブは、LIMで管理されるライセンスに使用します。

ディレクティブ	説明

`-store-license-pool-credentials "<lim_url>|<lim_pool_name>|<lim_pool_pwd>|<proxy_url>|<proxy_user>|<proxy_pwd>"`	Fortify Static Code AnalyzerでライセンスにLIMが使用されるように、LIMのライセンスプール資格情報を保存します。プロキシ情報はオプションです。Fortify Static Code Analyzerでは、このディレクティブで指定されたプールパスワードとプロキシ資格情報が`fortify-sca.properties`ファイルに暗号化されたデータとして保存されます。Fortify Static Code Analyzerのインストール後にライセンスプール資格情報が変更された場合は、このディレクティブを再度実行して新しい資格情報を保存できます。
	例:
	`sourceanalyzer -store-license-pool-credentials "https://<ip_address>:<port>|TeamA|mypassword"`
	関連付けられたプロパティ名:
	`com.fortify.sca.lim.Url`
	`com.fortify.sca.lim.PoolName`
	`com.fortify.sca.lim.PoolPassword`
	`com.fortify.sca.lim.ProxyUrl`
	`com.fortify.sca.lim.ProxyUsername`
	`com.fortify.sca.lim.ProxyPassword`

`-clear-license-pool-credentials`	LIMのライセンスプール資格情報を`fortify-sca.properties`ファイルから削除します。ライセンスプール資格情報が変更された場合は、このディレクティブを使用して削除してから、`-store-license-pool-credentials`ディレクティブを使用して新しい資格情報を保存できます。
`-request-detache`	このシステムで指定した期間(分単位)に排他的に使用されるように、LIMのライセンスプールからデタッチされたリースを要求します。これにより、企業のイントラネットから切断されている場合でもFortify Static Code Analyzerを実行できます。

ディレクティブ	説明
`d-lease <duration>`	**注:** このディレクティブを使用するには、デタッチされたリースが許可されるようにライセンスプールが設定されている必要があります。
`-release-detached-lease`	デタッチされたリースをライセンスプールに戻します。

ファイルとディレクトリの指定

ファイル指定子は、ワイルドカード文字を使用して長いファイルリストまたはディレクトリをFortify Static Code Analyzerに渡すことができる式です。Fortify Static Code Analyzerでは、2種類のワイルドカード文字が認識されます。単一のアスタリスク文字(*)はファイル名の一部に一致し、二重のアスタリスク文字(**)はディレクトリに再帰的に一致します。1つ以上のファイル、1つ以上のファイル指定子、またはファイルとファイル指定子の組み合わせを指定できます。

<files> | *<file_dir_specifiers>*

注: ファイル指定子は、C、C++、またはObjective-C++には適用されません。

次の表には、ファイルおよびディレクトリ指定子の例を示します。

ファイル/ディレクトリ指定子	説明
<dir> *<dir>*`/**/*`	名前付きディレクトリと任意のサブディレクトリ内またはディレクトリパラメータで使用時の名前付きディレクトリ内のすべてのファイルに一致します。
<dir>`/**/Example.java`	名前付きディレクトリまたは任意のサブディレクトリ内で見つかった`Example.java`という名前の任意のファイルに一致します。
<dir>`/*.java` *<dir>*`/*.jar`	名前付きディレクトリ内で見つかった指定した拡張子付きの任意のファイルに一致します。
<dir>`/**/*.kt` *<dir>*`/**/*.jar`	名前付きディレクトリまたは任意のサブディレクトリ内で見つかった指定した拡張子付きの任意のファイルに一致します。
<dir>`/**/beta/**`	パスに`beta`が含まれている(ファイル名として`beta`を含む)名前付きディレクトリ内で見つかったすべてのディレクトリおよびファ

ファイル/ディレクトリ指定子	説明
	イルに一致します。
`<dir>/**/classes/`	名前付きディレクトリおよび任意のサブディレクトリ内で見つかった`classes`という名前のすべてのディレクトリとファイルに一致します。
`**/test/**`	パスに`test`要素が含まれている(ファイル名として`test`を含む)現在のディレクトリツリー内のすべてのファイルに一致します。
`**/webgoat/*`	現在のディレクトリツリーにある任意の`webgoat`ディレクトリ内のすべてのファイルに一致します。 一致する: • `/src/main/java/org/owasp/webgoat` • `/test/java/org/owasp/webgoat` 一致しない(`assignments`ディレクトリが一致しない) • `/test/java/org/owasp/webgoat/assignments`

注: Windowsおよび多くのLinuxのシェルでは、アスタリスク文字(*)を含むパラメータが自動的に展開されます。そのため、ファイル指定子の式を引用符で囲む必要があります。また、Windowsでは、スラッシュ(/)の代わりにバックスラッシュ(\)をディレクトリ区切り文字として使用することもできます。

第19章: コマンドラインツール

Fortify Static Code Analyzerコマンドラインツールを使用すると、Fortify Security Contentの管理、インストール後の設定、スキャンの監視を実行できます。これらのツールは、*<sca_install_dir>*/binにあります。Windows用のツールは、.batまたは.cmdファイルとして提供されます。次の表では、Fortify Static Code Analyzerと一緒にインストールされるコマンドラインツールについて説明します。

注: デフォルトでは、Fortify Static Code Analyzerツールのログファイルは次のディレクトリに書き込まれます。

- Windows: C:\Users*<username>*\AppData\Local\Fortify*<tool_name>*-*<version>*\log
- Windows以外: *<userhome>*/.fortify/*<tool_name>*-*<version>*/log

ツール	説明	詳細情報
fortifyupdate	インストールされているセキュリティコンテンツを現在のバージョンと比較し、必要に応じてアップデートします	"セキュリティコンテンツの更新について" 次のページ
FPRUtility	このツールを使用すると、次の処理を実行できます。 • 監査されたプロジェクトをマージする • FPR署名を検証する • FPRファイルからの情報を表示する • ソースコードファイルと監査プロジェクトをFPRファイルに結合または分割する • FPRを変更する	*OpenText™ OpenText™ Fortify Static Code Analyzer アプリケーションおよびツールガイド*
scapostinstall	このツールを使用すると、Fortify Static Code Analyzerの以前のバージョンのpropertiesファイルを移行したり、ロケールを指定したり、セキュリティコンテンツの更新やFortify Software Security Centerに使用するプロキシサーバを指定したりできます。	"インストール後処理ツールの実行" ページ39
SCAState	分析フェーズ中にJVMに関する状態分析情報を提供します。	"Fortify Static Code Analyzerスキャンステータスの確認" ページ155

このセクションでは、次のトピックについて説明します。

セキュリティコンテンツの更新について

fortifyupdateコマンドラインツールを使用して、最新のFortify Secure Coding Rulepacksとメタデータを Fortifyからダウンロードできます。

fortifyupdateツールは、Fortifyインストールに含まれている既存のセキュリティコンテンツに関する情報を収集し、この情報を使用してFortifyルールパック更新サーバに問い合わせます。サーバは新しいセキュリティコンテンツまたは更新されたセキュリティコンテンツを返し、古いセキュリティコンテンツをFortify Static Code Analyzerインストールから削除します。インストールが最新の場合は、その旨のメッセージが表示されます。

セキュリティコンテンツの更新

fortifyupdateコマンドラインツールを使用して、セキュリティコンテンツをダウンロードするか、セキュリティコンテンツのローカルコピーをインポートします。このツールは*<sca_install_dir>*/binディレクトリにあります。

このツールのデフォルトの読み込みタイムアウトは180秒です。タイムアウト設定を変更するには、server.properties環境設定ファイルにrulepackupdate.SocketReadTimeoutSecondsプロパティを追加します。詳細については、『*OpenText™ OpenText™ Fortify Static Code Analyzerアプリケーションおよびツールガイド*』を参照してください。

fortifyupdateの基本的なコマンドライン構文を次の例に示します。

```
fortifyupdate [<options>]
```

Fortifyルールパック更新サーバからの最新のFortify Secure Coding Rulepacksおよび外部メタデータを使用してFortify Static Code Analyzerインストールを更新するには、次のコマンドを入力します。

```
fortifyupdate
```

ローカルシステムからセキュリティコンテンツを更新するには、次のコマンドを入力します。

```
fortifyupdate -import <my_local_rules>.zip
```

資格情報を使用してFortify Software Security Centerサーバからセキュリティコンテンツを更新するには、次のコマンドを入力します。

```
fortifyupdate -url <ssc_url> -sscUser <username> -sscPassword <password>
```

fortifyupdateコマンドラインオプション

次の表では、fortifyupdateオプションについて説明します。

fortifyupdateオプション	説明
`-acceptKey`	公開鍵を受諾する場合に指定します。このオプションを指定すると、公開鍵の入力を求めるプロンプトは表示されません。`-url`オプションを使用して非標準の場所から更新する場合は、このオプションを使用して公開鍵を受諾します。
`-acceptSSLCertificate`	サーバによって提供されるSSL証明書を使用する場合に指定します。
`-import <file>.zip`	セキュリティコンテンツを含むZIPファイルをインポートします。デフォルトでは、Rulepackが`<sca_install_dir>/Core/config/rules`ディレクトリにインポートされます。
`-coreDir <dir>`	fortifyupdateで更新が保存されるコアディレクトリを指定します。このオプションが指定されていない場合、fortifyupdateによる更新は*<sca_install_dir>*で実行されます。 **重要** `<sca_install_dir>/config/keys`フォルダのコンテンツをコピーして、このディレクトリ内の`config/keys`フォルダに貼り付けしてから、`fortifyupdate`を実行してください。
`-includeMetadata`	外部メタデータのみを更新することを指定します。
`-includeRules`	ルールパックのみを更新することを指定します。
`-locale <locale>`	ロケールを指定します。指定したロケールにセキュリティコンテンツが存在しない場合は、英語がデフォルトです。有効な値は次のとおりです。 • `en`（英語） • `es`（スペイン語） • `ja`（日本語） • `ko`（韓国語）

fortifyupdateオプション	説明
	• pt_BR (ポルトガル語(ブラジル)) • zh_CN (簡体字中国語) • zh_TW (繁体字中国語) **注:** この値では、大文字と小文字を区別しません。 *または、fortify.properties環境設定ファイルでセキュリティコンテンツ更新のデフォルトロケールを指定できます。詳細については、『OpenText™ OpenText™ Fortify Static Code Analyzerアプリケーションおよびツールガイド』を参照してください。*
-proxyhost <*host*>	プロキシサーバのネットワーク名またはPアドレスを指定します。
-proxyport <*port*>	プロキシサーバのポート番号を指定します。
-proxyUsername <*username*>	プロキシサーバが認証を必要とする場合、ユーザ名を指定します。
-proxyPassword <*password*>	プロキシサーバが認証を必要とする場合、パスワードを指定します。
-showInstalledRules	現在インストールされているRulepackを、カスタムルールとカスタムメタデータを含めて表示します。
-showInstalledExternalMetadata	現在インストールされている外部メタデータを表示します。
-url <*url*>	セキュリティコンテンツをダウンロードするURLを指定します。デフォルトのURLは https://update.fortify.com、または server.properties環境設定ファイル内の rulepackupdate.serverプロパティに設定された値です。 server.properties環境設定ファイルの詳細については、『OpenText™ OpenText™ Fortify Static Code Analyzerアプリケーションおよびツールガイド』を参照してください。

fortifyupdateオプション	説明
	Fortify Software Security Center URLを指定することで、Fortify Software Security Centerサーバからセキュリティコンテンツをダウンロードできます。
-urlオプションを使用してFortify Software Security Centerからセキュリティコンテンツを更新する場合は、次のいずれかのタイプの資格情報を指定します。	
-sscUsername -sscPassword	ユーザ名とパスワードでFortify Software Security Centerユーザアカウントを指定します。
-sscAuthToken	タイプがUnifiedLoginToken、CIToken、またはToolsConnectTokenのFortify Software Security Center認証トークンを指定します。

Fortify Static Code Analyzerスキャンステータスの確認

SCAStateツールを使用して、分析フェーズ中に最新の状態分析情報を確認します。

Fortify Static Code Analyzerの状態を確認するには

1. Fortify Static Code Analyzerスキャンを開始します。
2. 別のコマンドウィンドウを開きます。
3. コマンドプロンプトで次のコマンドを入力します。

```
SCAState [<options>]
```

SCAStateコマンドラインオプション

次の表では、SCAStateオプションについて説明します。

SCAStateオプション	説明
-a | --all	使用可能なすべての情報を表示します。
-debug	SCAStateの振る舞いをデバッグする場合に役立つ情報を表示します。
-ftd | --full-thread-dump	各スレッドのスレッドダンプを出力します。

SCAStateオプション	説明
-h | --help	SCAStateツールのヘルプ情報を表示します。
-hd <*filename*> | --heap-dump <*filename*>	ヒープダンプの書き込みファイルを指定します。このファイルはリモートスキャンの作業ディレクトリを基準に解釈されます。これは必ずしもSCAStateを実行しているディレクトリとは限りません。
-liveprogress	実行中のスキャンの現在のステータスを表示します。これはデフォルトです。可能であれば、この情報は別の端末ウィンドウに表示されます。
-nogui	別のウィンドウではなく、現在の端末ウィンドウにFortify Static Code Analyzerの状態情報を表示します。
-pi | --program-info	スキャン中のソースコードに関する情報(含まれるソースファイルと関数の数など)が表示されます。
-pid <*process_id*>	現在実行中のFortify Static Code Analyzerプロセスを指定します。複数のFortify Static Code Analyzerプロセスが同時に実行されている場合は、このオプションを使用します。 Windowsシステム上でプロセスIDを取得するには、次の手順を実行します。 1. コマンドウィンドウを開きます。 2. コマンドプロンプトで`tasklist`と入力します。 プロセスのリストが表示されます。 3. リスト内で`java.exe`プロセスを見つけ、PIDをメモします。 Linuxシステム上でプロセスIDを取得するには、次の手順を実行します。 • コマンドプロンプトで`ps aux | grep sourceanalyzer`と入力します。
-progress	コマンドが発行された時点までのスキャン情報を表示します。これには、経過時間、分析の現在のフェーズ、および取得済みの結果数が含まれます。
-properties	環境設定を表示します(パスワードなどの機密情報は含まれません)。

SCAStateオプション	説明
`-scaversion`	現在実行中のsourceanalyzerのFortify Static Code Analyzerバージョン番号を表示します。
`-td`| `--thread-dump`	メインスキャンスレッドのスレッドダンプを出力します。
`-timers`	Fortify Static Code Analyzerで計測されるタイマおよびカウンタからの情報を表示します。
`-version`	SCAStateバージョンを表示します。
`-vminfo`	JVM標準MXBeansで提供される、ClassLoadingMXBean、CompilationMXBean、GarbageCollectorMXBeans、MemoryMXBean、OperatingSystemMXBean、RuntimeMXBean、およびThreadMXBeanに関する統計情報を表示します。
<none>	スキャン進行状況情報を表示します(これは`-progress`と同じです)。

注: Fortify Static Code Analyzerでは、Javaプロセス情報をTMPシステム環境変数の場所に書き込みます。Windowsシステムでは、TMPシステム環境変数の場所は `C:\Users\<username>\AppData\Local\Temp`です。別の場所を指すようにこのTMPシステム環境変数を変更した場合、SCAStateは`sourceanalyzer` Javaプロセスを見つけ出せず、予期された結果を返しません。この問題を解決するには、新しいTMPの場所に一致するようにTMPシステム環境変数を変更します。Fortifyでは、SCAStateをWindows管理者として実行することを推奨します。

第20章: パフォーマンスの改善

この章では、Fortify Static Code Analyzerでさまざまな種類のコードベースを分析する際に、メモリ使用量とパフォーマンスを最適化するためのガイドラインとヒントについて説明します。

このセクションでは、次のトピックについて説明します。

ウィルス対策ソフトウェア

ウィルス対策ソフトウェアを使用すると、Fortify Static Code Analyzerのパフォーマンスが悪影響を受ける可能性があります。Fortifyでは、スキャン時間が長い場合に、ウィルス対策ソフトウェアスキャンから内部のFortify Static Code Analyzerファイルを一時的に除外することが推奨されています。ソースコードが存在するディレクトリでも同じ操作を実行できますが、Fortify分析時のパフォーマンスの影響は内部ディレクトリの場合よりも小さいです。

デフォルトでは、Fortify Static Code Analyzerの内部ファイルが次の場所に作成されます。

- Windows: `c:\Users\<username>\AppData\Local\Fortify\sca<version>`
- Windows以外: `<userhome>/.fortify/sca<version>`

ここで、*<version>*は使用しているFortify Static Code Analyzerのバージョンです。

ハードウェアに関する考慮事項

さまざまなソースコードがあるため、メモリ使用量やスキャン時間を正確に予測することはできません。次のさまざまな要因によってメモリ使用量やパフォーマンスが影響を受けます。

- コードのタイプ
- コードベースのサイズと複雑さ
- 使用される補助言語(JSP、JavaScript、HTMLなど)
- 脆弱性の数
- 脆弱性のタイプ(使用されるアナライザ)

Fortifyでは、実際のアプリケーションスキャンの結果に基づいて、次の「最適な推測」というハードウェアの推奨事項が開発されました。次の表には、アプリケーションの複雑性に基づいた推奨事項のリストを示します。一般に、使用可能なコアの数を増やすと、スキャン時間が短縮される可能性があります。

アプリケーションの複雑さ	CPUコア数	RAM (GB)	説明
単純	4	16	サーバまたはデスクトップ上で実行されるスタンドアロンシステム(バッチジョブやコマンドラインツールなど)。
中間	8	32	複雑なコンピュータモデルで動作するスタンドアロンシステム(税額計算システムやスケジューリングシステムなど)。
複雑	16	128	トランザクションデータ処理を備えた3階層ビジネスシステム(財務システムや商用Webサイトなど)。
非常に複雑	32	256	コンテンツを配信するシステム(アプリケーションサーバ、データベースサーバ、コンテンツ管理システムなど)。

注: TypeScriptスキャンおよびJavaScriptスキャンでは、分析時間が大幅に長くなります。アプリケーション内のコード行の合計がTypeScriptまたはJavaScriptの20%を超える場合は、2番目に高い推奨事項を使用します。

システム要件については、ドキュメント『*Fortify ソフトウェアシステム要件*』を参照してください。ただし、大規模で複雑なアプリケーションの場合は、Fortify Static Code Analyzerでより高い能力を持つハードウェアが必要です。その内容は次のとおりです。

- **ディスクI/O** - Fortify Static Code Analyzerは I/O集中型であるため、ハードドライブが高速になるほど、多くのI/Oトランザクションが節約されます。Fortifyでは、7,200 RPMドライブが推奨されていますが、10,000 RPMドライブ(WD Raptorなど)またはSSDドライブの方が適切です。
- **メモリ** - 最適なパフォーマンスを得るために必要なメモリの量を決定する方法の詳細については、「"メ

モリのチューニング" ページ173」を参照してください。

- **CPU** - Fortifyでは、2.1 GHz以上のプロセッサが推奨されています。

サンプルスキャン

以下のサンプルスキャンは、専用の仮想マシン上でFortify Static Code Analyzerバージョン24.2.0を使用して実行されました。これらのスキャンは、Fortify Security Content 2024 Update 1を使用して実行されました。次の表は、いくつかの一般的なオープンソースプロジェクトで期待できるスキャン時間を示しています。

言語	プロジェクト名	変換時間 (mm:ss)	分析(スキャン)時間(mm:ss)	問題の合計数	LOC	システム設定
.NET (C#)	SharpZipLib	1:51	2:03	799	41,773	8個のCPUと32GBのRAMを搭載したWindows VM
C/C++	nasm 0.98.38	0:38	3:30	871	35,960	8個のCPUと32GBのRAMを搭載したLinux VM
Java	WebGoat 8	0:39	1:08	284	23,412	
Java	WordPress for Android	0:14	1:47	535	35,167	
JavaScript	Hackademic-next	1:08	2:49	775	212,510	
PHP	CakePHP	0:21	3:01	5,720	136,463	
PHP	phpBB 3	0:39	2:35	1,350	206,728	
Python 3	numpy-1.13.3	2:31	9:37	272	562,731	
Swift	MediaBrowser	0:24	1:50	10	17,611	4個のCPUと16 GBのRAMを搭載したmacOS VM
TypeScript	prisma	1:31	5:00	88	148,730	8個のCPUと32GBのRAMを搭載したLinux VM

チューニングオプション

Fortify Static Code Analyzerでは、複雑なプロジェクトの処理に長い時間がかかる場合があります。次のようにさまざまなフェーズで時間が費やされます。

- 変換
- 分析

Fortify Static Code Analyzerでは、大きな分析結果ファイル(FPR)が生成される可能性があります。その結果、監査とFortify Software Security Centerへのアップロードに長い時間がかかる場合があります。このフェーズは、次のように呼ばれます。

- 監査/アップロード

次の表では、時間のかかるさまざまなフェーズでパフォーマンスを向上させる方法に関するヒントを示します。

フェーズ	オプション	説明	詳細情報
変換	`-export-build-session` `-import-build-session`	さまざまなコンピュータでの変換とスキャン	"モバイルビルドセッション" ページ46
分析	`-quick`	クイックスキャンの実行	"クイックスキャン" 下
分析	`-scan-precision`	スキャン精度の設定	"短縮ダイヤルを使用したスキャン速度の設定" ページ163
分析	`-bin`	バイナリに関連するファイルのスキャン	"コードベースの分割" ページ164
分析	`-Xmx<size>M|G`	最大ヒープサイズの設定	"メモリのチューニング" ページ173
分析	`-Xss<size>M|G`	各スレッドのスタックサイズの設定	"メモリのチューニング" ページ173
分析 監査/アップロード	`-filter <file>`	フィルタファイルを使用したフィルタの適用	"フィルタファイルの使用" ページ166
分析 監査/アップロード	`-disable-source-bundling`	FPRファイルからのソースファイルの除外	"FPRからのソースコードの除外" ページ167

クイックスキャン

クイックスキャンモードを使用すると、プロジェクトを高速にスキャンして、重大な問題や優先度の高い問題を特定できます。Fortify Static Code Analyzerは、分析の深さを減らすことでスキャンを高速に実行します。さらに、Quick Viewフィルタセットも適用されます。クイックスキャン設定は設定可能です。クイックスキャンモードの設定について詳しくは、"fortify-sca-quickscan.properties" ページ214を参照してください。

クイックスキャンは、評価を通じて多くのアプリケーションを取得し、問題をすばやく見つけて修復を開始できるようにするための最適な方法です。パフォーマンスがどの程度向上するかは、アプリケーションの複雑さとサイズによって異なります。このスキャンはフルスキャンよりも高速ですが、結果セットの堅牢性は劣ります。可能な限りフルスキャンを実行することをお勧めします。

リミッタ

Fortify Static Code Analyzerの分析の深さは、利用可能なリソースに依存する場合があります。Fortify Static Code Analyzerは、複雑性のメトリックを使用して、これらのリソースと検出可能な脆弱性の数のバランスを取ります。このため、Fortify Static Code Analyzerが十分なリソースを使用できないと見なされる場合は、特定の機能を実行しないことがあります。

Fortify Static Code Analyzerでは、ユーザがFortify Static Code Analyzerのリミッタプロパティを使用して「カットオフ」ポイントを制御できます。アナライザごとに異なるリミッタが用意されています。クイックスキャンを使用して、これらのリミッタの事前定義されたセットを実行できます。リミッタについて詳しくは、"fortify-sca-quickscan.properties" ページ214を参照してください。

クイックスキャンモードを有効にするには、`-scan`オプションとともに`-quick`オプションを使用します。クイックスキャンモードを有効にすると、Fortify Static Code Analyzerは標準の*<sca_install_dir>*`/Core/config/fortify-sca.properties`ファイルに加えて*<sca_install_dir>*`/Core/config/fortify-sca-quickscan.properties`ファイルのプロパティを適用します。`fortify-sca-quickscan.properties`ファイルを編集して、Fortify Static Code Analyzerが使用するリミッタを調整できます。`fortify-sca.properties`を変更すると、クイックスキャンの動作にも影響します。Fortifyでは、クイックスキャンモードでパフォーマンスを調整し、正確なスキャンを実行するためにフルスキャンをデフォルト設定のままにすることを推奨します。クイックスキャンモードのプロパティの詳細については、「"Fortify Static Code Analyzerのプロパティファイル" ページ185」を参照してください。

クイックスキャンとフルスキャンの使用

- **フルスキャンを定期的に実行する** - フルスキャンは、クイックスキャンモードでは検出されない問題を検出できる可能性があるため、定期的に実行することが重要です。ソフトウェアのイテレーションごとに、少なくとも1回はフルスキャンを実行します。可能であれば、週末など、開発ワークフローが中断されないタイミングで定期的にフルスキャンを実行します。
- **クイックスキャンとフルスキャンを比較する** - クイックスキャンの正確な影響を評価するには、同じコードベースでクイックスキャンとフルスキャンを実行します。Fortify Audit Workbenchでクイックスキャンの結果を開き、それをフルスキャンにマージします。問題を**新しい問題**でグループ化して、フルスキャンで検出されたがクイックスキャンでは検出されなかった問題のリストを生成します。
- **クイックスキャンとFortify Software Security Center** — フルスキャンの結果が上書きされるのを防ぐため、デフォルトではFortify Software Security Centerは、クイックスキャンモードでスキャンされてアップロードされたFPRファイルを無視します。ただし、Fortify Software Security Centerのアプリケーション版は、クイックスキャンでスキャンされたFPRファイルを処理するように設定できます。詳しくは、*OpenText™ Fortify Software Security Centerユーザガイド*の分析結果の処理ルールを参照してください。

短縮ダイヤルを使用したスキャン速度の設定

分析フェーズの精度レベルを指定して、スキャンの速度と深さを設定できます。これらの精度レベルを使用して、たとえば、パイプラインに適合するようにスキャン時間を調整すると、開発者がコードの作業を続けながら一連の脆弱性をすばやく見つけ出すことができます。短縮ダイヤル設定を使用したスキャンは、フルスキャンよりも高速ですが、結果セットの堅牢性は劣ります。可能な限りフルスキャンを実行することをお勧めします。

精度レベルでは、設定プロパティを各レベルに関連付けることで、スキャンの深さと精度を制御します。各レベルの設定プロパティファイルは、`<sca_install_dir>/Core/config/scales`ディレクトリにあります。レベルごとに1つのファイル(`level-<precision_level>.properties`)があります。これらのファイルの設定を変更して、独自の精度レベルを作成できます。

メモ

- Fortify Software Security Centerでは、4未満の精度レベルで作成され、アップロードされた分析結果がデフォルトでブロックされます。ただし、これらの精度レベルでスキャンされ、アップロードされた監査プロジェクトが処理されるように、Fortify Software Security Centerのアプリケーションバージョンを設定できます。
- 短縮ダイヤルスキャンをフルスキャンとマージすると、アプリケーションに残っている以前のスキャンの問題が削除される場合があります(フルスキャンを実行すると再び検出されます)。

スキャンの短縮ダイヤル設定を指定するには、次の例に示すように、スキャンフェーズに`-scan-precision`(または`-p`)オプションを含める必要があります。

```
sourceanalyzer -b MyProject -scan -scan-precision <level> -f MyResults.fpr
```

注: 短縮ダイヤル設定と`-quick`オプションを同じスキャンコマンドで使用することはできません。

次の表では、4つの精度レベルについて説明します。

精度レベル	説明
1	これは最も高速なスキャンであり、数個のファイルをスキャンする場合にお勧めします。この精度レベルのスキャンでは、Buffer Analyzer、Control Flow Analyzer、Dataflow Analyzer、およびNull Pointer Analyzerがデフォルトで無効になります。
2	この精度レベルのスキャンでは、すべてのアナライザがデフォルトで有効になります。リミッタを減らして実行すると、スキャンの実行速度が向上します。その結果、検出される問題の数が少なくなります。
3	この精度レベルでは、中間開発スキャンの速度が最大50%向上します(報告される問題も

精度レベル	説明
	減少します)。具体的には、このレベルではJavaやC/C++などの型付け言語のスキャン時間が向上します。
4	これはフルスキャンと同じです。

fortify-sca.propertiesファイルのcom.fortify.sca.PrecisionLevelプロパティを使用してスキャン精度レベルを指定することもできます。例:

```
com.fortify.sca.PrecisionLevel=1
```

コードベースの分割

大規模なプロジェクトを独立したモジュールに分割すると、効率が向上します。たとえば、ポータルアプリケーションを構成する複数のモジュールが互いに独立しているか、モジュール間のやり取りが少ない場合は、モジュールを個別に変換およびスキャンできます。ただし、何らかのやり取りがある場合は、データフローの問題を検出できない可能性があることに注意する必要があります。

C/C++の場合は、-binオプションとともに-scanオプションを使用すると、スキャン時間が短縮される可能性があります。バイナリファイルをパラメータとして渡す必要があります(-bin *<filename>*.exe -scanまたは-bin *<filename>*.dll -scanなど)。Fortify Static Code Analyzerでは、バイナリに関連するファイルが検索され、スキャンされます。これは、makefileに複数のバイナリがある場合に便利です。

次の表は、コードベースを分割する際に役立つFortify Static Code Analyzerコマンドラインオプションのリストです。

オプション	説明
-bin *<binary>*	スキャンするソースファイルのサブセットを指定します。ビルド時に名前付きバイナリにリンクされたソースファイルのみがスキャンに含められます。このオプションを複数回使用すると、スキャンに複数のバイナリが含まれるように指定できます。
-show-binaries	他のバイナリの生成時に作成されたが、使用されていないオブジェクトが表示されます。ビルドに完全に統合されている場合は、作成されたバイナリがすべて表示されます。
-show-build-tree	-binオプションとともに使用すると、バイナリを作成するために使用されるファイルと、それらのファイルをツリーレイアウトで作成するために使用されるファイルがすべて表示されます。-binオプションが存在しない場合、Fortify Static Code Analyzerではバイナリごとにツリーが表示されます。

アナライザと言語の制限

場合によっては、1つのアナライザの実行や特定の言語の分析に膨大な時間が費やされることがあります。このアナライザや言語は、セキュリティ要件にとって重要ではない可能性があります。実行するアナライザとFortify Static Code Analyzerで変換する言語を制限できます。

アナライザの無効化

特定のアナライザを無効にするには、スキャン時にFortify Static Code Analyzerに`-analyzers`オプションを追加して、有効にするアナライザのカンマ区切りまたはコロン区切りリストを指定する必要があります。`-analyzers`オプションの有効なパラメータ値は、`buffercontentconfiguration`、`controlflow`、`dataflow`、`nullptr`、`structural`、および`semantic`です。

たとえば、Dataflow、Control Flow、およびBufferアナライザのみを含むスキャンを実行するには、次のスキャンコマンドを使用します。

```
sourceanalyzer -b MyProject -analyzers dataflow:controlflow:buffer -scan -f
MyResults.fpr
```

Fortify Static Code Analyzerのプロパティファイル(`<sca_install_dir>/Core/config/fortify-sca.properties`)に`com.fortify.sca.DefaultAnalyzers`を設定して、同じ操作を実行することもできます。たとえば、前のスキャンコマンドと同等の結果を得る場合は、プロパティファイルに次の値を設定します。

```
com.fortify.sca.DefaultAnalyzers=dataflow:controlflow:buffer
```

言語の無効化

特定の言語を無効にするには、除外する言語のリストを指定する`-disable-language`オプションを変換フェーズに含めます。有効な言語の値は、`abap`、`actionscript`、`apex`、`cfml`、`cobol`、`configuration`、`cpp`、`dart`、`dotnet`、`golang`、`java`、`javascript`、`jsp`、`kotlin`、`objc`、`php`、`plsql`、`python`、`ruby`、`scala`、`sql`、`swift`、`tsql`、`typescript`、および`vb`です。

たとえば、SQLおよびPHPファイルを除外する変換を実行するには、次のコマンドを使用します。

```
sourceanalyzer -b MyProject <src_files> -disable-language sql:php
```

Fortify Static Code Analyzerのプロパティファイル(`<sca_install_dir>/Core/config/fortify-sca.properties`)で`com.fortify.sca.DISabledLanguages`プロパティを設定して、言語を無効にすることもできます。たとえば、前の変換コマンドと同等の結果を得る場合は、プロパティファイルに次の値を設定します。

```
com.fortify.sca.DISabledLanguages=sql:php
```

FPRファイルの最適化

この章では、監査結果(FPR)ファイルに関連するパフォーマンスの問題を処理する方法について説明します。これには、スキャン時間の短縮、FPRファイルサイズの削減、および大きなFPRファイルを開くためのヒントが含まれます。

フィルタファイルの使用

ファイルを使用して、特定の脆弱性インスタンス、ルール、および脆弱性カテゴリを分析結果から除外できます。特定の問題カテゴリまたはルールが特定のスキャンに関連していないと判断した場合は、それらをFPRに追加しないようにFortify Static Code Analyzerを設定できます。フィルタファイルを使用すると、スキャン時間と分析結果ファイルのサイズの両方を削減できます。

たとえば、指定したファイルを読み込むだけの単純なプログラムをスキャンする場合、パス操作の問題は機能の一部として計画されておらず、確認する必要がないことがあります。パス操作の問題をフィルタで除外するには、次の1行を含むファイルを作成します。

```
Path Manipulation
```

このファイルを`filter.txt`という名前で保存します。次の例に示すように、分析フェーズで`-filter`オプションを使用します。

```
sourceanalyzer -b MyProject -scan -filter filter.txt -f MyResults.fpr
```

`MyResults.fpr`の分析出力には、パス操作カテゴリの問題は含まれません。フィルタファイルの詳細と例については、「"フィルタファイルによる問題の除外" ページ180」を参照してください。

フィルタセットの使用

問題テンプレートのフィルタによって、Fortify Static Code Analyzerの結果の表示方法が決まります。フィルタに加えてフィルタセットを使用すると、同時に使用するフィルタを選択できます。各FPRには、関連付けられた問題テンプレートがあります。フィルタセットを使用すると、問題テンプレートのフィルタで指定した条件に基づいて問題の数を減らすことができます。これにより、FPRのサイズを大幅に削減できます。

これを行うには、Fortify Audit Workbenchを使用してフィルタセット内にフィルタを作成してから、そのフィルタセットと、そのフィルタセットが含まれている問題テンプレートを使用してFortify Static Code Analyzerスキャンを実行します。フィルタセットの作成方法の詳細と基本的な例については、「"フィルタセットによる問題の除外" ページ183」を参照してください。

注: フィルタセットで問題をフィルタリングすると、FPRのサイズは削減されますが、通常、スキャン時間は短縮されません。Fortify Static Code Analyzerは、問題を計算してFPRファイルに書き込むかどうかを判断した後で、フィルタセットを調べます。フィルタセット内のフィルタによって、Fortify Static Code Analyzerがロードするルールタイプが決まります。

FPRからのソースコードの除外

FPRからソースコード情報を除外することで、FPRファイルのサイズを削減できます。これは、大きなソースファイルやコードベースの場合に特に便利です。通常、この方法を使用しても、小さなソースファイルのスキャン時間は短縮されません。

Fortify Static Code AnalyzerでFPRにソースコードが含まれないようにするために使用できるプロパティがあります。*<sca_install_dir>*/Core/config/fortify-sca.propertiesいずれのプロパティもファイル内で設定するか、コマンドラインでオプションを指定することができます。次の表では、これらの設定について説明します。

プロパティ名	説明
com.fortify.sca. FPRDisableSourceBundling=true **コマンドラインオプション:** -disable-source-bundling	FPRからソースコードを除外します。
com.fortify.sca. FVDLDisableSnippets=true **コマンドラインオプション:** –fvdl-no-snippets	FPRからコードスニペットを除外します。

次のコマンドライン例では、両方のオプションを使用して、FPRからソースコードとコードスニペットの両方を除外しています。

```
sourceanalyzer -b MyProject -disable-source-bundling -fvdl-no-snippets -
scan -f MySourcelessResults.fpr
```

FPRファイルサイズの削減

FPRファイルのサイズを減らすには、いくつかの方法があります。結果に影響を与えずにこれを実現する最も簡単な方法は、「"FPRからのソースコードの除外" 上」で説明されているように、FPRからソースコードを除外することです。また、マージされたFPRのサイズをFPRUtilityで削減することもできます(『*OpenText™ OpenText™ Fortify Static Code Analyzer アプリケーションおよびツールガイド*』を参照)。

FPRから除外する項目を選択するために使用できるプロパティが他にもいくつかあります。これらのプロパティを*<sca_install_dir>*/Core/config/fortify-sca.propertiesファイルで設定するか、分析(スキャン)フェーズのコマンドラインでオプションを指定することができます。

プロパティ名	説明
com.fortify.sca. FPRDisableMetatable	FPRからメタテーブルを除外します。Fortify Audit Workbenchは、メタテーブルを使用して関数ビューに

プロパティ名	説明

`=true`
コマンドラインオプション:
`-disable-metatable`
	情報をマップします。

`com.fortify.sca.FVDLDisableDescriptions=true`
コマンドラインオプション:
`-fvdl-no-descriptions`
	FPRからルールの説明を除外します。カスタムの説明を使用しない場合は、Fortify Taxonomy (https://vulncat.fortify.com)の説明が使用されます。

`com.fortify.sca.FVDLDisableEngineData=true`
コマンドラインオプション:
`-fvdl-no-enginedata`
	FPRからエンジンデータを除外します。これは、Fortify Audit WorkbenchでファイルをFPRに多くの警告が含まれている場合に便利です。
	注: FPRからエンジンデータを除外する場合は、FPRをFortify Software Security Centerにアップロードする前に、ローカルでFPRを現在の監査プロジェクトとマージする必要があります。FPRにはFortify Static Code Analyzerのバージョンが含まれていないため、Fortify Software Security Centerはサーバ上でFPRをマージできません。

`com.fortify.sca.FVDLDisableProgramData=true`
コマンドラインオプション:
`-fvdl-no-progdata`
	FPRからプログラムデータを除外します。これにより、Fortify Audit Workbenchの関数ビューから汚染されたソースの情報が削除されます。通常、このプロパティがFPRファイルの全体的なサイズに及ぼす影響は最小限です。

大きなFPRファイルを開く

大きなFPRファイルをFortify Audit Workbenchで開くのに必要な時間を短縮するために、*<sca_install_dir>*`/Core/config/fortify.properties`ファイルにいくつかのプロパティを設定できます。これらのプロパティについて詳しくは、『*OpenText™ OpenText™ Fortify Static Code Analyzer アプリケーションおよびツールガイド*』を参照してください。次の表は、大きなFPRファイルを開く時間を短縮するために使用できるプロパティについて説明しています。

プロパティ名	説明
`com.fortify.model.DisableProgramInfo=true`	Fortify Audit Workbenchのコードナビゲーション機能の使用を無効にします。

プロパティ名	説明

com.fortify.
model.IssueCutOffStartIndex
=<*num*>(包括的)
 	
com.fortify.
model.IssueCutOffEndIndex
=<*num*>(排他的)
	問題カットオフの開始インデックスと終了インデックスを設定します。IssueCutOffStartIndexプロパティ(包括的)とIssueCutOffEndIndex(排他的)で、表示する問題のサブセットを指定できます。たとえば、最初の100個の問題を表示するには、次のように指定します。
	
	`com.fortify.model.`
	`IssueCutOffStartIndex=0`
	
	`com.fortify.model.`
	`IssueCutOffEndIndex=101`
	
	IssueCutOffStartIndexはデフォルトで0なので、このプロパティを指定する必要はありません。

com.fortify.
model.IssueCutOffByCategoryStar
tIndex=
<*num*>(包括的)
 	
com.fortify.
model.IssueCutOffByCategoryEndI
ndex=
<*num*>(排他的)
	問題カットオフの開始インデックスをカテゴリ別に設定します。これら2つのプロパティは、前のカットオフプロパティと似ていますが、カテゴリごとに指定する点が異なります。たとえば、各カテゴリの最初の5つの問題を表示するには、次のように指定します。
	
	`com.fortify.model.`
	`IssueCutOffByCategoryEndIndex=6`

com.fortify. model.MinimalLoad=true	FPRからロードされるデータを最小化します。関数ビューの使用も制限され、Fortify Audit WorkbenchがFPRからソースをロードできなくなる場合があります。
com.fortify. model.MaxEngineErrorCount= <*num*>	FPRからロードする、Fortify Static Code Analyzerによって報告される警告の数を指定します。スキャンの警告が多いプロジェクトでは、この数をデフォルトの3000から減らすと、大きなFPRファイルのロード時間を短縮できます。
com.fortify. model.ExecMemorySetting	Fortify Audit Workbenchでiidmigratorやfortifyupdateなどの外部コマンドラインツールを開始するためのJVMヒープメモリサイズを指定します。

長時間実行されているスキャンの監視

Fortify Static Code Analyzerの実行中に大規模で複雑なスキャンを実行すると、完了までに長い時間がかかることがよくあります。スキャン中に、状況を常に把握できるとは限りません。Fortifyでは、デバッグログをカスタマサポートチームに提供することを推奨していますが、Fortify Static Code Analyzerで実行されている操作とそのパフォーマンスをリアルタイムで確認するには、いくつかの方法があります。

SCAStateツールの使用

SCAStateコマンドラインツールを使用すると、分析フェーズ中に最新の状態分析情報を確認できます。SCAStateツールは*<sca_install_dir>*/binディレクトリにあります。分析のライブビューに加えて、Fortify Static Code Analyzerが分析フェーズ中にどこで時間を費やしているかを示すタイマとカウンタのセットも用意されています。SCAStateの使用の詳細については、「*"Fortify Static Code Analyzerスキャンステータスの確認" ページ155*」を参照してください。

JMXツールの使用

ツールを使用すると、JMXテクノロジでFortify Static Code Analyzerを監視できます。これらのツールでは、長期間にわたってFortify Static Code Analyzerのパフォーマンスを追跡する方法が提供されます。これらのツールの詳細については、完全なOracleドキュメント(http://docs.oracle.comで入手可能)を参照してください。

注: これらはサードパーティのツールであり、Fortifyでは提供またはサポートされていません。

JConsoleの使用

JConsoleは、JMX仕様に準拠した対話型のモニタリングツールです。JConsoleの欠点は、出力を保存できないことです。

JConsoleを使用するには、最初に追加のJVMパラメータをいくつか設定する必要があります。次の環境変数を設定します。

```
export SCA_VM_OPTS="-Dcom.sun.management.jmxremote -
Dcom.sun.management.jmxremote.port=9090 -
Dcom.sun.management.jmxremote.ssl=false -
Dcom.sun.management.jmxremote.authenticate=false"
```

JMXパラメータを設定したら、Fortify Static Code Analyzerのスキャンを開始します。スキャン時に次のコマンドを使用して、JConsoleを起動してFortify Static Code Analyzerをローカルまたはリモートで監視します。

```
jconsole <host_name>:9090
```

Java VisualVMの使用

Java VisualVMでは、JConsoleと同じ機能が提供されています。また、JVMに関する詳細情報も提供され、監視情報をアプリケーションスナップショットファイルに保存できます。これらのファイルは保存して、後でJava VisualVMで開くことができます。

JConsoleと同様に、Java VisualVMを使用する前に、「"JConsoleの使用" 前のページ」の説明と同じJVMパラメータを設定する必要があります。

JVMパラメータを設定したら、スキャンを開始します。次のコマンドを使用すると、Java VisualVMを起動してスキャンをローカルまたはリモートで監視できます。

```
jvisualvm <host_name>:9090
```

第21章: トラブルシューティング

このセクションでは、次のトピックについて説明します。

終了コード

次の表に、取り得るFortify Static Code Analyzer終了コードを示します。

終了コード	説明
0	成功
1	一般的な障害
2	入力ファイルが不正です (これは、Fortify Static Code Analyzerでサポートされていない拡張子を持つファイルの変換が試みられたことを示す場合があります)
3	プロセスがタイムアウトしました
4	分析が完了し、番号付きの警告メッセージがコンソールまたはログファイルに書き込まれました
5	分析が完了し、番号付きのエラーメッセージがコンソールまたはログファイルに書き込まれました
6	スキャンフェーズで問題の結果を生成できませんでした
7	有効なライセンスを検出できないか、または実行時にLIMライセンスが期限切れになりました

デフォルトで、Fortify Static Code Analyzerでは終了コード0、1、2、3、または7だけが返されます。

<sca_install_dir>/Core/Config/fortify-sca.propertiesファイル内の com.fortify.sca.ExitCodeLevelプロパティを設定することで、デフォルトの終了コードオプションを拡張できます。

有効な値は次のとおりです。

- nothing—デフォルトの終了コード(0、1、2、3、または7)を返します。
- warnings—デフォルトの終了コードに加えて、終了コード4と5を返します。
- errors—デフォルトの終了コードに加えて、終了コード5を返します。
- no_output_file—デフォルトの終了コードに加えて、終了コード6を返します。

メモリのチューニング

スキャンに必要な物理RAMの量は、コードの複雑さによって異なります。デフォルトでは、システムで使用可能な物理メモリに基づいて、Fortify Static Code Analyzerにより使用するメモリが自動的に割り当てされます。通常はこれで十分です。"出力オプション" ページ142で説明するように、-Xmxコマンドラインオプションを使用してJavaヒープサイズを調整できます。

このセクションでは、分析中にOutOfMemoryエラーが発生した場合の解決方法について説明します。

注: このセクションで説明するメモリ割り当てオプションを設定し、SCA_VM_OPTS環境変数を設定してすべてのスキャンに対して実行することができます。

Javaヒープの枯渇

Javaヒープの枯渇は、Fortify Static Code Analyzerスキャン時に発生する可能性のある最も一般的なメモリの問題です。これは、Fortify Static Code Analyzerでコードのスキャンに使用するJava仮想マシンに割り当てるヒープスペースが少なすぎるために発生します。次の症状からJavaヒープの枯渇を識別できます。

症状

これらのメッセージの1つ以上がFortify Static Code Analyzerログファイルとコマンドライン出力に出現します。

```
There is not enough memory available to complete analysis. For details on
making more memory available, please consult the user manual.
java.lang.OutOfMemoryError: Java heap space java.lang.OutOfMemoryError: GC
overhead limit exceeded
```

解決策

Javaヒープの枯渇の問題を解決するには、スキャンの開始時にFortify Static Code Analyzer Java仮想マシンに多くのヒープ領域を割り当てます。ヒープサイズを大きくするには、Fortify Static Code Analyzerスキャンの実行時に-Xmxコマンドラインオプションを使用します。たとえば、-Xmx1Gは1GBを使用できる

ようにします。このパラメータを使用する前に、Javaヒープ領域で許容される最大値を決定してください。最大値は、使用可能な物理メモリによって異なります。

32 GB ~ 48 GBのヒープサイズは、内部JVMの実装上推奨されていません。この範囲のヒープサイズでは、32 GBよりもパフォーマンスが低下します。32GB未満のヒープサイズはJVMによって最適化されます。スキャンで32GBを超えるサイズが必要な場合は、64GB以上が必要になります。ガイドラインとして、メモリを大量に消費するその他のプロセスが実行されていないと仮定すると、使用可能なメモリの2/3を超えて割り当てないでください。

システムがFortify Static Code Analyzer実行専用の場合は、変更する必要があります。ただし、メモリを大量に消費するその他のプロセスとシステムリソースが共有されている場合は、それらの他のプロセスの許容値を差し引いてください。

注: 常駐していても(Fortify Static Code Analyzerの実行中にアクティブではないプロセスは、オペレーティングシステムがディスクにスワップする可能性があっても考慮する必要はありません。環境内で使用可能なメモリよりも多くの物理メモリをFortify Static Code Analyzerに割り当てると、「スラッシュ」が発生する可能性があります。この場合、通常はシステム上の他の機能とともにスキャンの速度が低下します。

ネイティブヒープの枯渇

ネイティブヒープの枯渇はまれなシナリオで、Java仮想マシンが起動時にJavaメモリ領域を割り当てできるものの、ネイティブ操作(ガーベジコレクションなど)のために残されたリソースがあまりに少なくなります。最終的には、プロセスが即座に終了する致命的なメモリ割り当てエラーが発生します。

症状

ネイティブヒープの枯渇は、Fortify Static Code Analyzerプロセスの異常終了およびコマンドラインでの次の出力により識別できます。

```
# A fatal error has been detected by the Java Runtime Environment: # #
java.lang.OutOfMemoryError: requested ... bytes for GrET ...
```

これは致命的なJava仮想マシンエラーであるため、通常は、作業ディレクトリにファイル名`hs_err_pidNNN.log`で作成されたエラーログが伴います。

解決策

この問題はプロセス内で過大な負荷が発生していることが原因であるため、Javaメモリ領域(Javaヒープ)に使用されるメモリの量を減らすことが解決策です。この値を小さくすると、負荷の問題が減り、スキャンを正常に完了させることができます。

スタックオーバーフロー

Javaアプリケーションのスレッドごとに独自のスタックがあります。スタックには、戻りアドレス、関数/メソッド呼び出しの引数などが保持されます。スレッドが再帰的アルゴリズムを使用して大規模な構造体を処理する傾向がある場合、すべての戻りアドレスのために大きなスタックが必要になる場合があります。JVMでは、`-Xss`オプションを使用してそのサイズを設定できます。

症状

通常、このメッセージはFortify Static Code Analyzerログファイルに出現しますが、コマンドライン出力にも表示される場合があります。

```
java.lang.StackOverflowError
```

解決策

デフォルトのスタックサイズは16MBです。スタックサイズを大きくするには、`sourceanalyzer`コマンドに`-Xss`オプションを渡します。たとえば、`-Xss32M`を使用するとスタックが32MBに増えます。

複雑な関数のスキャン

Fortify Static Code Analyzerスキャン中に、Dataflow Analyzerで分析を完了できない関数が出現し、次のメッセージをレポートする場合があります。

```
Function <name> is too complex for <analyzer> analysis and will be skipped
(<identifier>)
```

ここで:

- *<name>*は、ソースコード関数の名前です。
- *<analyzer>* は、アナライザの名前です。
- *<identifier>* は、複雑性のタイプであり、次のいずれかになります。
  - `l`: 個別の場所が多すぎる
  - `m`: メモリ不足
  - `s`: スタックサイズが小さすぎる
  - `t`: 時間がかかりすぎる分析
  - `v`: 関数アクセス数が制限を超えた

Fortify Static Code Analyzerによる分析の深さは、利用可能なリソースに依存する場合があります。Fortify Static Code Analyzerは、複雑性のメトリックを使用して、これらのリソースと検出可能な脆弱性の数のバランスを取ります。このため、Fortify Static Code Analyzerが十分なリソースを使用できないと見なされる場合は、特定の関数の分析を中止することがあります。これは通常、「Function too complex」メッセージが表示される場合です。

このメッセージが表示されても、Fortify Static Code Analyzerでプログラム内の機能が完全に無視されたことを意味するわけではありません。たとえば、Dataflow Analyzerでは通常、分析を完了するまでに何度も関数にアクセスするため、それまでのアクセスではこの複雑性の限界に達していない可能性があります。この場合、結果には、前回の訪問で学習した情報がすべて含まれます。

リミッタと呼ばれるFortify Static Code Analyzerプロパティを使用して、「中止」ポイントを制御できます。アナライザごとに異なるリミッタが用意されています。

次のセクションでは、この問題の解決方法について説明します。

Dataflow Analyzerのリミッタ

Dataflow Analyzerには、次の3種類の複雑性識別子があります。

- `l`: 個別の場所が多すぎる
- `m`: メモリ不足
- `s`: スタックサイズが小さすぎる
- `v`: 関数アクセス数が制限を超えた

`s`で識別される問題を解決するには、`-Xss`を16MBを超える値に設定してスタックサイズを増やします。

複雑性識別子`m`を解決するには、Fortify Static Code Analyzerの物理メモリを増やします。

複雑性識別子`l`を解決するには、Fortify Static Code Analyzerプロパティファイル*<sca_install_dir>*`/Core/config/fortify-sca.properties`またはコマンドラインで次のリミッタを調整できます。

プロパティ名	デフォルト値
`com.fortify.sca.limiters.MaxTaintDefForVar`	1000
`com.fortify.sca.limiters.MaxTaintDefForVarAbort`	4000
`com.fortify.sca.limiters.MaxFieldDepth`	4

`MaxTaintDefForVar`リミッタは関数の複雑性を表すディメンションレス値ですが、`MaxTaintDefForVarAbort`は上限を示します。`MaxFieldDepth`リミッタを使用して、Dataflow Analyzerが特定のオブジェクトを分析する際の精度を測定します。Fortify Static Code Analyzerでは常に可能な限り最高の精度でオブジェクトの分析を試みます。

指定された関数が指定された精度で`MaxTaintDefForVar`制限を超える場合、Dataflow Analyzerではその関数を(`MaxFieldDepth`リミッタを減らすことによって)低い精度で分析します。精度を低下させると、分析の複雑性が軽減されます。精度をさらに下げることができない場合、Fortify Static Code Analyzerでは、分析が終了するか複雑性が`MaxTaintDefForVarAbort`リミッタを超えるまで、最低の精度で分析を続行します。言い換えれば、Fortify Static Code Analyzerでは関数から少なくとも何らかの結果を得るために、最低の精度で最善を尽くそうとします。Fortify Static Code Analyzerが`MaxTaintDefForVarAbort`リミッタに達すると、関数の分析が完全に中止され、「Function too complex」という警告が表示されます。

複雑性識別子`v`を解決するには、`com.fortify.sca.limiters.MaxFunctionVisits`プロパティを調整できます。このプロパティは、テイント伝播アナライザから関数にアクセスする最大回数を設定します。デフォルトは`50`です。

制御フローアナライザとNullポインタアナライザのリミッタ

制御フローアナライザとNullポインタアナライザには、次の2種類の複雑性識別子があります。

- m: メモリ不足
- t: 時間がかかりすぎる分析

Dataflow Analyzerが関数の複雑性を処理する方法により、無限に時間がかかるという問題はありません。ただし、Control Flow AnalyzerとNull Pointer Analyzerは、複雑な関数を分析する際に非常に長い時間がかかる場合があります。そのため、Fortify Static Code Analyzerではこのような場合に分析を中止する方法が用意されていて、複雑性識別子tで「Function too complex」というメッセージが表示されます。

これらのアナライザで関数分析に費やす最大時間を変更するには、Fortify Static Code Analyzerプロパティファイル*<sca_install_dir>*/Core/config/fortify-sca.propertiesまたはコマンドラインで次のプロパティ値を調整できます。

プロパティ名	説明	デフォルト値
com.fortify.sca.CtrlflowMaxFunctionTime	単一の関数に制御フロー分析の時間制限(ミリ秒)を設定します。	600000 (10分)
com.fortify.sca.NullPtrMaxFunctionTime	単一の関数にNullポインタ分析の時間制限(ミリ秒)を設定します。	300000 (5分)

複雑性識別子mを解決するには、Fortify Static Code Analyzerの物理メモリを増やします。

注: これらのリミッタや時間設定を増やすと、複雑な関数の分析にかかる時間が長くなります。リミッタ/時間の特定の値における正確なパフォーマンスへの影響を具体的に示すのは、当該関数に依存するため困難です。「Function too complex」という警告を表示しないようにする場合は、リミッタ/時間を極めて高い値に設定できます。ただし、スキャン時間が許容できないものになる可能性があります。

問題の非決定性

並行分析モードで実行すると、問題の非決定性が発生する可能性があります。問題が発生した場合は、カスタマサポートに問い合わせ、並行分析モードを無効にしてください。並行分析モードを無効にすると逐次分析になり、処理速度は大幅に低下しますが、複数回のスキャンで決定論的な結果が得られます。

並行分析モードを無効にするには、次の手順を実行します。

1. *<sca_install_dir>*/Core/configディレクトリにあるfortify-sca.propertiesファイルをテキストエディタで開きます。
2. com.fortify.sca.MultithreadedAnalysisプロパティの値をfalseに変更します。

```
com.fortify.sca.MultithreadedAnalysis=false
```

ログファイルの場所の確認

デフォルトでは、Fortify Static Code Analyzerによって次の場所にログファイルが作成されます。

- Windows: `C:\Users\<username>\AppData\Local\Fortify\sca<version>\log`
- Windows以外: `<userhome>/.fortify/sca<version>/log`

ここで、*<version>*は使用しているFortify Static Code Analyzerバージョンです。

次の表で、Fortify Static Code Analyzerのデフォルトのログファイルについて説明します。

ファイル名	説明
`sca.log` `scaX.log`	標準ログには、sourceanalyzerの実行時に発生した情報メッセージ、警告、およびエラーのログが記録されます。
`sca_FortifySupport.log` `scaX_FortifySupport.log`	Fortify Supportログの内容は次のとおりです。 • 標準ログファイルと同じログメッセージに、詳細を追加 • 標準ログファイルに含まれていない追加の詳細メッセージ このログファイルは、カスタマサポートまたは開発チームが問題をトラブルシュートする場合に役立ちます。

解決できない警告またはエラーが発生した場合は、Fortify Supportログファイルをカスタマサポートにお送りください。

ログファイルの設定

Fortify Static Code Analyzerでログファイルに書き込む情報を設定するには、ログプロパティを設定(「"ログプロパティ" ページ212」を参照)します。次のログファイル設定を指定できます。

- ログファイルの場所と名前

  プロパティ: `com.fortify.sca.LogFile`
- ログレベル("ログレベルについて" 次のページを参照)

  プロパティ: `com.fortify.sca.LogLevel`
- sourceanalyzerを実行するごとにログファイルを上書きするかどうか

  プロパティ: `com.fortify.sca.ClobberLogFile`

  コマンドラインオプション: `-clobber-log`

ログレベルについて

選択したログレベルでは、そのレベルと同等以上のすべてのログメッセージが出力されます。次の表は、ログレベルを最低から最高の順に示しています。たとえば、デフォルトのログレベルであるINFOには、INFO、WARN、ERROR、およびFATALレベルのログメッセージが含まれます。`<sca_install_dir>/Core/config/fortify-sca.properties`ファイル内の`com.fortify.sca.LogLevel`プロパティ、またはコマンドラインで`-D`オプションを使用して、ログレベルを設定できます。

ログレベル	説明
DEBUG	カスタマサポートまたは開発チームが問題のトラブルシューティングに使用できる情報を含む
INFO	変換またはスキャンプロセスに関する基本的な情報
WARN	変換またはスキャンが停止しなかったが、正確な結果を得るために注意が必要になる可能性がある問題に関する情報
ERROR	注意が必要になる可能性がある問題に関する情報
FATAL	変換またはスキャンが中止したエラーに関する情報

問題の報告と機能拡張の要求

フィードバックは、この製品の成功に不可欠です。機能拡張やパッチを要求したり、問題を報告したりするには、カスタマサポート (https://www.microfocus.com/support)にアクセスしてください。

カスタマサポートに連絡する場合は、次の情報を含める必要があります。

- 製品: Fortify Static Code Analyzer
- Fortify Static Code Analyzerのバージョン番号および任意の独立したFortify Static Code Analyzerモジュール: バージョン番号を特定するには、次のコマンドを実行します。

```
sourceanalyzer -version
```

- プラットフォーム: (たとえば、Red Hat Enterprise Linux *<version>*)
- オペレーティングシステム: (Linuxなど)

機能拡張を要求する場合は、機能拡張の説明を含めてください。

問題を報告する場合は、サポートが問題を再現できるよう十分な詳細を入力してください。説明が詳しくなるほど、サポートが問題を分析して解決するまでの時間を短縮できます。問題が発生したときのログファイル、またはログファイルで関連する部分も含めてください。

付録A: 分析のフィルタリング

このセクションでは、スキャンフェーズ中に分析結果(FPR)から脆弱性を除外する2つの方法について説明します。フィルタファイルを使用して、特定の脆弱性インスタンス、ルール、および脆弱性カテゴリに基づいて問題を削除できます。また、フィルタセット(Fortify Audit Workbenchで作成)を使用して、問題テンプレートにある非表示の問題を削除することもできます。

注意 Fortifyでは、上級ユーザである場合にのみフィルタファイルを使用することが推奨されています。通常、監査人はFortify Static Code Analyzerで検出されたすべての問題を確認して評価する必要があるため、標準監査ではフィルタファイルを使用しないでください。

このセクションでは、次のトピックについて説明します。

フィルタファイルによる問題の除外

`sourceanalyzer`コマンドを実行すると、特定の脆弱性インスタンス、ルール、および脆弱性カテゴリをフィルタ処理するファイルを作成できます。`-filter`分析オプションを使用してファイルを指定します。

フィルタファイルは、任意のテキストエディタで作成できるテキストファイルです。このファイルで必要ないフィルタ項目のみを指定します。

注: このセクションで説明するフィルタタイプは、フィルタファイルとスキャンポリシーファイルの両方に適用されます(「"分析へのスキャンポリシーの適用" ページ48」を参照)。

次の表に、使用可能なフィルタタイプとそれぞれの例を示します。

フィルタタイプ	メモ	例

カテゴリ	カテゴリは、すべてのサブカテゴリを包含するに過ぎません。
	
	注: Fortify Static Code Analyzerでは、分析が実行される前の初期化フェーズでカテゴリフィルタが適用されます。
		`Poor Error Handling`
		
		`J2EE Bad Practices: Leftover Debug Code`

インスタンス	特定の問題のインスタンスID	`6291C6A33303ED270C269917AA8A1005`

フィルタタイプ	メモ	例
ID	**注:** Fortify Static Code Analyzerでは、分析フェーズの後にインスタンスIDフィルタが適用されます。	
ルールID	特定の問題のレポートを生成するルールID **注:** Fortify Static Code Analyzerでは、分析が実行される前の初期化フェーズでルールIDフィルタが適用されます。	`823FE039-A7FE-4AAD-B976-9EC53FFE4A59`
優先度[1]	優先度は「Fortify優先順位」とも呼ばれます。優先度の値は、昇順で、 `low`、`medium`、`high`、`critical`です。	`priority <= low` `priority < medium`
影響[1]		`impact < 0.5`
見込み[1]		`likelihood <= 1.5`
信頼度[1]		`confidence < 1.8`
可能性[1]		`probability <= 1.2`
精度[1]		`accuracy <= 1.0`

[1]優先度フィルタとメタデータフィルタには、「未満」(<)または「以下」(<=)を使用します。

参照情報

"フィルタファイルの例" 下

フィルタファイルの例

たとえば、次の出力は`EightBall.java`サンプルのスキャンによる出力です。このサンプルプロジェクトは、`basic/eightball`ディレクトリ内の`Fortify_SCA_Samples_<version>.zip`アーカイブに含まれています。

次のコマンドは、分析結果を生成するために実行されます。

```
sourceanalyzer -b eightball EightBall.java sourceanalyzer -b eightball -
scan
```

次の結果は、検出された5つの問題を示しています。

```
[F7A138CDE5235351F6A4405BA4AD7C53 : low : Unchecked Return Value : semantic
] EightBall.java(12) : Reader.read() [6291C6A33303ED270C269917AA8A1005 :
high : Path Manipulation : dataflow ] EightBall.java(12) : ->new FileReader
(0) EightBall.java(8) : <=> (filename) EightBall.java(8) : <-
>Integer.parseInt(0->return) EightBall.java(6) : <=> (filename)
EightBall.java(4) : ->EightBall.main(0) [176CC0B182267DD538992E87EF41815F :
critical : Path Manipulation : dataflow ] EightBall.java(12) : ->new
FileReader(0) EightBall.java(6) : <=> (filename) EightBall.java(4) : -
>EightBall.main(0) [E4B3ACF92911ED6D98AAC15876739EC7 : high : Unreleased
Resource : Streams : controlflow ] EightBall.java(12) : start -> loaded :
new FileReader(...) EightBall.java(14) : loaded -> end_of_scope : end scope
: Resource leaked EightBall.java(12) : start -> loaded : new FileReader
(...) EightBall.java(12) : java.io.IOException thrown EightBall.java(12) :
loaded -> loaded : throw EightBall.java(12) : loaded -> end_of_scope : end
scope : Resource leaked : java.io.IOException thrown
[BB9F74FFA0FF75C9921D0093A0665BEB : low : J2EE Bad Practices : Leftover
Debug Code : structural ] EightBall.java(4)
```

次の処理を実行するフィルタファイルの例を次に示します。

- J2EE Bad Practiceカテゴリに関連する結果をすべて削除する
- インスタンスIDに基づいてパス操作を削除する
- 特定のルールIDから生成されたデータフローの問題を削除する

```
#This is a category to filter from scan output
J2EE Bad Practices

#This is an instance ID of a specific issue to be filtered #from scan
output
6291C6A33303ED270C269917AA8A1005

#This is a specific Rule ID that leads to the reporting of a #specific
issue in the scan output: in this case the #dataflow sink for a Path
Manipulation issue.
823FE039-A7FE-4AAD-B976-9EC53FFE4A59
```

フィルタ処理された出力をテストするには、このテキストをコピーし、`test_filter.txt`という名前でファイルに貼り付けます。

test_filter.txtファイルにフィルタ処理を適用するには、次のコマンドを実行します。

```
sourceanalyzer -b eightball -scan -filter test_filter.txt
```

フィルタ処理された分析では、次の結果が生成されます。

```
[176CC0B182267DD538992E87EF41815F : critical : Path Manipulation : dataflow
] EightBall.java(12) : ->new FileReader(0) EightBall.java(6) : <=>
(filename) EightBall.java(4) : ->EightBall.main(0)
[E4B3ACF92911ED6D98AAC15876739EC7 : high : Unreleased Resource : Streams :
controlflow ] EightBall.java(12) : start -> loaded : new FileReader(...)
EightBall.java(14) : loaded -> end_of_scope : end scope : Resource leaked
EightBall.java(12) : start -> loaded : new FileReader(...) EightBall.java
(12) : java.io.IOException thrown EightBall.java(12) : loaded -> loaded :
throw EightBall.java(12) : loaded -> end_of_scope : end scope : Resource
leaked : java.io.IOException thrown
```

フィルタセットによる問題の除外

Fortify Audit Workbenchで作成した問題テンプレートでフィルタセットを使用して、分析結果から問題をフィルタできます。分析フェーズ中に問題を非表示にするフィルタセットを適用すると、Fortify Static Code Analyzerはその非表示の問題をFPRに書き込みません。これを行うには、Fortify Audit Workbenchを使用してフィルタセットを作成し、そのフィルタセットと、そのフィルタセットが含まれている問題テンプレートを使用してFortify Static Code Analyzerスキャンを実行します。Fortify Audit Workbenchでフィルタとフィルタセットを作成する方法について詳しくは、『*OpenText™ Fortify Audit Workbenchユーザガイド*』を参照してください。

次の例では、FPRから問題を削除するために、問題テンプレート内でフィルタを作成して使用する方法の基本的な手順について説明します。

1. OWASP Top 10 2021を使用し、この標準のカテゴリに分類された問題のみを表示するとします。Fortify Audit Workbenchで、OWASP_Filterという名前の新しいフィルタセットを作成します。
2. Fortify Audit Workbenchで、OWASP_Filterフィルタセット内に表示フィルタを作成します。

   ```
   [OWASP Top 10 2021]にAが含まれていない場合は問題を非表示にする
   ```

   このフィルタは、問題を検索して、名前に「A」を含むOWASP Top 10 2021のカテゴリに問題がマップされない場合は、それを非表示にします。OWASP Top 10 2021のカテゴリはすべて「A」(A01、A02、...、A10)で始まるため、文字「A」を含まないカテゴリはOWASP Top 10 2021に存在しません。このフィルタによって、問題はFortify Audit Workbenchのビューに表示されなくなりますが、FPRには残ったままになります。
3. Fortify Audit Workbenchで、問題テンプレートをIssueTemplate.xmlという名前のファイルにエクスポートします。

4. Fortify Static Code Analyzerを使用し、次のコマンドを指定して、分析フェーズでフィルタセットを指定します。

```
sourceanalyzer -b MyProject -scan -project-template IssueTemplate.xml -
Dcom.fortify.sca.FilterSet=OWASP_Filter -f MyFilteredResults.fpr
```

フィルタセットで問題をフィルタすると、FPRのサイズは削減されますが、通常、スキャン時間は短縮されません。Fortify Static Code Analyzerは、問題を計算してFPRファイルに書き込むかどうかを判断した後で、フィルタセットを調べます。フィルタセット内のフィルタによって、Fortify Static Code Analyzerがロードするルールタイプが決まります。

付録B: 環境設定オプション

Fortify Static Code Analyzerインストーラでは、一連のプロパティファイルがシステムに保存されます。プロパティファイルには、Fortify Static Code Analyzerのランタイム分析、出力、およびパフォーマンスに環境設定可能な設定が含まれています。

このセクションでは、次のトピックについて説明します。

Fortify Static Code Analyzerのプロパティファイル

プロパティファイルは、*<sca_install_dir>*/Core/configディレクトリに配置されています。インストールされたプロパティファイルには、デフォルト値が含まれています。Fortifyでは、プロパティファイルのプロパティを変更する前に、プロジェクトリーダーに相談することが推奨されています。環境設定ファイル内のプロパティはいずれも、任意のテキストエディタで変更できます。-Dオプションを使用して、コマンドラインでプロパティを指定することもできます。

次の表に、Fortify Static Code Analyzerプロパティファイルを示します。Fortify Static Code Analyzerアプリケーションおよびツールのプロパティファイルについては、『*OpenText™ OpenText™ Fortify Static Code Analyzer アプリケーションおよびツールガイド*』を参照してください。

プロパティファイル名	説明	詳細情報
fortify-sca.properties	Fortify Static Code Analyzer環境設定プロパティを定義します。	"fortify-sca.properties" ページ187
fortify-sca-quickscan.properties	Fortify Static Code Analyzerのクイックスキャンに適用される環境設定プロパティを定義します。	"fortify-sca-quickscan.properties" ページ214
fortify-rules.properties	ルールの動作を決定する環境設定プロパティを定義します。	"fortify-rules.properties" ページ217

プロパティファイルの形式

プロパティファイルの各プロパティは、文字列のペアで構成されます。1番目の文字列はプロパティ名、2番目の文字列はプロパティ値です。

```
com.fortify.sca.fileextensions.htm=HTML
```

上記のように、プロパティでは.htmファイルで使用される変換が設定されます。プロパティ名は com.fortify.sca.fileextensions.htmであり、値はHTMLに設定されています。

注: Windowsシステムのパスをプロパティ値として指定する場合は、バックスラッシュ文字(\)をバックスラッシュでエスケープする必要があります(例: com.fortify.sca.ASPVirtualRoots.Library=C:\\WebServer\\CustomerA\\inc)。

無効なプロパティは、プロパティファイルからコメントアウトされています。これらのプロパティを有効にするには、コメント記号(#)を削除してから、プロパティファイルを保存します。次の例では、プロパティファイルでcom.fortify.sca.LogFileプロパティが無効であり、環境設定の一部ではありません。

```
# default location for the log file
#com.fortify.sca.LogFile=${com.fortify.sca.ProjectRoot}/sca/log/sca.log
```

プロパティ設定の優先順位

Fortify Static Code Analyzerでは、プロパティ設定が特定の順序で使用されます。以前に設定したプロパティは、指定した値で上書きできます。プロパティファイルを変更する際は、この順序に注意してください。

次の表には、Fortify Static Code Analyzerプロパティの優先順序を示します。

順序	プロパティの指定	説明
1	-Dオプション付きのコマンドライン	コマンドラインで指定されたプロパティの優先度は最高であり、任意のスキャンで指定できます。
2	Fortify Static Code Analyzerのクイックスキャン環境設定ファイル	**注:** クイックスキャンとスキャン精度レベルのいずれかを指定できます。したがって、これらのプロパティ設定の優先度はどちらも2番目です。 クイックスキャン環境設定ファイル(fortify-sca-quickscan.properties)で指定されたプロパティの優先度は2番目ですが、-quickオプションを付けてクイックスキャンモードが有効になっている場合に限られます。

順序	プロパティの指定	説明
	Fortify Static Code Analyzerのスキャン精度プロパティファイル	スキャン精度プロパティファイルで指定されたプロパティの優先度は2番目ですが、`-scan-precision`オプションを付けてスキャン精度が有効になっている場合に限られます。
3	Fortify Static Code Analyzer環境設定ファイル	Fortify Static Code Analyzer環境設定ファイル(`fortify-sca.properties`)で指定されたプロパティの優先度は最低です。すべてのスキャンでより永続的にプロパティ値が変更されるように、このファイルを編集します。

また、Fortify Static Code Analyzerは内部でデフォルト値が定義されている一部のプロパティに依存します。

fortify-sca.properties

次のセクションでは、`fortify-sca.properties`ファイルで使用できるプロパティについて説明します。このプロパティファイルで使用できるその他のプロパティについては、"fortify-sca-quickscan.properties" ページ214を参照してください。各プロパティの説明には、値の型、デフォルト値、同等のコマンドラインオプション(該当する場合)、および例が含まれます。

変換と分析フェーズのプロパティ

次の表に示す`fortify-sca.properties`ファイルのプロパティは、変換フェーズや分析(スキャン)フェーズに適用される一般的なプロパティです。

プロパティ名	説明
変換とスキャン	
`com.fortify.sca.BuildID`	ビルドのビルドIDを指定します。 **値の型:** 文字列 **デフォルト:** (なし) **コマンドラインオプション:** `-b`
`com.fortify.sca.CmdlineOptionsFileEncoding`	*@<filename>*で指定されたコマンドラインオプションファイルのエンコードを指定します("その他のオプション" ページ145」を参照)。たとえば、このプロパティを使用して、オプションファイルでUnicodeのファイルパスを指定できます。有効なエンコード名は`java.nio.charset.Charset`で定義されています。 **注:** このプロパティは`fortify-sca.properties`ファイルでのみ有効であり、`fortify-sca-quickscan.properites`ファイルや`-D`オプションでは機能しません。

プロパティ名	説明
	値の型: 文字列 **デフォルト:** JVMシステムのデフォルト エンコーディング **例:** `com.fortify.sca.CmdlineOptionsFileEncoding=UTF-8`
`com.fortify.sca.DISabledLanguages`	変換フェーズから除外する言語のコロン区切りリストを指定します。有効な言語の値は、`abap`、`actionscript`、`apex`、`cfml`、`cobol`、`configuration`、`cpp`、`dart`、`dotnet`、`golang`、`java`、`javascript`、`jsp`、`kotlin`、`objc`、`php`、`plsql`、`python`、`ruby`、`scala`、`sql`、`swift`、`tsql`、`typescript`、および`vb`です。 **値の型:** 文字列 **デフォルト:** (なし) **コマンドラインオプション:** `-disable-language`
`com.fortify.sca.EnabledLanguages`	変換する言語のコロン区切りリストを指定します。有効な言語の値は、`abap`、`actionscript`、`apex`、`cfml`、`cobol`、`configuration`、`cpp`、`dart`、`dotnet`、`golang`、`java`、`javascript`、`jsp`、`kotlin`、`objc`、`php`、`plsql`、`python`、`ruby`、`scala`、`sql`、`swift`、`tsql`、`typescript`、および`vb`です。 **値の型:** 文字列 **デフォルト:** `com.fortify.sca.DISabledLanguages`プロパティで明示的に除外しない限り、指定したソース内のすべての言語が変換されます。 **コマンドラインオプション:** `-enable-language`
`com.fortify.sca.ProjectRoot`	変換および分析フェーズで生成された中間ファイルを保存するディレクトリを指定します。Fortify Static Code Analyzerでは、このプロジェクトのルート ディレクトリにある中間ファイルを広く活用します。場合によっては、このディレクトリをネットワークドライブではなくローカルストレージに配置すると、分析のパフォーマンスが向上します。 **値の型:** 文字列(パス) **デフォルト(Windows):** `${win32.LocalAppdata}\Fortify` **注:** `${win32.LocalAppdata}`は、Windowsのローカルアプリケーションデータシェルフォルダをポイントする特殊な変数です。 **デフォルト(Windows以外):** `$home/.fortify` **コマンドラインオプション:** `-project-root` **例:** `com.fortify.sca.ProjectRoot=C:\Users\<username>\AppData\Local\`
変換	

`com.fortify.sca.fileextensions.java`
`com.fortify.sca.fileextensions.cs`
`com.fortify.sca.fileextensions.js`
	ビルド統合を必要としない言語の特定のファイル名拡張子を変換する方法を指定します。有効な拡張子の種類は、`ABAP`、`ACTIONSCRIPT`、`APEX`、`APEX_OBJECT`、`APEX_TRIGGER`、`ARCHIVE`、`ASPNET`、`ASP`、`ASPX`、`BITCODE`、`BSP`、`BYTECODE`、`CFML`、`COBOL`、`CSHARP`、`DART`、`DOCKERFILE`、`FLIGHT`、`GENERIC`、`GO`、`HOCON`、`HTML`、`INI`、`JAVA`、`JAVA_PROPERTIES`、`JAVASCRIPT`、`JSP`、`JSPX`、`KOTLIN`、`MSIL`、`MXML`、`OBJECT`、`PHP`、`PLSQL`、`PYTHON`、`RUBY`、`RUBY_ERB`、`SCALA`、`SWIFT`、`SWC`、`SWF`、`TLD`、`SQL`、`TSQL`、`TYPESCRIPT`、`VB`、`VB6`、`VBSCRIPT`、`VISUAL_FORCE`、`VUE`、および`XML`です。

プロパティ名	説明
com.fortify.sca. fileextensions.py com.fortify.sca. fileextensions.rb com.fortify.sca. fileextensions.aspx com.fortify.sca. fileextensions.php **注:** これは部分的なリストです。完全なリストについては、プロパティファイルを参照してください。	**値の型:** 文字列(有効な言語タイプ) **既定:** 完全なリストについては、fortify-sca.propertiesファイルを参照してください。 **例:** com.fortify.sca.fileextensions.java=JAVA com.fortify.sca.fileextensions.cs=CSHARP com.fortify.sca.fileextensions.js=TYPESCRIPT com.fortify.sca.fileextensions.py=PYTHON com.fortify.sca.fileextensions.swift=SWIFT com.fortify.sca.fileextensions.razor=ASPNET com.fortify.sca.fileextensions.php=PHP com.fortify.sca.fileextensions.tf=HCL oracle:*<path_to_script>*の値を指定して、言語タイプをプログラムで指定することもできます。指定した拡張子と一致するファイル名のコマンドラインパラメータを1つ受け入れるスクリプトを指定します。このスクリプトは、有効なFortify Static Code Analyzerファイルの種類(前のリストを参照)をstdoutに書き込み、戻り値0で終了する必要があります。ゼロ以外の戻りコードが返された場合、またはスクリプトが存在しない場合、このファイルは変換されず、ログファイルにFortify Static Code Analyzerの警告が書き込まれます。 **例:** com.fortify.sca.fileextensions.jsp= oracle:*<path_to_script>*
com.fortify.sca. compilers.javac= com.fortify.sca. util.compilers.JavacCompiler com.fortify.sca. compilers.c++= com.fortify.sca. util.compilers.GppCompiler com.fortify.sca. compilers.make= com.fortify.sca. util.compilers.TouchlessCompiler com.fortify.sca. compilers.mvn= com.fortify.sca. util.compilers.MavenAdapter **注:** これは部分的なリストです。完全なリストについては、 プロパティファイルを参照	カスタム名のコンパイラを指定します。 **値の型:** 文字列(コンパイラ) **デフォルト:** 完全なリストについては、fortify-sca.propertiesファイルの「Compilers」セクションを参照してください。 **例:** 「my-gcc」がgccコンパイラであることをFortify Static Code Analyzerに知らせるには、次のようにします。 com.fortify.sca. compilers.my-gcc= com.fortify.sca.util.compilers. GccCompiler **メモ:** • コンパイラ名の先頭または末尾をアスタリスク(*)にして、0個以上の文字と一致させることができます。 • clang/clang++の実行は、gcc/g++コマンド名ではサポートされていません。次のように指定できます。com.fortify.sca.compilers.g++= com.fortify.sca.util.compilers.GppCompiler

プロパティ名	説明
してください。	
com.fortify.sca. UseAntListener	trueに設定すると、Fortify Static Code Analyzerによってコンパイラオプションに com.fortify.dev.ant.SCAListenerが追加されます。 **値の型:** ブール値 **デフォルト:** false
com.fortify.sca. exclude	変換から除外するファイルを1つ以上指定します。複数のファイルはセミコロン(Windows)またはコロン(Windows以外)で区切ります。ファイル指定子の使い方について詳しくは、"ファイルとディレクトリの指定" ページ149を参照してください。 **注:** Fortify Static Code Analyzerでは、ビルド統合のない変換時にのみこのプロパティが使用されます。ほとんどのコンパイラまたはビルドツールと統合すると、このプロパティで指定されている場合でも、コンパイラまたはビルドツールで処理されるすべてのソースファイルがFortify Static Code Analyzerによって変換されます。ただし、Fortify Static Code AnalyzerのxcodebuildおよびMSBuild統合では-excludeオプションがサポートされています。 **値の型:** 文字列 **デフォルト:** 無効 **コマンドラインオプション:** -exclude **例:** com.fortify.sca.exclude=file1.x;file2.x
com.fortify.sca. InputFileEncoding	ソースファイルのエンコーディングタイプを指定します。Fortify Static Code Analyzerでは、異なる形式でエンコードされたソースファイルを含むプロジェクトをスキャンできます。マルチエンコーディングされたプロジェクトを使用するには、Fortify Static Code Analyzerで最初にソースコードファイルが読み込まれる際に、変換フェーズで-encodingオプションを指定する必要があります。Fortify Static Code Analyzerでは、このエンコーディングがビルドセッションで記憶され、FVDLファイルに反映されます。 通常、エンコーディングタイプを指定しないと、Fortify Static Code Analyzerではエンコーディングパラメータなしでjava.io.InputStreamReaderコンストラクタからfile.encodingが使用されます。一部のケース(ActionScriptパーサの場合など)では、Fortify Static Code AnalyzerのデフォルトはUTF-8です。 **値の型:** 文字列 **デフォルト:** (なし) **コマンドラインオプション:** -encoding **例:** com.fortify.sca.InputFileEncoding=UTF-16
com.fortify.sca. RegExecutable	Windowsプラットフォームでは、reg.exeシステムユーティリティへのパスを指定します。Cygwin内からFortify Static Code Analyzerを実行する場合でも、Cygwin構文ではなくWindows構文でパスを指定します。バックスラッシュをエスケープする場合は、バックスラッシュを追加します。 **値の型:** 文字列(パス)

プロパティ名	説明
	デフォルト: `reg` **例:** `com.fortify.sca.RegExecutable=C:\\Windows\\System32\\reg.exe`
`com.fortify.sca.xcode.TranslateAfterError`	xcodebuildサブプロセスがゼロ以外の終了コードで終了した場合にxcodebuildタッチレスアダプタが変換を続行するかどうかを指定します。falseに設定すると、xcodebuildでゼロ以外の終了コードが発生した後で変換が中止され、同じ終了コードでFortify Static Code Analyzerタッチレスビルドが停止します。trueにすると、Fortify Static Code Analyzerタッチレスビルドによってxcodebuildが終了する前に識別されたビルドファイルの変換が実行され、(他のエラーが発生しない限り)終了コード0でFortify Static Code Analyzerが終了します。 この設定に関係なく、ゼロ以外のコードでxcodebuildが終了した場合は、xcodebuildの終了コード、stdout、およびstderrがログファイルに書き込まれます。 **値の型:** ブール値 **デフォルト:** `false`
スキャン	
`com.fortify.sca.AddImpliedMethods`	trueに設定すると、Fortify Static Code Analyzerで、継承による実装が発生した場合に暗黙的なメソッドが生成されます。 **値の型:** ブール値 **デフォルト:** `true`
`com.fortify.sca.alias.Enable`	trueに設定すると、エイリアス分析が有効になります。 **値の型:** ブール値 **デフォルト:** `true`
`com.fortify.sca.analyzer.controlflow.EnableTimeOut`	Control Flow Analyzerのタイムアウトを有効にするかどうかを指定します。 **値の型:** ブール値 **デフォルト:** `true`
`com.fortify.sca.BinaryName`	スキャンするソースファイルのサブセットを指定します。ビルド時に名前付きバイナリにリンクされたソースファイルのみがスキャンに含められます。 **値の型:** 文字列(パス) **デフォルト:** (なし) **コマンドラインオプション:** `-bin`または`-binary-name`
`com.fortify.sca.DefaultAnalyzers`	実行する分析タイプのカンマまたはコロン区切りリストを指定します。このプロパティの有効な値は、`buffer`、`content`、`configuration`、`controlflow`、`dataflow`、`nullptr`、`semantic`、および`structural`です。 **値の型:** 文字列 **デフォルト:** このプロパティはコメントアウトされ、すべての分析タイプがスキャンで使用されます。 **コマンドラインオプション:** `-analyzers`

プロパティ名	説明
com.fortify.sca.DisableFunctionPointers	trueに設定すると、スキャン時に関数ポインタが無効になります。 **値の型:** ブール値 **デフォルト:** false
com.fortify.sca.EnableAnalyzer	デフォルトのアナライザに加えて、スキャンに使用するアナライザのカンマまたはコロン区切りのリストを指定します。このプロパティの有効な値は、buffer、content、configuration、controlflow、dataflow、nullptr、semantic、およびstructuralです。 **値の型:** 文字列 **デフォルト:** (なし)
com.fortify.sca.ExitCodeLevel	デフォルトの終了コードオプションを拡張します。終了コードとこのプロパティの有効な値については、"終了コード" ページ172を参照してください。
com.fortify.sca.FilterFile	スキャンのフィルタファイルのパスを指定します。詳しくは、"フィルタファイルによる問題の除外" ページ180を参照してください。 **値の型:** 文字列(パス) **デフォルト:** (なし) **コマンドラインオプション:** -filter
com.fortify.sca.FilteredInstanceIDs	フィルタファイルを使用して除外するIIDのカンマ区切りリストを指定します。 **値の型:** 文字列 **デフォルト:** (なし) **例:** com.fortify.sca.FilteredInstanceIDs=CA4E1623A2424919B98EC19FCA279FFA,4418B3DC072647158B3758E6183C14CD
com.fortify.sca.hoa.Enable	trueに設定すると、高次分析が有効になります。 **値の型:** ブール値 **デフォルト:** true
com.fortify.sca.IncludeScaModules	プロジェクトスキャンで使用する個別のモジュールとして事前にスキャンされたライブラリのビルドIDのカンマまたはコロン区切りリストを指定します。各ビルドIDが既存のスキャンされたライブラリを示している必要があります。 **値の型:** 文字列(ビルドID) **デフォルト:** (なし) **コマンドラインオプション:** -include-modules **例:** com.fortify.sca.IncludeScaModules=LibA,LibB
com.fortify.sca.LowSeverityCutoff	重大度の抑制のカットオフレベルを指定します。Fortify Static Code Analyzerでは、このプロパティに指定された値よりも重大度が低い問題を検出した場合に無視します。 **値の型:** 数値

プロパティ名	説明
	デフォルト: 1.0
com.fortify.sca.MaxPassthroughChainDepth	関数呼び出しの入力パラメータと出力パラメータ間のテイントパスの長さを指定します。 **値の型:** 整数 **デフォルト:** 4
com.fortify.sca.MultithreadedAnalysis	Fortify Static Code Analyzerを並行分析モードで実行するかどうかを指定します。 **値の型:** ブール値 **デフォルト:** true
com.fortify.sca.Phase0HigherOrder.Languages	高次分析を実行する言語のカンマ区切りリストを指定します。有効な値は、python、swift、ruby、javascript、およびtypescriptです。 **値の型:** 文字列 **デフォルト:** python,ruby,swift,javascript,typescript
com.fortify.sca.Phase0HigherOrder.Timeout.Hard	高次分析の合計時間(秒)を指定します。ハードタイムアウト制限に達すると、アナライザはすぐに終了します。 Fortifyでは、何らかの問題で分析時間が長くなりすぎる場合に備えて、このタイムアウト制限を推奨します。Fortifyでは、固定ポイントのパスリミッタまたはソフトタイムアウトのいずれかが先に発生するように、ハードタイムアウトをソフトタイムアウトより約50%長く設定することをお勧めします。 **値の型:** 数値 **デフォルト:** 2700
com.fortify.sca.PrecisionLevel	スキャンの精度を指定します。スキャンの精度を下げると、実行速度が向上します。有効な値は、1、2、3、および4です。 **値の型:** 数値 **デフォルト:** (なし) **コマンドラインオプション:** -scan-precision | -p
com.fortify.sca.ProjectTemplate	スキャンに使用する問題テンプレートファイルを指定します。これにより、ローカルコンピュータのスキャンのみが影響を受けます。FPRをFortify Software Security Centerにアップロードする場合は、アプリケーションバージョンに割り当てられた問題テンプレートが使用されます。 **値の型:** 文字列 **デフォルト:** (なし) **コマンドラインオプション:** -project-template **例:** com.fortify.sca.ProjectTemplate=test_issuetemplate.xml
com.fortify.sca.QuickScanMode	trueに設定すると、Fortify Static Code Analyzerでクイックスキャンが実行されます。Fortify Static Code Analyzerでは、fortify-sca.properties設定ファイルの代わりにfortify-sca-quickscan.propertiesの設定が使用されます。

プロパティ名	説明
	値の型: ブール値 **デフォルト:** (無効) **コマンドラインオプション:** `-quick`
`com.fortify.sca.ScanPolicy`	レポートされた脆弱性の優先順位を決めるためのスキャンポリシーを指定します(「"分析へのスキャンポリシーの適用" ページ48」を参照)。有効なスキャンポリシー値は、`classic`、`security`、および`devops`です。 **値の型:** 文字列 **デフォルト:** `security` **コマンドラインオプション:** `-sc`または`-scan-policy`
`com.fortify.sca.ScanScaModule`	trueに設定すると、Fortify Static Code Analyzerでこのプロジェクトのモジュール式スキャンが実行されます。これにより、後続のスキャンで`include-modules`オプション(または`com.fortify.sca.IncludeScaModules`プロパティ)とともにこのライブラリのビルドIDを使用できるようになります。詳細については、「"短縮ダイヤルを使用したスキャン速度の設定" ページ163」を参照してください。 `-scan`コマンドラインオプションが指定されている場合、このプロパティは無視されます。 **値の型:** ブール値 **デフォルト:** false **コマンドラインオプション:** `-scan-module`
`com.fortify.sca.SuppressLowSeverity`	trueに設定すると、スキャンで検出された重大度の低い問題がFortify Static Code Analyzerによって無視されます。 **値の型:** ブール値 **デフォルト:** `true`
`com.fortify.sca.ThreadCount`	並行分析モードのスレッド数を指定します。リソース制約のために使用するスレッド数を減らす必要がある場合にのみ、このプロパティを追加します。スキャン時間の増加やスキャンに関する問題が発生した場合は、使用するスレッド数を減らして問題を解決できます。 **値の型:** 整数 **デフォルト:** (使用可能なプロセッサコア数)
`com.fortify.sca.TypeInferenceFunctionTimeout`	1つの関数の分析で型推論に使用できる時間(秒)。0に設定した場合、または指定しなかった場合は、無制限になります。 **値の型:** 倍精度 **デフォルト:** `60`
`com.fortify.sca.TypeInferenceLanguages`	型推論を使用する言語のカンマまたはコロン区切りのリスト。この設定により、動的に型付けされた言語の分析精度が向上します。 **値の型:** 文字列 **デフォルト:** `javascript,python,ruby,typescript`

プロパティ名	説明
com.fortify.sca.TypeInferencePhase0Timeout	フェーズ0 (プロシージャ間分析)で型推論に使用できる合計時間(秒)を指定します。0に設定した場合、または指定しなかった場合は、無制限になります。 **値の型:** 倍精度 **デフォルト:** 300
com.fortify.sca.UniversalBlacklist	すべてのアナライザから隠す関数のコロン区切りリストを指定します。 **値の型:** 文字列 **デフォルト:** .*yyparse.*

正規表現分析のプロパティ

次の表に示すfortify-sca.propertiesファイルのプロパティは、正規表現分析に適用されます。

プロパティ名	説明
com.fortify.sca.regex.Enable	trueに設定すると、正規表現分析が有効になります。 **値の型:** ブール値 **デフォルト:** true
com.fortify.sca.regex.ExcludeBinaries	trueに設定すると、正規表現分析からバイナリファイルが除外されます。 **値の型:** ブール値 **デフォルト:** true
com.fortify.sca.regex.MaxSize	正規表現分析でスキャンされるファイルの最大サイズ(メガバイト単位)を指定します。この最大ファイルサイズを超えるファイルは正規表現分析から除外されます。 **値の型:** 数値 **デフォルト:** 10

参照情報

"正規表現分析" ページ49

LIMライセンスのプロパティ

次の表に示すfortify-sca.propertiesファイルのプロパティは、LIMでのライセンスに適用されます。

プロパティ名	説明
com.fortify.sca.lim.Url	LIMサーバのAPI URLを指定します。この値をテキストエディタで直接編集しないでください。この値を変更するには、コマンドラインオプションを使用します。 **値の型:** 文字列 **デフォルト:** (なし)

プロパティ名	説明
	コマンドラインオプション: `-store-license-pool-credentials` **例:** `https://<ip_address>:<port>`
`com.fortify.sca.lim.PoolName`	LIMライセンスプールの名前を指定します。この値をテキストエディタで直接編集しないでください。この値を変更するには、コマンドラインオプションを使用します。 **値の型:** 文字列 **デフォルト:** (なし) **コマンドラインオプション:** `-store-license-pool-credentials`
`com.fortify.sca.lim.PoolPassword`	LIMライセンスプールのパスワード(暗号化)を指定します。この値をテキストエディタで直接編集しないでください。この値を変更するには、コマンドラインオプションを使用します。 **値の型:** 文字列 **デフォルト:** (なし) **コマンドラインオプション:** `-store-license-pool-credentials`

`com.fortify.sca.lim.ProxyUrl`	LIMサーバへの接続に使用するプロキシサーバを指定します。
	値の型: 文字列
	デフォルト: (なし)
	例:
	`http://proxy.example.com:8080`
	`https://proxy.example.com`
	コマンドラインオプション: `-store-license-pool-credentials`

`com.fortify.sca.lim.ProxyUsername`	LIMサーバに接続するためのプロキシ認証用の暗号化されたユーザ名を指定します。この値をテキストエディタで直接編集しないでください。この値を変更するには、コマンドラインオプションを使用します。 **値の型:** 文字列 **デフォルト:** (なし) **コマンドラインオプション:** `-store-license-pool-credentials`
`com.fortify.sca.lim.ProxyPassword`	LIMサーバに接続するためのプロキシ認証用の暗号化パスワードを指定します。この値をテキストエディタで直接編集しないでください。この値を変更するには、コマンドラインオプションを使用します。 **値の型:** 文字列 **デフォルト:** (なし) **コマンドラインオプション:** `-store-license-pool-credentials`
`com.fortify.sca.lim.RequireTrustedSSLCert`	trueに設定すると、信頼された証明書がない状態でLIMサーバに接続しようとしたときに失敗します。このプロパティをfalseに設定すると、信頼された証明書がない状態でLIMサーバに接続しようとしたときに警告メッセージが表示されます。 **値の型:** ブール値

プロパティ名	説明
	デフォルト: `true`
`com.fortify.sca.lim.WaitForInitialLicense`	trueに設定し、LIMライセンスプールの資格情報が保存されている場合、Fortify Static Code AnalyzerではLIMライセンスが使用可能になるのを待機してから変換またはスキャンが開始されます。このプロパティをfalseに設定すると、LIMライセンスを取得できない場合にFortify Static Code Analyzerが中止します。 **値の型:** ブール値 **デフォルト:** `true`

参照情報

"LIMライセンスディレクティブ" ページ148

ルールのプロパティ

次の表に示す`fortify-sca.properties`ファイルのプロパティは、ルール(およびカスタムルール)とRulepackに適用されます。

プロパティ名	説明
`com.fortify.sca.DefaultRulesDir`	Fortifyによって提供される暗号化ルールファイルの検索時に使用するディレクトリを設定します。 **値の型:** 文字列(パス) **デフォルト:** `${com.fortify.Core}/config/rules`
`com.fortify.sca.RulesFile`	カスタムのルールパックまたはディレクトリを指定します。ディレクトリを指定すると、そのディレクトリ内の`.bin`および`.xml`拡張子を持つすべてのファイルが含められます。 **値の型:** 文字列(パス) **デフォルト:** (なし) **コマンドラインオプション:** `-rules`
`com.fortify.sca.CustomRulesDir`	カスタムルールの検索時に使用するディレクトリを設定します。 **値の型:** 文字列(パス) **デフォルト:** `${com.fortify.Core}/config/customrules`
`com.fortify.sca.RulesFileExtensions`	ルールファイルのファイル拡張子のリストを指定します。このリストに拡張子が含まれている`<sca_install_dir>/Core/config/rules`(または`-rules`オプションで指定されたディレクトリ)内のファイルが含められます。`.bin`拡張子は、このプロパティの値に関係なく常に含められます。このプロパティの区切り記号は、システムパスのセパレータです。 **値の型:** 文字列 **デフォルト:** `.xml`

プロパティ名	説明
com.fortify.sca. NoDefaultRules	trueに設定すると、デフォルトのルールパックのルールがロードされません。Fortify Static Code Analyzerでは、説明要素と言語ライブラリのルールパックが処理されますが、ルールは処理されません。 **値の型:** ブール値 **デフォルト:** (なし) **コマンドラインオプション:** -no-default-rules
com.fortify.sca. NoDefaultIssueRules	trueに設定すると、問題の直接の原因となるデフォルトのルールパックのルールが無効になります。Fortify Static Code Analyzerでは、関数の動作を特徴付けるルールは引き続きロードされます。これは、カスタムの問題ルールを作成するときに役立ちます。 **値の型:** ブール値 **デフォルト:** (なし) **コマンドラインオプション:** -no-default-issue-rules
com.fortify.sca. NoDefaultSourceRules	trueに設定すると、デフォルトのルールパックのソースルールが無効になります。これは、カスタムのソースルールを作成するときに役立ちます。 **注:** 特徴付けソースルールは無効になりません。 **値の型:** ブール値 **デフォルト:** (なし) **コマンドラインオプション:** -no-default-source-rules
com.fortity.sca. NoDefaultSinkRules	trueに設定すると、デフォルトのルールパックのシンクルールが無効になります。これは、カスタムのシンクルールを作成するときに役立ちます。 **注:** 特徴付けシンクルールは無効になりません。 **値の型:** ブール値 **デフォルト:** (なし) **コマンドラインオプション:** -no-default-sink-rules

JavaおよびKotlinのプロパティ

次の表に示すfortify-sca.propertiesファイルのプロパティは、JavaおよびKotlinコードの変換に適用されます。

プロパティ名	説明
com.fortify.sca. JavaClasspath	JavaまたはKotlinソースコードの分析時に使用するクラスパスを指定します。複数のパスはセミコロン(Windows)またはコロン(Windows以外)で区切ります。 **値の型:** 文字列(パス) **デフォルト:** (なし)

プロパティ名	説明
	コマンドラインオプション: `-cp`または`-classpath`
`com.fortify.sca.JdkVersion`	JavaまたはKotlin変換用のJavaソースコードのバージョンを指定します。 **値の型:** 文字列 **デフォルト:** `11` **コマンドラインオプション:** `-jdk`または`-source`
`com.fortify.sca.CustomJdkDir`	Fortify Static Code Analyzerインストール(*`<sca_install_dir>`*`/Core/bootcp/`)に含まれていないJDKバージョンを格納しているディレクトリを指定します。 **値の型:** 文字列(パス) **デフォルト:** (なし) **コマンドラインオプション:** `-custom-jdk-dir`
`com.fortify.sca.JavaSourcepath`	スキャンに含まれていないがネームレゾリューションに使用されるJavaまたはKotlinソースファイルディレクトリのセミコロン(Windows)またはコロン(Windows以外)区切りリストを指定します。ソースパスはクラスパスに似ていますが、解決にはクラスファイルではなくソースファイルが使用されます。 **値の型:** 文字列(パス) **デフォルト:** (なし) **コマンドラインオプション:** `-sourcepath`
`com.fortify.sca.Appserver`	JSPファイルを処理するアプリケーションサーバを指定します。有効な値は、`weblogic`または`websphere`です。 **値の型:** 文字列 **デフォルト:** (なし) **コマンドラインオプション:** `-appserver`
`com.fortify.sca.AppserverHome`	アプリケーションサーバのホームディレクトリを指定します。WebLogicの場合、これは`server/lib`を含むディレクトリへのパスです。WebSphereの場合、これは`JspBatchCompiler`スクリプトを含むディレクトリへのパスです。 **値の型:** 文字列(パス) **デフォルト:** (なし) **コマンドラインオプション:** `-appserver-home`
`com.fortify.sca.AppserverVersion`	WebLogicまたはWebSphereアプリケーションサーバのバージョンを指定します。 **値の型:** 文字列 **デフォルト:** (なし) **コマンドラインオプション:** `-appserver-version`
`com.fortify.sca.JavaExtdirs`	WebLogicおよびWebSphereアプリケーションサーバのクラスパスに暗黙的に含めるディレクトリを指定します。 **値の型:** 文字列

プロパティ名	説明
	デフォルト: (なし) **コマンドラインオプション:** `-extdirs`
`com.fortify.sca.JavaSourcepathSearch`	trueに設定すると、Fortify Static Code Analyzerで、ターゲットファイルリストによって参照されるJavaソースファイルのみが変換されます。それ以外の場合、Fortify Static Code Analyzerではソースパスに含まれるすべてのファイルが変換されます。 **値の型:** ブール値 **デフォルト:** `true`
`com.fortify.sca.DefaultJarsDirs`	一般的に使用されるJARファイルのディレクトリのセミコロンまたはコロン区切りリストを指定します。これらのディレクトリにあるJARファイルは、クラスパスオプション(`-cp`)の最後に追加されます。 **値の型:** 文字列 **デフォルト:** `default_jars`
`com.fortify.sca.DecompileBytecode`	trueに設定すると、Javaバイトコードが変換用に逆コンパイルされます。 **値の型:** ブール値 **デフォルト:** `false`
`com.fortify.sca.jsp.UseSecurityManager`	trueに設定すると、JSPパーサでJSPセキュリティマネージャが使用されます。 **値の型:** ブール値 **デフォルト:** `true`
`com.fortify.sca.jsp.DefaultEncoding`	JSPのエンコードを指定します。 **値の型:** 文字列(エンコーディング) **デフォルト:** `ISO-8859-1`
`com.fortify.sca.jsp.LegacyDataflow`	trueに設定すると、JSP関連のデータフローに対する追加のフィルタリングが有効になり、誤検出の量が減少します。 **値の型:** ブール値 **デフォルト:** `false` **コマンドラインオプション:** `-legacy-jsp-dataflow`
`com.fortify.sca.KotlinJvmDefault`	本体がKotlinインターフェイスに入っているメソッドの`DefaultImpls`クラスの生成を指定します。有効な値は次のとおりです。 • `disable` — 本体を持つメソッドが含まれている各インターフェイスに対して、`DefaultImpls`クラスを生成するように指定します。 • `all` — インターフェイスに`@JvmDefaultWithCompatibility`の注釈が付く場合に、`DefaultImpls`クラスを生成するように指定します。 • `all-compatibility` — インターフェイスに `@JvmDefaultWithoutCompatibility`の注釈が付かない限り、`DefaultImpls`クラスを生成するように指定します。 **値の型:** 文字列

プロパティ名	説明
	デフォルト: `disable`
`com.fortify.sca.ShowUnresolvedSymbols`	trueに設定すると、変換されたJavaソースファイルで参照されている未解決のタイプ、フィールド、および関数が、変換の最後に表示されます。 **値の型:** ブール値 **デフォルト:** `false` **コマンドラインオプション:** `-show-unresolved-symbols`

参照情報

"Javaコードの変換" ページ53

"Kotlinコードの変換" ページ62

Visual StudioおよびMSBuildプロジェクトのプロパティ

次の表に示す`fortify-sca.properties`ファイルのプロパティは、.NETプロジェクトおよびソリューションの変換に適用されます。

プロパティ名	説明

`WinForms.TransformDataBindings`
`WinForms.TransformMessageLoops`
`WinForms.TransformChangeNotificationPattern`
`WinForms.CollectionMutationMonitor.Label`
`WinForms.ExtractEventHandlers`
	さまざまな.NETオプションを設定します。
	値の型: ブール値および文字列
	デフォルトと例:
	`WinForms.TransformDataBindings=true`
	`WinForms.TransformMessageLoops=true`
	`WinForms.TransformChangeNotificationPattern=true`
	`WinForms.CollectionMutationMonitor.Label=WinFormsDataSource`
	`WinForms.ExtractEventHandlers=true`

`com.fortify.sca.ASPVirtualRoots.<virtual_path>`	使用する仮想ルートへのフルパスのセミコロン区切りリストを指定します。
	値の型: 文字列
	デフォルト: (なし)
	例:
	`com.fortify.sca.ASPVirtualRoots.Library=c:\\WebServer\\CustomerTwo\\Stuff`
	`com.fortify.sca.ASPVirtualRoots.Include=c:\\WebServer\\CustomerOne\\inc`

`com.fortify.sca.DisableASPExternalEntries`	trueに設定すると、スキャンのASP外部エントリが無効になります。 **値の型:** ブール値 **デフォルト:** `false`

参照情報

"Visual Studioプロジェクトの変換" ページ66

JavaScriptおよびTypeScriptのプロパティ

次の表に示すfortify-sca.propertiesファイルのプロパティは、JavaScriptおよびTypeScriptコードの変換に適用されます。

プロパティ名	説明
com.fortify.sca. EnableDOMModeling	trueに設定すると、Fortify Static Code Analyzerで、変換フェーズ中にHTMLファイルによって生成されたDOMツリーをモデル化するJavaScriptコードが生成され、DOM関連の問題(クロスサイトスクリプティング(XSS)の問題など)が特定されます。変換するコードに、埋め込みまたは参照先のJavaScriptコードを含むHTMLファイルが含まれている場合は、このプロパティを有効にしてください。 **注:** このプロパティを有効にすると、変換時間が増える可能性があります。 **値の型:** ブール値 **デフォルト:** false
com.fortify.sca. DOMModeling.tags	com.fortify.sca.EnableDOMModelingプロパティをtrueに設定した場合は、Fortify Static Code AnalyzerのDOMモデリングに含める追加のHTMLタグ名をカンマ区切りで指定できます。 **値の型:** 文字列 **デフォルト:** body、button、div、form、iframe、input、head、html、およびp。 **例:** com.fortify.sca.DOMModeling.tags=ul,li
com.fortify.sca. JavaScript.src.domain.whitelist	Fortify Static Code Analyzerで参照先のJavaScriptファイルをスキャン用にダウンロードできる信頼されたドメイン名を指定します。URLの区切り記号として縦棒を使用します。 **値の型:** 文字列 **デフォルト:** (なし) **例:** com.fortify.sca.JavaScript. src.domain.whitelist= http://www.xyz.com|http://www.123.org
com.fortify.sca. DisableJavascriptExtraction	trueに設定すると、JSP、JSPX、PHP、およびHTMLファイルに埋め込まれたJavaScriptコードは抽出されず、スキャンされません。 **値の型:** ブール値 **デフォルト:** false
com.fortify.sca. EnableTranslationMinifiedJS	trueに設定すると、圧縮されたJavaScriptファイルの変換が有効になります。 **値の型:** ブール値

プロパティ名	説明
	デフォルト: `false`
`com.fortify.sca.skip.libraries.ES6` `com.fortify.sca.skip.libraries.jQuery` `com.fortify.sca.skip.libraries.javascript` `com.fortify.sca.skip.libraries.typescript`	変換されないJavaScriptまたはTypeScript技術ライブラリファイルのカンマまたはコロン区切りリストを指定します。ファイル名には正規表現を使用できます。 `com.fortify.sca.skip.libraries.jQuery`プロパティ値に含まれる各ファイル名の`.min.js`または`.js`の前に、正規表現`'(-\d\.\d\.\d)?'`が自動的に挿入されます。 **値の型:** 文字列 **デフォルト:** • ES6: `es6-shim.min.js,system-polyfills.js, shims_for_IE.js` • jQuery: `jquery.js,jquery.min.js, jquery-migrate.js,jquery-migrate.min.js, jquery-ui.js,jquery-ui.min.js, jquery.mobile.js,jquery.mobile.min.js, jquery.color.js,jquery.color.min.js, jquery.color.svg-names.js, jquery.color.svg-names.min.js, jquery.color.plus-names.js, jquery.color.plus-names.min.js, jquery.tools.min.js` • javascript: `bootstrap.js, bootstrap.min.js, typescript.js, typescriptServices.js` • typescript: `typescript.d.ts, typescriptServices.d.ts`
`com.fortify.sca.follow.imports`	trueに設定すると、importステートメントで組み込まれたファイルが変換に含められます。 **値の型:** ブール値 **デフォルト:** `true`
`com.fortify.sca.exclude.unimported.node.modules`	trueに設定すると、インポートされたnode_modulesのみが変換に含められます。 **値の型:** ブール値 **デフォルト:** `true`

参照情報

"JavaScriptおよびTypeScriptコードの変換" ページ75

Pythonのプロパティ

次の表に示すfortify-sca.propertiesファイルのプロパティは、Pythonコードの変換に適用されます。

プロパティ名	説明
com.fortify.sca.PythonPath	追加のインポートディレクトリのセミコロン区切り(Windows)またはコロン区切り(Windows以外)リストを指定します。Fortify Static Code Analyzerでは、Pythonランタイムシステムでインポートファイルの検索に使用されるPYTHONPATH環境変数は考慮されません。このプロパティを使用して、追加のインポートディレクトリを指定します。 **値の型:** 文字列(パス) **デフォルト:** (なし) **コマンドラインオプション:** -python-path
com.fortify.sca.PythonVersion	スキャンするPythonソースコードのバージョンを指定します。有効な値は、2および3です。 **値の型:** 数値 **デフォルト:** 3 **コマンドラインオプション:** -python-version
com.fortify.sca.PythonNoAutoRootCalculation	trueに設定すると、モジュールとパッケージのインポートに使用するすべてのプロジェクトファイルに共通のルートディレクトリの自動計算が無効になります。詳しくは、「"インポート済みのモジュールとパッケージを含める" ページ82」を参照してください。 **値の型:** ブール値 **デフォルト:** false **コマンドラインオプション:** -python-no-auto-root-calculation
com.fortify.sca.DjangoTemplateDirs	Djangoテンプレートのパスのセミコロン区切り(Windows)またはコロン区切り(Windows以外)リストを指定します。Fortify Static Code Analyzerでは、Djangoのsettings.pyファイルのTEMPLATE_DIRS設定は使用されません。 **値の型:** 文字列(パス) **デフォルト:** (なし) **コマンドラインオプション:** -django-template-dirs
com.fortify.sca.DjangoDisableAutodiscover	Fortify Static Code AnalyzerでDjangoテンプレートを自動検出しないことを指定します。 **値の型:** ブール値 **デフォルト:** (なし) **コマンドラインオプション:** -django-disable-autodiscover

参照情報

"Pythonコードの変換" ページ80

Goのプロパティ

次の表に示すfortify-sca.propertiesファイルのプロパティは、Goコードの変換に適用されます。

プロパティ名	説明
com.fortify.sca.GOPATH	プロジェクト/ワークスペースのルートディレクトリを指定します。 **値の型:** 文字列 **デフォルト:** (GOPATHシステム環境変数)
com.fortify.sca.GOROOT	Goインストールの場所を指定します。 **値の型:** 文字列 **デフォルト:** (GOROOTシステム環境変数)
com.fortify.sca.GOPROXY	1つ以上のプロキシURLをカンマ区切りで指定します。directまたはoffを指定することもできます。 **値の型:** 文字列 **デフォルト:** (GOPROXYシステム環境変数)

参照情報

"Goコードの変換" ページ89

Rubyのプロパティ

次の表に示すfortify-sca.propertiesファイルのプロパティは、Rubyコードの変換に適用されます。

プロパティ名	説明
com.fortify.sca.RubyLibraryPaths	Rubyライブラリを含むディレクトリへのパスを1つ以上指定します。 **値の型:** 文字列(パス) **デフォルト:** (なし) **コマンドラインオプション:** -ruby-path
com.fortify.sca.RubyGemPaths	RubyGemsの場所へのパスを1つ以上指定します。プロジェクトにスキャンするgemが関連付けられている場合は、この値を設定します。 **値の型:** 文字列(パス) **デフォルト:** (なし) **コマンドラインオプション:** -rubygem-path

参照情報

"Rubyコードの変換" ページ94

COBOLのプロパティ

次の表に示すfortify-sca.propertiesファイルのプロパティは、COBOLコードの変換に適用されます。

プロパティ名	説明
com.fortify.sca.CobolCopyDirs	Fortify Static Code Analyzerでコピーブックファイルを検索する、1つ以上のディレクトリをセミコロン区切りまたはコロン区切りで指定します。 **値の型:** 文字列(パス) **デフォルト:** (なし) **コマンドラインオプション:** -copydirs
com.fortify.sca.CobolDialect	COBOLの方言を指定します。方言の有効な値は、COBOL390またはMICROFOCUSです。方言の値では、大文字と小文字を区別しません。 **値の型:** 文字列 **デフォルト:** COBOL390 **コマンドラインオプション:** -dialect
com.fortify.sca.CobolCheckerDirectives	1つ以上のCOBOLチェッカディレクティブをセミコロン区切りで指定します。 **値の型:** 文字列 **デフォルト:** (なし) **コマンドラインオプション:** -checker-directives
com.fortify.sca.CobolLegacy	trueに設定すると、レガシーCOBOL変換が有効になります。 **値の型:** ブール値 **デフォルト:** false **コマンドラインオプション:** -cobol-legacy
com.fortify.sca.CobolFixedFormat	trueに設定すると、固定形式のCOBOLが指定され、すべてのコード行のカラム8 ~ 72のソースコードのみを検索するようにFortify Static Code Analyzerに指示されます(レガシーCOBOL変換のみ)。 **値の型:** ブール値 **デフォルト:** false **コマンドラインオプション:** -fixed-format
com.fortify.sca.CobolCopyExtensions	1つ以上のコピーブックファイル拡張子をセミコロン区切りまたはコロン区切りで指定します(レガシーCOBOL変換のみ)。 **値の型:** 文字列 **デフォルト:** (なし) **コマンドラインオプション:** -copy-extensions

参照情報

"COBOLコードの変換" ページ96

PHPのプロパティ

次の表に示すfortify-sca.propertiesファイルのプロパティは、PHPコードの変換に適用されます。

プロパティ名	説明
com.fortify.sca.PHPVersion	PHPのバージョンを指定します。有効なバージョンのリストについては、*Fortifyソフトウェアシステム要件*のドキュメントを参照してください。 **値の型:** 文字列 **デフォルト:** 8.2 **コマンドラインオプション:** -php-version
com.fortify.sca.PHPSourceRoot	PHPのソースルートを指定します。 **値の型:** ブール値 **デフォルト:** (なし) **コマンドラインオプション:** -php-source-root

参照情報

"PHPコードの変換" ページ104

ABAPのプロパティ

次の表に示すプロパティは、ABAPコードの変換に適用されます。

プロパティ名	説明
com.fortify.sca.AbapDebug	trueに設定すると、Fortify Static Code Analyzerでメッセージをデバッグするためにステートメントが追加されます。 **値の型:** ブール値 **デフォルト:** (なし)
com.fortify.sca.AbapIncludes	Fortify Static Code AnalyzerでABAPのINCLUDEディレクティブが検出されたときに、名前付きディレクトリが検索されます。 **値の型:** 文字列(パス) **デフォルト:** (なし)

FlexおよびActionScriptのプロパティ

次の表に示すfortify-sca.propertiesファイルのプロパティは、FlexおよびActionScriptコードの変換に適用されます。

プロパティ名	説明
com.fortify.sca.FlexLibraries	「リンク先」のライブラリをセミコロン(Windows)またはコロン(Windows以外)で区切って指定します。このリストには、flex.swc、framework.swc、およびplayerglobal.swcを含める必要があります(これらは通常、Flex SDKルートのframeworks/libsディレクトリにあります)。このプロパティは、主にActionScriptを解決するために使用します。 **値の型:** 文字列(パス) **デフォルト:** (なし) **コマンドラインオプション:** -flex-libraries
com.fortify.sca.FlexSdkRoot	有効なFlex SDKのルートの場所を指定します。このフォルダには、flex-config.xmlファイルを含むフレームワークフォルダが含まれている必要があります。mxmlc実行可能ファイルを含むbinフォルダも含まれている必要があります。 **値の型:** 文字列(パス) **デフォルト:** (なし) **コマンドラインオプション:** -flex-sdk-root
com.fortify.sca.FlexSourceRoots	Flexプロジェクトの追加のソースディレクトリを指定します。複数のディレクトリはセミコロン(Windows)またはコロン(Windows以外)で区切ります。 **値の型:** 文字列(パス) **デフォルト:** (なし) **コマンドラインオプション:** -flex-source-root

ColdFusion (CFML)のプロパティ

次の表に示すfortify-sca.propertiesファイルのプロパティは、CFMLコードの変換に適用されます。

プロパティ名	説明
com.fortify.sca.CfmlUndefinedVariablesAreTainted	trueに設定すると、Fortify Static Code AnalyzerでCFMLページ内の未定義の変数が汚染されたものとして処理されます。これは、register-globals-style脆弱性を監視するDataflow Analyzerに対するヒントとして機能します。ただし、このプロパティを有効にすると、インクルードページ内の変数がそれ以前に発生したインクルードページ内の汚染された値に初期化されるデータフローの検出に支障をきたします。 **値の型:** ブール値 **デフォルト:** false

プロパティ名	説明
com.fortify.sca. CaseInsensitiveFiles	trueに設定すると、大文字と小文字を区別しないファイルシステムを使用して開発され、大文字と小文字が区別されるファイルシステム上でスキャンされたアプリケーションでは、CFMLファイルの大文字と小文字が区別されなくなります。 **値の型:** ブール値 **デフォルト:** (無効)
com.fortify.sca. SourceBaseDir	ColdFusionプロジェクトのベースディレクトリを指定します。 **値の型:** 文字列(パス) **デフォルト:** (なし) **コマンドラインオプション:** -source-base-dir

参照情報

"ColdFusionコードの変換" ページ116

SQLのプロパティ

次の表に示すfortify-sca.propertiesファイルのプロパティは、SQLコードの変換に適用されます。

プロパティ名	説明
com.fortify.sca. SqlLanguage	SQL言語のバリアントを指定します。有効なSQL言語タイプの値は、PLSQL (Oracle PL/SQLの場合)およびTSQL (Microsoft T-SQLの場合)です。 **値の型:** 文字列 **デフォルト:** TSQL **コマンドラインオプション:** -sql-language

参照情報

"SQLの変換" ページ117

出力のプロパティ

次の表に示すfortify-sca.propertiesファイルのプロパティは、分析出力に適用されます。

プロパティ名	説明
com.fortify.sca. ResultsFile	結果が書き込まれるファイル。 **値の型:** 文字列 **デフォルト:** (なし) **コマンドラインオプション:** -f

プロパティ名	説明
	例: com.fortify.sca.ResultsFile=MyResults.fpr
com.fortify.sca. Renderer	出力形式を制御します。有効な値は、fpr、fvdl、text、およびautoです。autoのデフォルトでは、-fオプションで指定されたファイルの拡張子に基づいて出力形式が選択されます。 **値の型:** 文字列 **デフォルト:** auto **コマンドラインオプション:** -format
com.fortify.sca. OutputAppend	trueに設定すると、Fortify Static Code Analyzerで既存の結果ファイルに結果が追加されます。 **値の型:** ブール値 **デフォルト:** false **コマンドラインオプション:** -append
com.fortify.sca. ResultsAsAvailable	trueに設定すると、Fortify Static Code Analyzerで使用可能になった結果が出力されます。これは、(出力ファイルを指定するための) -fオプションを指定せずに、stdoutに出力する場合に役立ちます。 **値の型:** ブール値 **デフォルト:** false
com.fortify.sca. BuildLabel	スキャンされたプロジェクトのラベルを指定します。Fortify Static Code Analyzerでは、このラベルは使用されませんが、結果に含められます。 **値の型:** 文字列 **デフォルト:** (なし) **コマンドラインオプション:** -build-label
com.fortify.sca. BuildProject	スキャンされたプロジェクトの名前を指定します。Fortify Static Code Analyzerでは、この名前は使用されませんが、結果に含められます。 **値の型:** 文字列 **デフォルト:** (なし) **コマンドラインオプション:** -build-project
com.fortify.sca. BuildVersion	スキャンされたプロジェクトのバージョンを指定します。Fortify Static Code Analyzerでは、このバージョン番号は使用されませんが、結果に含められます。 **値の型:** 文字列 **デフォルト:** (なし) **コマンドラインオプション:** -build-version
com.fortify.sca. MachineOutputMode	情報をインタラクティブに出力せずに、スクリプトまたはFortify Static Code Analyzerツールで使用できる形式で出力します。スキャンの進行状況を1行で表示せずに、コンソールの前の行の下に新しい行を出力して、更新された進行状況を表示します。 **値の型:** ブール値

プロパティ名	説明
	デフォルト: (無効) **コマンドラインオプション:** `-machine-output`
`com.fortify.sca.SnippetContextLines`	問題の周囲に表示するコードの行数を設定します。スニペットでは常に、エラーが発生した行の両側にある2行のコードが含められます。デフォルトでは、5行のコードが表示されます。 **値の型:** 数値 **デフォルト:** `2`
`com.fortify.sca.FVDLDisableDescriptions`	trueに設定すると、分析結果ファイル(FVDL)からFortify Security Contentの説明が除外されます。 **値の型:** ブール値 **デフォルト:** `false` **コマンドラインオプション:** `-fvdl-no-descriptions`
`com.fortify.sca.FVDLDisableEngineData`	trueに設定すると、分析結果ファイル(FVDL)からエンジンデータが除外されます。 **値の型:** ブール値 **デフォルト:** `false` **コマンドラインオプション:** `-fvdl-no-enginedata`
`com.fortify.sca.FVDLDisableLabelEvidence`	trueに設定すると、分析結果ファイル(FVDL)からラベル証拠が除外されます。 **値の型:** ブール値 **デフォルト:** `false`
`com.fortify.sca.FVDLDisableProgramData`	trueに設定すると、分析結果ファイル(FVDL)から`ProgramData`セクションが除外されます。 **値の型:** ブール値 **デフォルト:** `false` **コマンドラインオプション:** `-fvdl-no-progdata`
`com.fortify.sca.FVDLDisableSnippets`	trueに設定すると、分析結果ファイル(FVDL)からコードスニペットが除外されます。 **値の型:** ブール値 **デフォルト:** `false` **コマンドラインオプション:** `-fvdl-no-snippets`
`com.fortify.sca.FVDLStylesheet`	分析結果のスタイルシートの場所を指定します。 **値の型:** 文字列(パス) **デフォルト:** `${com.fortify.Core}/resources/sca/fvdl2html.xsl`

モバイルビルドセッション(MBS)のプロパティ

次の表に示すfortify-sca.propertiesファイルのプロパティは、MBSファイルに適用されます。

プロパティ名	説明
com.fortify.sca.MobileBuildSessions	trueに設定すると、Fortify Static Code Analyzerでソースファイルがビルドセッションディレクトリにコピーされます。 **値の型:** ブール値 **デフォルト:** false
com.fortify.sca.ExtractMobileInfo	trueに設定すると、Fortify Static Code Analyzerでモバイルビルドセッションからビルド IDとFortify Static Code Analyzerのバージョン番号が抽出されます。 **注:** Fortify Static Code Analyzerでは、このプロパティを使用してもモバイルビルドは抽出されません。 **値の型:** ブール値 **デフォルト:** false

参照情報

"モバイルビルドセッション" ページ46

プロキシプロパティ

次の表に示すfortify-sca.propertiesファイルのプロパティは、プロキシ設定に適用されます。

プロパティ名	説明
com.fortify.sca.https.proxyHost	プロキシホスト名を指定します。 **値の型:** 文字列 **デフォルト:** (なし)
com.fortify.sca.https.proxyPort	プロキシポート番号を指定します。 **値の型:** 数値 **デフォルト:** (なし)

ログプロパティ

次の表に示すfortify-sca.propertiesファイルのプロパティは、ログファイルに適用されます。

プロパティ名	説明
com.fortify.sca.LogFile	デフォルトのログファイルの名前と場所を指定します。

プロパティ名	説明
	値の型: 文字列(パス) **デフォルト:**${com.fortify.sca.ProjectRoot}/log/sca.logおよび${com.fortify.sca.ProjectRoot}/log/sca_FortifySupport.log **コマンドラインオプション:** -logfile
com.fortify.sca.LogLevel	両方のログファイルの最小ログレベルを指定します。有効な値は、DEBUG、INFO、WARN、ERROR、およびFATALです。詳しくは、"ログファイルの場所の確認" ページ178および"ログファイルの設定" ページ178を参照してください。 **値の型:** 文字列 **デフォルト:** INFO
com.fortify.sca.ClobberLogFile	trueに設定すると、Fortify Static Code Analyzerでsourceanalyzerを実行するたびにログファイルが上書きされます。 **値の型:** ブール値 **デフォルト:** false **コマンドラインオプション:** -clobber-log
com.fortify.sca.PrintPerformanceDataAfterScan	trueに設定すると、Fortify Static Code Analyzerでスキャンの完了後にパフォーマンス関連のデータがFortify Supportログファイルに書き込まれます。デバッグモードでは、この値は自動的にtrueに設定されます。 **値の型:** ブール値 **デフォルト:** false

参照情報

"ログファイルの設定" ページ178

デバッグのプロパティ

次の表に示すfortify-sca.propertiesファイルのプロパティは、デバッグ設定に適用されます。

プロパティ名	説明
com.fortify.sca.Debug	Fortify Supportログファイルにデバッグ情報を含めます。この情報は、カスタマサポートのトラブルシューティングにのみ役立ちます。 **値の型:** ブール値 **デフォルト:** false **コマンドラインオプション:** -debug
com.fortify.sca.DebugVerbose	これはcom.fortify.sca.Debugプロパティと同じですが、特に解析エラーに関する詳細が含まれています。 **値の型:** ブール値 **デフォルト:** (無効)

プロパティ名	説明
	コマンドラインオプション: `-debug-verbose`
`com.fortify.sca.Verbose`	trueに設定すると、Fortify Supportログファイルに詳細メッセージが含められます。 **値の型:** ブール値 **デフォルト:** `false` **コマンドラインオプション:** `-verbose`
`com.fortify.sca.DebugTrackMem`	trueに設定すると、追加のパフォーマンス情報がFortify Supportログに書き込まれます。 **値の型:** ブール値 **デフォルト:** (無効) **コマンドラインオプション:** `-debug-mem`
`com.fortify.sca.CollectPerformanceData`	trueに設定すると、パフォーマンスを追跡する追加のタイマが有効になります。 **値の型:** ブール値 **デフォルト:** (無効)
`com.fortify.sca.Quiet`	trueに設定すると、コマンドラインの進行状況情報が無効になります。 **値の型:** ブール値 **デフォルト:** `false` **コマンドラインオプション:** `-quiet`
`com.fortify.sca.MonitorSca`	trueに設定すると、Fortify Static Code Analyzerのメモリ使用量が監視され、JVMガーベッジコレクションが過剰に発生したときに警告が表示されます。 **値の型:** ブール値 **デフォルト:** `true`

fortify-sca-quickscan.properties

Fortify Static Code Analyzerでは、クイックスキャンと呼ばれるより詳細なスキャンが提供されています。このオプションでは、`fortify-sca-quickscan.properties`ファイルのプロパティ値を使用して、プロジェクトがクイックスキャンモードでスキャンされます。デフォルトでは、クイックスキャンによって分析の深さが減少し、Quick Viewフィルタセットが適用されます。Quick Viewフィルタセットでは、優先度の高い重大な問題のみが提供されます。

注: このファイルのプロパティは、スキャンのコマンドラインで`-quick`オプションを指定した場合にのみ使用されます。

次の表には、2つのデフォルト値セット(クイックスキャンのデフォルト値と通常スキャンのデフォルト値)を示します。デフォルト値が1つのみ表示されている場合は、通常スキャンとクイックスキャンの両方の値が同じです。

プロパティ名	説明
com.fortify.sca. CtrlflowMaxFunctionTime	単一の関数に制御フロー分析の時間制限(ミリ秒)を設定します。 **値の型:** 整数 **クイックスキャンのデフォルト:** 30000 **デフォルト:** 600000
com.fortify.sca. DisableAnalyzers	スキャン時に無効にするアナライザのカンマ区切りリストまたはコロン区切りリストを指定します。有効なアナライザ名は、buffer、content、configuration、controlflow、dataflow、nullptr、semantic、およびstructuralです。 **値の型:** 文字列 **クイックスキャンのデフォルト:** controlflow:buffer **デフォルト:** (なし)
com.fortify.sca. FilterSet	使用するフィルタセットを指定します。このプロパティを問題テンプレートとともに使用すると、スキャン後ではなくスキャン時にフィルタ処理できます。使用するフィルタセットを含む問題テンプレートを指定するには、「"変換と分析フェーズのプロパティ" ページ187」の「com.fortify.sca.ProjectTemplate」を参照してください。 このプロパティをQuick Viewに設定すると、大きな影響を受ける可能性があり、発生する可能性が高いルールと、大きな影響を受ける可能性があるが、発生する可能性は低いルールが実行されます。フィルタ処理された問題はFPRに書き込まれないため、FPRのサイズを削減できます。フィルタセットの詳細については、『*OpenText™ Fortify Audit Workbenchユーザガイド*』を参照してください。 **値の型:** 文字列 **クイックスキャンのデフォルト:** Quick View **デフォルト:** (なし)
com.fortify.sca. FPRDisableMetatable	Fortify Audit Workbenchで関数ビューの情報を含むメタテーブルの作成を無効にします。このメタテーブルでは、ソースウィンドウで変数を右クリックして宣言を表示できます。C/C++のスキャンに非常に長い時間がかかる場合は、このプロパティをtrueに設定すれば、スキャン時間が数時間短縮される可能性があります。 **値の型:** ブール **クイックスキャンのデフォルト:** true **デフォルト:** false **コマンドラインオプション:** -disable-metatable
com.fortify.sca. FPRDisableSourceBundling	FPRファイルにソースコードが含まれることを無効にします。スキャン時にFortify

プロパティ名	説明
	Static Code Analyzerでマーク付きのソースコードファイルが生成されないようにします。クイックスキャンの結果として生成されるFPRファイルをFortify Software Security Centerにアップロードする場合は、このプロパティを`false`に設定する必要があります。 **値の型:** ブール値 **クイックスキャンのデフォルト:** `true` **デフォルト:** `false` **コマンドラインオプション:** `-disable-source-bundling`
`com.fortify.sca.NullPtrMaxFunctionTime`	単一の関数にNullポインタ分析の時間制限(ミリ秒)を設定します。標準のデフォルトは5分間です。この値を短い制限に設定すると、スキャン時間全体が短縮されます。 **値の型:** 整数 **クイックスキャンのデフォルト:** `10000` **デフォルト:** `300000`
`com.fortify.sca.TrackPaths`	制御フロー分析のパストラッキングを無効にします。パスパストラッキングでは、問題に関するより詳細なレポートが提供されますが、より長いスキャン時間が必要です。これをJSPでのみ無効にするには、`NoJSP`に設定します。すべての関数を無効にするには、`None`を指定します。 **値の型:** 文字列 **クイックスキャンのデフォルト:** (なし) **デフォルト:** `NoJSP`
`com.fortify.sca.limiters.ConstraintPredicateSize`	Buffer Analyzerで複雑な計算にサイズ制限を指定します。スキャン時間を改善するためにBuffer Analyzerで指定されたサイズ値より大きい計算をスキップします。 **値の型:** 整数 **クイックスキャンのデフォルト:** `10000` **デフォルト:** `500000`
`com.fortify.sca.limiters.MaxChainDepth`	Dataflow Analyzerでデータが追跡される呼び出しの最大深さを制御します。この値を大きくしてデータフロー分析の範囲を拡大すると、スキャン時間が長くなります。 **注:** 呼び出しの深さは、プログラムのエントリポイント(`main()`など)からの呼び出しの深さではなく、テイントソースとシンク間にあるデータフローパス上の呼び出しの最大深さを示します。 **値の型:** 整数 **クイックスキャンのデフォルト:** `3` **デフォルト:** `5`
`com.fortify.sca.`	テイント伝播アナライザから関数にアクセスされる回数を設定します。

プロパティ名	説明
limiters.MaxFunctionVisits	**値の型:** 整数 **クイックスキャンのデフォルト:** 5 **デフォルト:** 50
com.fortify.sca. limiters.MaxPaths	単一のデータフロー脆弱性についてレポートするパスの最大数を制御します。この値を変更しても、検出される結果は変更されません。個々の結果で表示されるデータフローパスの数のみが表示されます。 **注:** Fortifyでは、スキャン時間が長くなる可能性があるため、このプロパティを5より大きい値に設定することは推奨されていません。 **値の型:** 整数 **クイックスキャンのデフォルト:** 1 **デフォルト:** 5
com.fortify.sca. limiters.MaxTaintDefForVar	Dataflow Analyzerに複雑さの制限を設定します。データフローでは、この複雑さメトリックを特定の精度レベルで超える関数の分析精度が徐々に減少します。 **値の型:** 整数 **クイックスキャンのデフォルト:** 250 **デフォルト:** 1000
com.fortify.sca. limiters.MaxTaintDefForVarAbort	関数の複雑さにハード制限を設定します。関数の複雑性が最小精度レベルでこの制限を超える場合は、アナライザで関数の分析がスキップされます。 **値の型:** 整数 **クイックスキャンのデフォルト:** 500 **デフォルト:** 4000

fortify-rules.properties

このトピックでは、fortify-rules.propertiesファイルで使用できるプロパティについて説明します。これらのプロパティを使用すると、個々のルールの動作を変更したり、ルールによって弱点を特定する方法を改善するための情報を提供したりできます。

プロパティ名	説明
com.fortify.sca.rules. password_regex.global	言語固有のルールプロパティが設定されている場合を除き、すべての言語のパスワード識別子に一致する正規表現。 **値の型:** 文字列 **デフォルト:** (?i)(s|_)?

プロパティ名	説明
	`(user|usr|member|admin|guest|login|default|new|current|old|client|server|proxy|sqlserver|my|mysql|mongo|mongodb|db|database|ldap|smtp|email|email(_)?smtp)?(_|\.)?pass(wd|word|phrase)`
`com.fortify.sca.rules.password_regex.abap`	ABAPコードのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (`com.fortify.sca.rules.password_regex.global`の値)
`com.fortify.sca.rules.password_regex.actionscript`	ActionScriptコードのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (`com.fortify.sca.rules.password_regex.global`の値)
`com.fortify.sca.rules.password_regex.apex`	Salesforce Alesコードのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (`com.fortify.sca.rules.password_regex.global`の値)
`com.fortify.sca.rules.password_regex.cfml`	ColdFusion (CFML)コードのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (なし)
`com.fortify.sca.rules.password_regex.cobol`	COBOLコードのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列

プロパティ名	説明
	デフォルト: (com.fortify.sca.rules.password_regex.globalの値)
com.fortify.sca.rules.password_regex.config	XMLのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。正規表現修飾子は使用しないでください。この値では、大文字と小文字が区別されません。 **値の型:** 文字列 **デフォルト:** (s|_)?(user|usr|member|admin|guest|login|default|new|current|old|client|server|proxy|sqlserver|my|mysql|mongo|mongodb|db|database|ldap|smtp|email|email(_)?smtp)?(_|\.)?pass(wd|word|phrase)
com.fortify.sca.rules.password_regex.cpp	CおよびC++コードのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (com.fortify.sca.rules.password_regex.globalの値)
com.fortify.sca.rules.password_regex.dotnet	.NETコードのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (com.fortify.sca.rules.password_regex.globalの値)
com.fortify.sca.rules.password_regex.docker	Dockerfileのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** .*pass(wd|word|phrase).*
com.fortify.sca.rules.password_regex.golang	Goコードのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。

プロパティ名	説明
	値の型: 文字列 **デフォルト:** (com.fortify.sca.rules.password_regex.globalの値)
com.fortify.sca.rules.password_regex.java	Javaコードのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (com.fortify.sca.rules.password_regex.globalの値)
com.fortify.sca.rules.password_regex.javascript	JavaScriptおよびTypeScriptコードのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (com.fortify.sca.rules.password_regex.globalの値)
com.fortify.sca.rules.password_regex.json	JSONのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (?i).*pass(wd|word|phrase).*
com.fortify.sca.rules.password_regex.jsp	JSPコードのパスワード識別子に一致するために使用される正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (com.fortify.sca.rules.password_regex.globalの値)
com.fortify.sca.rules.password_regex.objc	Objective-CおよびObjective-C++コードのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列

プロパティ名	説明
	デフォルト: `(?i)(s|_)?(user|usr|member|admin|guest|login|default|new|current|old|client|server|proxy|sqlserver|my|mysql|mongo|mongodb|db|database|ldap|smtp|email|email(_)?smtp)?(_|\.)?token|pin|pass(wd|word|phrase))`
`com.fortify.sca.rules.password_regex.php`	PHPコードのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (`com.fortify.sca.rules.password_regex.global`の値)
`com.fortify.sca.rules.password_regex.properties`	プロパティファイルのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (`com.fortify.sca.rules.password_regex.global`の値)
`com.fortify.sca.rules.password_regex.powershell`	PowerShellファイル内のパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (`com.fortify.sca.rules.password_regex.global`の値)
`com.fortify.sca.rules.password_regex.python`	Pythonコードのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (`com.fortify.sca.rules.password_regex.global`の値)
`com.fortify.sca.rules.password_regex.ruby`	Rubyコードのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティ

プロパティ名	説明
	が上書きされます。 **値の型:** 文字列 **デフォルト:** (`com.fortify.sca.rules.password_regex.global`の値)
`com.fortify.sca.rules.password_regex.sql`	SQLコードのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (`com.fortify.sca.rules.password_regex.global`の値)
`com.fortify.sca.rules.password_regex.swift`	Swiftコードのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** `(?i)(s|_)?(user|usr|member|admin|guest|login|default|new|current|old|client|server|proxy|sqlserver|my|mysql|mongo|mongodb|db|database|ldap|smtp|email|email(_)?smtp)?(_|\.)?(token|pin|pass(wd|word|phrase))`
`com.fortify.sca.rules.password_regex.vb`	VB6コードのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (`com.fortify.sca.rules.password_regex.global`の値)
`com.fortify.sca.rules.password_regex.yaml`	YAMLのパスワード識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現パスワードルールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** `(?i).*pass(wd|word|phrase).*`

プロパティ名	説明
com.fortify.sca.rules.key_regex.global	言語固有の正規表現キールールプロパティが設定されている場合を除き、すべての言語のキー識別子に一致する正規表現。 **値の型:** 文字列 **デフォルト:** (?i)((enc|dec)(ryption|rypt)?|crypto|secret|private)(_)?key
com.fortify.sca.rules.key_regex.abap	ABAPコードのキー識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現キールールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (com.fortify.sca.rules.key_regex.globalの値)
com.fortify.sca.rules.key_regex.actionscript	ActionScriptコードのキー識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現キールールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (com.fortify.sca.rules.key_regex.globalの値)
com.fortify.sca.rules.key_regex.cfml	CFMLコードのキー識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現キールールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (com.fortify.sca.rules.key_regex.globalの値)
com.fortify.sca.rules.key_regex.cpp	CおよびC++コードのキー識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現キールールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (com.fortify.sca.rules.key_regex.globalの値)
com.fortify.sca.rules.key_regex.golang	Goコードのキー識別子に一致する正規表現。このプロパティを設

プロパティ名	説明
	定すると、グローバル正規表現キールールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (`com.fortify.sca.rules.key_regex.global`の値)
`com.fortify.sca.rules.key_regex.java`	Javaコードのキー識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現キールールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (`com.fortify.sca.rules.key_regex.global`の値)
`com.fortify.sca.rules.key_regex.javascript`	JavaScriptおよびTypeScriptコードのキー識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現キールールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (`com.fortify.sca.rules.key_regex.global`の値)
`com.fortify.sca.rules.key_regex.jsp`	JSPコードのキー識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現キールールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (`com.fortify.sca.rules.key_regex.global`の値)
`com.fortify.sca.rules.key_regex.objc`	Objective-CおよびObjective-C++コードのキー識別子と照合するために使用する正規表現。このプロパティを設定すると、グローバル正規表現キールールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (`com.fortify.sca.rules.key_regex.global`の値)
`com.fortify.sca.rules.key_regex.php`	PHPコードのキー識別子に一致する正規表現。このプロパティを

プロパティ名	説明
	設定すると、グローバル正規表現キールールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (`com.fortify.sca.rules.key_regex.global`の値)
`com.fortify.sca.rules.key_regex.python`	Pythonコードのキー識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現キールールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (`com.fortify.sca.rules.key_regex.global`の値)
`com.fortify.sca.rules.key_regex.ruby`	Rubyコードのキー識別子と照合するために使用される正規表現。このプロパティを設定すると、グローバル正規表現キールールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (`com.fortify.sca.rules.key_regex.global`の値)
`com.fortify.sca.rules.key_regex.sql`	SQLコードのキー識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現キールールプロパティが上書きされます。 **デフォルト:** (`com.fortify.sca.rules.key_regex.global`の値)
`com.fortify.sca.rules.key_regex.swift`	Swiftコードのキー識別子と照合するために使用される正規表現。このプロパティを設定すると、グローバル正規表現キールールプロパティが上書きされます。 **値の型:** 文字列 **デフォルト:** (`com.fortify.sca.rules.key_regex.global`の値)
`com.fortify.sca.rules.key_regex.vb`	Visual Basic 6コードのキー識別子に一致する正規表現。このプロパティを設定すると、グローバル正規表現キールールプロパティが上書きされます。

プロパティ名	説明
	値の型: 文字列 **デフォルト:** (com.fortify.sca.rules.key_regex.globalの値)
com.fortify.sca.rules.GCPFunctionName	JSON/YAMLのCloud Build設定ファイルが存在しない場合に呼び出されるサーバレス関数の名前。 **値の型:** 文字列 **デフォルト:** (なし)
com.fortify.sca.rules.GCPHttpTrigger	trueに設定した場合、スキャンされるクラウド関数はHTTPトリガです。 **値の型:** ブール値 **デフォルト:** false
com.fortify.sca.rules.enable_wi_correlation	trueに設定し、サポートされているフレームワークを持つアプリケーションをFortify Static Code Analyzerでスキャンすると、OpenText™ Fortify WebInspectにインポートして結果を改善できる結果ファイルが生成されます。 **値の型:** ブール値 **デフォルト:** false

付録C: Fortify Javaの注釈

Fortifyでは、2つのバージョンのJava Fortify注釈ライブラリが提供されています。

- 保持ポリシーがCLASS (`FortifyAnnotations-CLASS.jar`)に設定された注釈。
  このバージョンのライブラリでは、コンパイル時にFortifyの注釈がバイトコードに反映されます。
- 保持ポリシーがSOURCE (`FortifyAnnotations-SOURCE.jar`)に設定された注釈。
  このバージョンのライブラリでは、Fortifyの注釈が使用されるコードがコンパイルされた後にバイトコードに反映されません。

Fortify製品を使用してアプリケーションのバイトコードを(たとえば、OpenText™ Fortify on Demandの評価で)分析する場合は、注釈の保持ポリシーがCLASSに設定されているバージョンを使用します。Fortify製品を使用してアプリケーションのソースコードを分析する場合は、いずれかのバージョンのライブラリを使用できます。ただし、Fortifyでは、保持ポリシーがSOURCEに設定されているライブラリを使用することが強く推奨されています。

重要 Fortifyの注釈を運用コードに残すと、コード内の潜在的なセキュリティ上の問題に関する情報が漏洩する可能性があるため、セキュリティ上のリスクになります。Fortifyでは、保持ポリシーがCLASSに設定されている注釈はFortify内部の分析にのみ使用し、アプリケーションの運用ビルドでは使用しないことが推奨されています。

このセクションでは、使用可能な注釈の概要を示します。サンプルアプリケーションは、`advanced/javaAnnotations`ディレクトリ内の`Fortify_SCA_Samples_<version>.zip`アーカイブに含まれています。ディレクトリに含まれている`README.txt`ファイルには、サンプルアプリケーション、アプリケーションから発生する可能性がある問題、およびこれらの問題をFortify Javaの注釈を使用して解決する方法が記述されています。

Fortify Javaの注釈には、次の2つの制限があります。

- 各注釈は1つの入力または1つの出力のみ(あるいはその両方)を指定できます。
- 同じターゲットには各タイプの注釈を1つのみ適用できます。

Fortifyでは、次の主要な3つのタイプの注釈が提供されています。

- "データフローの注釈" 次のページ
- "フィールドと変数の注釈" ページ230
- "その他の注釈" ページ231

独自のカスタム注釈がサポートされるルールを作成することもできます。詳細については、『カスタマサポート』を参照してください。

データフローの注釈

データフローの注釈には、データフロールールと同様に、ソース、シンク、パススルー、検証の4種類があります。すべてがメソッドに適用され、パラメータ名または文字列`this`および`return`によって入力、出力、またはその両方を指定します。また、データフローのソースおよびシンク注釈を関数の引数に適用することもできます。

ソース注釈

注釈パラメータで使用できる値は、`this`、`return`、または関数パラメータ名です。たとえば、ターゲットメソッドの出力に汚染を割り当てることができます。

```
@FortifyDatabaseSource("return") String [] loadUserProfile(String userID) { ... }
```

たとえば、ターゲットメソッドの引数に汚染を割り当てることができます。

```
void retrieveAuthCode(@FortifyPrivateSource String authCode) { ... }
```

特定のソース注釈に加えて、Fortifyには`FortifySource`という名前の汎用の*信頼できない*汚染ソースが用意されています。

ソース注釈の完全なリストを次に示します。

- `FortifySource`
- `FortifyDatabaseSource`
- `FortifyFileSystemSource`
- `FortifyNetworkSource`
- `FortifyPCISource`
- `FortifyPrivateSource`
- `FortifyWebSource`

パススルー注釈

パススルー注釈では、ターゲットメソッドの入力から出力にすべての汚染が転送されます。`FortifyNumberPassthrough`および`FortifyNotNumberPassthrough`の場合は、出力から汚染を割り当てたり、削除したりすることもできます。`in`注釈パラメータで使用できる値は、`this`または関数パラメータ名です。`out`注釈パラメータで使用できる値は、`this`、`return`、または関数パラメータ名です。

```
@FortifyPassthrough(in="a",out="return") String toLowerCase(String a) { ... }
```

データが純粋に数値であることを示すには、FortifyNumberPassthroughを使用します。数値データは、ソースに関係なく、クロスサイトスクリプティングなどの特定のタイプの問題を引き起こす可能性があります。FortifyNumberPassthroughを使用すると、このタイプの誤検出が減少します。プログラムで文字データを数値型(intやint[]など)に分解する場合は、FortifyNumberPassthroughを使用できます。プログラムで数値データを文字または文字列データに連結する場合は、FortifyNotNumberPassthroughを使用します。

パススルー注釈の完全なリストを次に示します。

- FortifyPassthrough
- FortifyNumberPassthrough
- FortifyNotNumberPassthrough

シンク注釈

シンク注釈では、適切なタイプの汚染がターゲットメソッドの入力に到達したときに問題が報告されます。注釈パラメータで使用できる値は、thisまたは関数パラメータ名です。

```
@FortifyXSSSink("a") void printToWebpage(int a) { ... }
```

関数の引数または戻りパラメータに注釈を適用することもできます。次の例では、汚染が引数aに到達したときに問題が報告されます。

```
void printToWebpage(int b, @FortifyXSSSink String a) { ... }
```

シンク注釈の完全なリストを次に示します。

- FortifySink
- FortifyCommandInjectionSink
- FortifyPCISink
- FortifyPrivacySink
- FortifySQLSink
- FortifySystemInfoSink
- FortifyXSSSink

検証注釈

検証注釈では、ターゲットメソッドの出力から汚染が削除されます。注釈パラメータで使用できる値は、this、return、または関数パラメータ名です。

```
@FortifyXSSValidate("return") String xssCleanse(String a) { ... }
```

検証シンク注釈の完全なリストを次に示します。

- FortifyValidate
- FortifyCommandInjectionValidate
- FortifyPCIValidate
- FortifyPrivacyValidate
- FortifySQLValidate
- FortifySystemInfoValidate
- FortifyXSSValidate

フィールドと変数の注釈

これらの注釈は、フィールドと変数(ほとんどの場合)に適用できます。

パスワードとプライベートの注釈

パスワードとプライベートの注釈を使用して、ターゲットのフィールドまたは変数がパスワードであるのか、プライベートデータであるのかを示します。

```
@FortifyPassword String x; @FortifyNotPassword String pass; @FortifyPrivate String y;
@FortifyNotPrivate String cc;
```

この例では、「x」という文字列がパスワードとして識別され、プライバシー違反およびハードコーディングされたパスワードがチェックされます。「pass」という文字列は、パスワードとして識別されません。注釈がない場合は、誤検知が発生する可能性があります。FortifyPrivateおよびFortifyNotPrivateの注釈も同様に機能しますが、プライバシー違反の問題は発生しません。

負以外の注釈とゼロ以外の注釈

これらの注釈を使用して、ターゲットのフィールドまたは変数で許可されていない値を指定します。

```
@FortifyNonNegative int index; @FortifyNonZero double divisor;
```

この例では、indexに負の値が割り当てられているか、divisorにゼロが割り当てられている場合に問題がレポートされます。

その他の注釈

戻り値チェックの注釈

FortifyCheckReturnValue注釈を使用して、戻り値をチェックする必要がある関数のリストにターゲットメソッドを追加します。

```
@FortifyCheckReturnValue int openFile(String filename){ ... }
```

危険の注釈

FortifyDangerous注釈を使用すると、ターゲット関数、フィールド、変数、またはクラスの使用がレポートされます。注釈パラメータに使用できる値は、CRITICAL、HIGH、MEDIUM、またはLOWです。これらの値は、Fortify Priority Orderの値に基づいて問題を分類する方法を示しています。

```
@FortifyDangerous{"CRITICAL"} public class DangerousClass { @FortifyDangerous{"HIGH"} String
dangerousField; @FortifyDangerous{"LOW"} int dangerousMethod() { ... } }
```

ドキュメントのフィードバックを送信する

このドキュメントに関するご意見は、電子メールでドキュメントチームまでお寄せください。

注: 弊社製品に関する技術的な問題が発生した場合は、ドキュメントチームに電子メールを送信しないでください。代わりに、https://www.microfocus.com/supportにご問い合わせてサポートを受けてください。

このコンピュータに電子メールクライアントが設定されている場合は、前のドキュメントチームに連絡するためのリンクをクリックすると、表題の行に以下の情報が付いた状態で電子メールウィンドウが開きます。

ユーザガイド(Fortify Static Code Analyzer 24.2.0)に関するフィードバック

電子メールにフィードバックを追加して、 [送信]をクリックします。

電子メールクライアントが使用できない場合は、前の情報をWebメールクライアントの新しいメッセージにコピーして、fortifydocteam@opentext.comにフィードバックを送信してください。

皆様のご意見をお待ちしております。