Oracle® Solaris Studio 12.3 コードアナライザユーザーズガイド

Part No: E26466
2011 年 12 月

ORACLE

Copyright ©2011 500 Oracle Parkway, Redwood City, CA 94065

このソフトウェアおよび関連ドキュメントの使用と開示は、ライセンス契約の制約条件に従うものとし、知的財産に関する法律により保護されています。ライセンス契約で明示的に許諾されている場合もしくは法律によって認められている場合を除き、形式、手段に関係なく、いかなる部分も使用、複写、複製、翻訳、放送、修正、ライセンス供与、送信、配布、発表、実行、公開または表示することはできません。このソフトウェアのリバース・エンジニアリング、逆アセンブル、逆コンパイルは互換性のために法律によって規定されている場合を除き、禁止されています。

ここに記載された情報は予告なしに変更される場合があります。また、誤りが無いことの保証はいたしかねます。誤りを見つけた場合は、オラクル社までご連絡ください。

このソフトウェアまたは関連ドキュメントを、米国政府機関もしくは米国政府機関に代わってこのソフトウェアまたは関連ドキュメントをライセンスされた者に提供する場合は、次の通知が適用されます。

U.S. GOVERNMENT END USERS:

Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

このソフトウェアもしくはハードウェアは様々な情報管理アプリケーションでの一般的な使用のために開発されたものです。このソフトウェアもしくはハードウェアは、危険が伴うアプリケーション（人的傷害を発生させる可能性があるアプリケーションを含む）への用途を目的として開発されていません。このソフトウェアもしくはハードウェアを危険が伴うアプリケーションで使用する際、安全に使用するために、適切な安全装置、バックアップ、冗長性（redundancy）、その他の対策を講じることは使用者の責任となります。このソフトウェアもしくはハードウェアを危険が伴うアプリケーションで使用したことに起因して損害が発生しても、オラクル社およびその関連会社は一切の責任を負いかねます。

OracleおよびJavaはOracle Corporationおよびその関連企業の登録商標です。その他の名称は、それぞれの所有者の商標または登録商標です。

Intel、Intel Xeonは、Intel Corporationの商標または登録商標です。すべてのSPARCの商標はライセンスをもとに使用し、SPARC International, Inc.の商標または登録商標です。AMD、Opteron、AMDロゴ、AMD Opteronロゴは、Advanced Micro Devices, Inc.の商標または登録商標です。UNIXは、The Open Groupの登録商標です。

このソフトウェアまたはハードウェア、そしてドキュメントは、第三者のコンテンツ、製品、サービスへのアクセス、あるいはそれらに関する情報を提供することがあります。オラクル社およびその関連会社は、第三者のコンテンツ、製品、サービスに関して一切の責任を負わず、いかなる保証もいたしません。オラクル社およびその関連会社は、第三者のコンテンツ、製品、サービスへのアクセスまたは使用によって損失、費用、あるいは損害が発生しても一切の責任を負いかねます。

120220@25097

目次

はじめに

『Oracle Solaris Studio 12.3 コードアナライザユーザーズガイド』では、コンパイラ、Discover、および Uncover による静的、動的メモリー、およびコードカバレージデータの収集、および、コードアナライザ GUI を実行してデータの分析と表示を行う方法など、コードアナライザツールの使用方法について説明します。

サポートされるプラットフォーム

この Oracle Solaris Studio リリースは、Oracle Solaris オペレーティングシステムを実行する SPARC ファミリーのプロセッサアーキテクチャーを使用するプラットフォームと、Oracle Solaris または特定の Linux システムを実行する x86 ファミリーのプロセッサアーキテクチャーを使用するプラットフォームをサポートします。

このドキュメントでは、次の用語を使用して x86 プラットフォームの違いを示しています。

- 「x86」は、64 ビットおよび 32 ビットの x86 互換製品を指します。
- 「x64」は、特定の 64 ビット x86 互換 CPU を指します。
- 「32 ビット x86」は、x86 ベースシステムで特定の 32 ビット情報を指します。

Linux システムに固有の情報は、サポートされている Linux x86 プラットフォームだけに関連し、Oracle Solaris システムに固有の情報は、SPARC および x86 システムでサポートされている Oracle Solaris プラットフォームだけに関連します。

サポートされているハードウェアプラットフォームおよびオペレーティングシステムリリースの完全なリストについては、Oracle Solaris Studio 12.3 リリースノートを参照してください。

Oracle Solaris Studio ドキュメント

Oracle Solaris Studio ソフトウェアの完全なドキュメントは、次のように見つけることができます。

- 製品のドキュメントは、リリースノート、リファレンスマニュアル、ユーザーガイド、チュートリアルも含め、Oracle Solaris Studio Documentation Web サイトにあります。

- コードアナライザ、パフォーマンスアナライザ、スレッドアナライザ、dbxtool、DLight、および IDE のオンラインヘルプには、これらのツール内の「ヘルプ」メニューだけでなく、F1 キー、および多くのウィンドウやダイアログボックスにある「ヘルプ」ボタンを使用してアクセスできます。
- コマンド行ツールのマニュアルページでは、ツールのコマンドオプションが説明されています。

関連するサードパーティの Web サイトリファレンス

このドキュメントには、詳細な関連情報を提供するサードパーティの URL が記載されています。

注 – このドキュメントで紹介するサードパーティ Web サイトが使用可能かどうかについては、Oracle は責任を負いません。このようなサイトやリソース上、またはこれらを経由して利用できるコンテンツ、広告、製品、またはその他の資料についても、Oracle は保証しておらず、法的責任を負いません。また、このようなサイトやリソースから直接あるいは経由することで利用できるコンテンツ、商品、サービスの使用または依存が直接のあるいは関連する要因となり実際に発生した、あるいは発生するとされる損害や損失についても、Oracle は一切の法的責任を負いません。

開発者向けのリソース

Oracle Solaris Studio を使用する開発者のための次のリソースを見つけるには、Oracle Technical Network Web サイトにアクセスしてください。

- リソースは頻繁に更新されます。
- ソフトウェアの最近のリリースに関連する完全なドキュメントへのリンク
- サポートレベルに関する情報
- ユーザーディスカッションフォーラム。

Oracle サポートへのアクセス

Oracle のお客様は、My Oracle Support にアクセスして電子サポートを受けることができます。詳細は、`http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` にアクセス、または、聴覚に障害がある方は、`http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` にアクセスしてください。

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 P–1 表記上の規則

字体または記号	意味	例
AaBbCc123	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	.login ファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 system%
AaBbCc123	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して示します。	system% **su** password:
AaBbCc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、rm *filename* と入力します。
『』	参照する書名を示します。	『コードマネージャ・ユーザーズガイド』を参照してください。
「」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	sun% **grep '^#define ** **XV_VERSION_STRING'**

Oracle Solaris OS に含まれるシェルで使用する、UNIX のデフォルトのシステムプロンプトとスーパーユーザープロンプトを次に示します。コマンド例に示されるデフォルトのシステムプロンプトは、Oracle Solaris のリリースによって異なります。

- C シェル

  machine_name% **command y|n** [*filename*]

- C シェルのスーパーユーザー

  machine_name# **command y|n** [*filename*]

- Bash シェル、Korn シェル、および Bourne シェル

  $ **command y|n** [*filename*]

- Bash シェル、Korn シェル、および Bourne シェルのスーパーユーザー

```
# command y|n [filename]
```

[ ] は省略可能な項目を示します。上記の例は、*filename* は省略してもよいことを示しています。

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: `Shift` キーを押します)。ただし、キーボードによっては `Enter` キーが `Return` キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば、`Ctrl-D` は `Control` キーを押したまま `D` キーを押すことを意味します。

◆ ◆ ◆ 第 1 章

概要

Oracle Solaris Studio コードアナライザは、Oracle Solaris 向けの C および C++ アプリケーションの開発者を支援するための統合ツールセットで、セキュリティーと品質の高い堅牢なソフトウェアを作成できるように設計されています。

この章には、次の情報が含まれます。

コードアナライザで分析されるデータ

コードアナライザは 3 種類のデータを分析します。

- コンパイル時に検出される静的コードエラー
- メモリーエラー検出ツールである Discover で検出される動的メモリーアクセスエラーおよび警告
- コードカバレージツールである Uncover で測定されるコードカバレージデータ

個々の分析へのアクセスを提供するほかに、コードアナライザは静的コード検査と動的メモリーアクセス検査を統合して、コードで見つかるエラーの信頼度を高めます。静的コード検査を動的メモリーアクセス分析およびコードカバレージ分析とともに使用することで、単独で機能するほかのエラー検出ツールでは見つけることのできない多くの重要なエラーをアプリケーション内に見つけることができます。

また、コードアナライザはコード内の中核となる問題、つまり、それらを修正すればほかの問題も解消される可能性の高い問題を特定します。通常、中核となる問題にはほかのいくつかの問題が関連しています。たとえば、それらの問題では割り当てポイントが共通であったり、問題が同じ関数の同じデータアドレスで発生したりするためです。

静的コード検査

静的コード検査は、コード内の一般的なプログラミングエラーをコンパイル時に検出します。CおよびC++ コンパイラの `-xanalyze=code` オプションは、コンパイラの実績ある幅広い制御およびデータフロー分析フレームワークを活用して、アプリケーションのプログラミングおよびセキュリティー上の潜在的な欠陥を分析します。

静的エラーデータの収集については、13 ページの「静的エラーデータの収集」を参照してください。

コードアナライザで分析される静的コードエラーのリストについては、19 ページの「静的コードの問題」を参照してください。

動的メモリーアクセス検査

多くの場合、コード内のメモリー関連のエラーは見つけることが困難です。プログラムを実行する前に Discover で計測機構を組み込むと、プログラムの実行中に Discover はメモリーアクセスエラーを動的に検出して報告します。たとえば、プログラムが配列を割り当て、それを初期化せずに、配列内のある場所から読み取ろうとする場合、プログラムは動作が不安定になることがあります。プログラムに Discover で計測機構を組み込んでから実行すると、Discover はこのエラーを検出します。

動的メモリーアクセスエラーデータの収集については、14 ページの「動的メモリーアクセスデータの収集」を参照してください。

コードアナライザで分析される動的メモリーアクセスの問題のリストについては、20 ページの「動的メモリーアクセスの問題」を参照してください。

コードカバレージ検査

コードカバレージは、ソフトウェアのテストの重要な部分です。Uncover はテストで実行される、または実行されないコードの領域に関する情報を提供し、テストスイートを向上させ、より多くのコードをテストできるようにします。コードアナライザは、Uncover によって収集されたデータを使用して、プログラム内のどの関数がカバーされていないか、また、該当する関数をカバーするテストを追加した場合にアプリケーションの合計カバレージが何パーセント増加するかを調べます。

コードカバレージデータの収集については、15 ページの「コードカバレージデータの収集」を参照してください。

コードアナライザを使用するための要件

コードアナライザは、Oracle Solaris Studio 12.3 C または C++ コンパイラでコンパイルされたバイナリから収集される、静的エラーデータ、動的メモリーアクセスエラーデータ、およびコードカバレージデータで機能します。

コードアナライザは、Solaris 10 10/08 オペレーティングシステムまたはそれ以降の Solaris 10 update、または Oracle Solaris 11 を実行している、SPARC ベースシステムまたは x86 ベースシステムで機能します。

コードアナライザ GUI

コンパイラ、Discover、または Uncover でデータを収集したあと、コードアナライザ GUI を起動して問題の表示と分析を行うことができます。

コードアナライザは各問題について、問題の説明、問題が見つかったソースファイルのパス名、およびそのファイルの該当するソース行を強調表示したコードスニペットを表示します。

コードアナライザでは、次のことができます:

- 問題の詳細を表示する。静的な問題の場合、詳細にはエラーパスが含まれます。動的メモリーアクセスの問題の場合、詳細には呼び出しスタックが含まれ、データが使用可能であれば割り当てスタックと解放スタックも含まれます。
- 問題が見つかったソースファイルを開く。
- エラーパスまたはスタック内の関数呼び出しから関連するソースコード行にジャンプする。
- プログラムでの関数の使用箇所をすべて見つける。
- 関数の宣言にジャンプする。
- オーバーライドされるまたはオーバーライドする関数の宣言にジャンプする。
- 関数のコールグラフを表示する。
- 問題の種類ごとに、コード例や考えられる原因などの詳細を表示する。
- 表示する問題を分析の種類、問題の種類、およびソースファイルによってフィルタリングする。
- すでに確認した問題を非表示にし、関心のない問題を閉じる。

GUI の使用に関する詳細は、GUI のオンラインヘルプと Oracle Solaris Studio 12.3 コードアナライザチュートリアルを参照してください。

クイックスタート

次の例では、プログラムをコンパイルして静的コードデータを収集し、デバッグ情報付きで再度コンパイルし、Discover で計測機構を組み込んでから実行して動的メモリーアクセスデータを収集し、Uncover で計測機構を組み込んでコードカバレージデータを収集し、コードアナライザを起動して収集済みデータを表示します。

```
% cc -xanalyze=code *.c
% cc -g *.c
% cp a.out a.out.save
% discover -a a.out
% a.out
% cp a.out.save a.out
% uncover a.out
% a.out
% uncover -a a.out.uc
% code-analyzer a.out
```

◆ ◆ ◆ 第 2 章

2 データの収集とコードアナライザの起動

コードアナライザでの分析のために収集されるデータは、ソースコードファイルが入っているディレクトリの *binary_name*.analyze ディレクトリに保存されます。*binary_name*.analyze ディレクトリは、プログラムのデータを収集するためにコンパイラ、Discover、または Uncover のいずれかを最初に実行すると作成されます。

この章には、次の情報が含まれます。

静的エラーデータの収集

C または C++ プログラムの静的エラーデータを収集するには、Oracle Solaris Studio 12.3 C または C++ コンパイラを使用して -xanalyze=code オプションでプログラムをコンパイルします。(-xanalyze=code オプションは、以前のリリースの Oracle Solaris Studio のコンパイラでは使用できません。)このオプションを使用すると、コンパイラは静的エラーを自動的に抽出し、データを *binary_name*.analyze ディレクトリの static サブディレクトリに書き込みます。

プログラムを -xanalyze=code オプションでコンパイルしたあと、別の手順でリンクする場合は、リンク手順でも -xanalyze=code オプションを指定する必要があります。

コンパイラはコード内の静的エラーをすべて検出できるわけではありません。

- 実行時にのみ使用可能になるデータに依存するエラーもあります。たとえば、次のコードの場合、ファイルから読み取られる ix の値が [0,9] の範囲外にあることを検出できないため、コンパイラは ABW (配列範囲外への書き込み) エラーを検出しません。

```
void f(int fd, int array[10])
{
  int ix;
  read(fd, &ix, sizeof(ix));
  array[ix] = 0;
}
```

- コードの実際のエラーである可能性もそうでない可能性もある、あいまいなエラーもあります。コンパイラはこのようなエラーを報告しません。
- 一部の複雑なエラーは、このリリースのコンパイラでは検出されません。

静的エラーデータを収集したあと、コードアナライザ GUI を起動してデータの分析と表示を行うか (16 ページの「コードアナライザ GUI の起動」を参照)、動的メモリーアクセスまたはコードカバレージのデータを収集するためにプログラムを再度コンパイルすることができます。

動的メモリーアクセスデータの収集

C または C++ プログラムの動的メモリーアクセスデータの収集は、Discover でバイナリに計測機構を組み込む手順と、計測機構付きバイナリを実行する手順の 2 つから成ります。

コードアナライザ用のデータを収集するために Discover でプログラムに計測機構を組み込むには、プログラムを Oracle Solaris Studio 12.3 C または C++ コンパイラでコンパイルしておく必要があります。-g オプションでコンパイルするとデバッグ情報が生成され、動的メモリーアクセスエラーおよび警告に関するソースコードおよび行番号情報をコードアナライザで表示できるようになります。

プログラムが最適化なしでコンパイルされている場合に、Discover はソースコードレベルでもっとも完全なメモリーエラー検出を提供します。最適化を使用してコンパイルした場合、一部のメモリーエラーは検出されません。

Discover で計測機構を組み込むことのできる、またはできないバイナリの種類については、『Oracle Solaris Studio 12.3 Discover および Uncover ユーザーズガイド』の「標準メモリー割り当て関数を再定義するバイナリを使用できる」および『Oracle Solaris Studio 12.3 Discover および Uncover ユーザーズガイド』の「プリロードまたは監査を使用するバイナリは使用できまい」を参照してください。

注 – Discover と Uncover の両方で使用するためのプログラムを 1 回で構築することができます。ただし、すでに計測機構の付いたバイナリに計測機構を組み込むことはできないため、Uncover を使用してカバレージデータの収集も行う予定であれば、Discover でバイナリに計測機構を組む前に Uncover 用のコピーを保存してください。例:

```
cp a.out a.out.save
```

バイナリから動的メモリーアクセスデータを収集するには:

1. Discover で `-a` オプションを使用してバイナリに計測機構を組み込みます。

   `discover -a` *binary_name*

   注 – Oracle Solaris Studio 12.3 のバージョンの Discover を使用する必要があります。以前のバージョンの Discover では `-a` オプションは使用できません。

2. 計測機構付きバイナリを実行します。動的メモリーアクセスデータが、*binary_name*`.analyze` ディレクトリの `dynamic` サブディレクトリに書き込まれます。

注 – Discover でバイナリに計測機構を組み込むときに指定できるその他の計測オプションについては、『Oracle Solaris Studio 12.3 Discover および Uncover ユーザーズガイド』の「計測オプション」または `discover` のマニュアルページを参照してください。`-c`、`-F`、`-N`、または `-T` オプションを `-a` オプションとともに使用できます。

動的メモリーアクセスデータを収集したあと、その前に収集した静的コードデータがあればそれも含め (16 ページの「コードアナライザ GUI の起動」を参照)、コードアナライザ GUI を起動してデータの分析と表示を行うことができます。あるいは、計測機構の付いていないバイナリのコピーを使用して、コードカバレージデータを収集することができます。

コードカバレージデータの収集

C または C++ プログラムのコードカバレージデータの収集は、Uncover でバイナリに計測機構を組み込む手順、計測機構付きバイナリを実行する手順、および Uncover を再度実行してコードアナライザ用のカバレージレポートを生成する手順の 3 つから成ります。

バイナリに計測機構を組み込んだあと、その計測機構付きバイナリを複数回実行してすべての実行に関するデータを蓄積してから、カバレージレポートを生成することができます。

コードアナライザ用のデータを収集するために Uncover でプログラムに計測機構を組み込むには、プログラムを Oracle Solaris Studio 12.3 C または C++ コンパイラでコンパイルしておく必要があります。-g オプションでコンパイルするとデバッグ情報が生成され、ソースコードレベルのカバレージ情報をコードアナライザで使用できるようになります。

注 – Discover で計測機構を組み込むためにプログラムをコンパイルしたときに、バイナリのコピーを保存した場合は、そのコピーの名前を元のバイナリ名に変更し、それを使用して Uncover で計測機構を組み込むことができます。例:

```
cp a.out.save a.out
```

バイナリからコードカバレージデータを収集するには:

1. Uncover でバイナリに計測機構を組み込みます。

   ```
   uncover binary_name
   ```

2. 計測機構付きバイナリを 1 回以上実行します。コードカバレージデータが、*binary_name*.uc ディレクトリに書き込まれます。
3. Uncover で -a オプションを使用して、蓄積されたデータからコードカバレージレポートを生成します。

   ```
   uncover -a  binary_name.uc
   ```

   カバレージレポートが、*binary_name*.analyze ディレクトリの coverage サブディレクトリに書き込まれます。

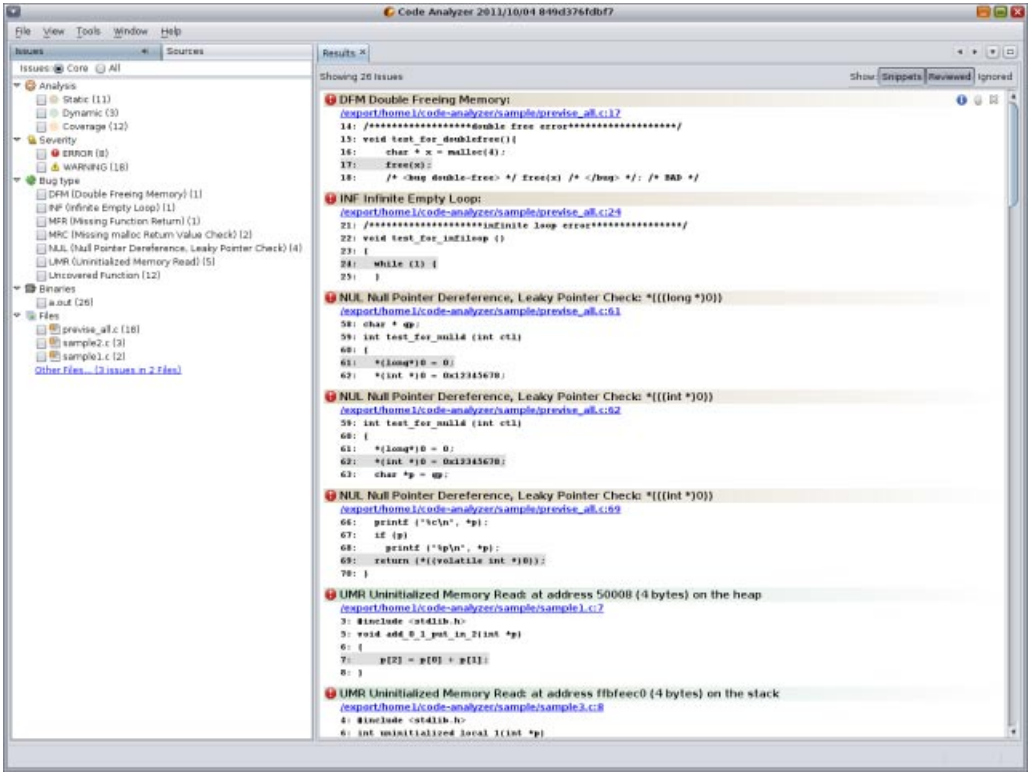
注 – Oracle Solaris Studio 12.3 のバージョンの Uncover を使用する必要があります。以前のバージョンの Uncover では -a オプションは使用できません。

コードアナライザ **GUI** の起動

コードアナライザ GUI を使用して、1 種類、2 種類、または 3 種類すべてのデータを分析できます。GUI を起動するには、code-analyzer コマンドと、収集したエラーデータを分析するバイナリへのパスを入力します。

```
code-analyzer binary_name
```

コードアナライザ GUI は、*binary_name*.analyze ディレクトリのデータを開き、表示します。



コードアナライザ GUI の実行中に、「開く」>「ファイル」を選択して別のバイナリを指定すると、そのバイナリについて収集されたデータの表示に切り替えることができます。

GUI のオンラインヘルプには、表示する結果のフィルタリング、問題の表示/非表示、および特定の問題に関する詳細の表示を行う、すべての機能の使用方法が説明されています。Oracle Solaris Studio 12.3 コードアナライザチュートリアルでは、サンプルプログラムを使用して、データの収集と分析のシナリオ全体を説明します。

◆ ◆ ◆ 付 録 A

コードアナライザで分析されるエラー

コンパイラ、Discover、および Uncover は、コード内の静的コードの問題、動的メモリーアクセスの問題、およびカバレージの問題を検出します。これらのツールで検出され、コードアナライザで分析されるエラーの種類について、以降の節で説明します。

静的コードの問題

静的コード検査では、次の種類のエラーが検出されます。

- ABR: 配列範囲外からの読み取り (beyond Array Bounds Read)
- ABW: 配列範囲外への書き込み (beyond Array Bounds Write)
- DFM: メモリーの二重解放 (Double Freeing Memory)
- ECV: 明示的型キャスト違反 (Explicit type Cast Violation)
- FMR: 解放されたメモリーからの読み取り (Freed Memory Read)
- FMW: 解放されたメモリーへの書き込み (Freed Memory Write)
- INF: 空の無限ループ (INFinite empty loop)
- メモリーリーク
- MFR: 関数の復帰なし (Missing Function Return)
- MRC: malloc 戻り値の検査なし (Missing malloc Return value Check)
- NFR: 初期化されていない関数の復帰 (uNinitialized Function Return)
- NUL: NULL ポインタ間接参照、リークの可能性があるポインタの検査 (NULl pointer dereference, leaky pointer check)
- RFM: 解放済みメモリーを返す (Return Freed Memory)
- UMR: 初期化されていないメモリーの読み取り、初期化されていないメモリーの読み取りビット操作 (Uninitialized Memory Read, Uninitialized Memoey Read bit operation)

- URV: 使用されていない戻り値 (Unused Return Value)
- VES: スコープ外での局所変数の使用 (out-of-scope local Variable usage)

動的メモリーアクセスの問題

動的メモリーアクセス検査では、次の種類のエラーが検出されます。

- ABR: 配列範囲外からの読み取り (beyond Array Bounds Read)
- ABW: 配列範囲外への書き込み (beyond Array Bounds Write)
- BFM: 不正な空きメモリー (Bad Free Memory)
- BRP: 不正な realloc アドレスパラメータ (Bad Realloc address Parameter)
- CGB: 破損したガードブロック (Corrupted Guard Block)
- DFM: メモリーの二重解放 (Double Freeing Memory)
- FMR: 解放されたメモリーからの読み取り (Freed Memory Read)
- FMW: 解放されたメモリーへの書き込み (Freed Memory Write)
- IMR: 無効なメモリーからの読み取り (Invalid Memory Read)
- IMW: 無効なメモリーへの書き込み (Invalid Memory Write)
- メモリーリーク
- OLP: 送り側と受け側の重複 (OverLaPping source and destination)
- PIR: 部分的に初期化された領域からの読み取り (Partially Initialized Read)
- SBR: スタック境界を越える読み取り (beyond Stack Bounds Read)
- SBW: スタック境界を越える書き込み (beyond Stack Bounds Write)
- UAR: 非割り当てメモリーからの読み取り (UnAllocated memory Read)
- UAW: 非割り当てメモリーへの書き込み (UnAllocated memory Write)
- UMR: 非初期化メモリーからの読み取り (Unitialized Memory Read)

動的メモリーアクセス検査では、次の種類の警告が検出されます。

- AZS: 0 サイズの割り当て (Allocating Zero Size)
- メモリーリーク
- SMR: 投機的な非初期化メモリーからの読み取り (Speculative unitialized Memory Read)

コードカバレージの問題

コードカバレージ検査では、カバーされていない関数が特定されます。結果では、見つかったコードカバレージの問題に「カバーされていない関数」というラベルが付けられ、潜在的なカバレージの割合が示されます。この割合は、該当する関数をカバーするテストを追加した場合にアプリケーションの合計カバレージが何パーセント増加するかを示しています。

索引