PreEmptive Solutions

企業向け難読化ソリューション

テクノロジ、プロセスおよびコントロール

Gabriel Torok, PreEmptive Solutions
Sebastian Holst, PreEmptive Solutions
Publication Date: August 30, 2006

要約

このホワイト ペーパーでは、企業向け難読化（隠蔽）ソリューションの重要な特性を列挙し、その適合性と価値を、階層化セキュリティ プログラムにおいて、またリスクに基づく IT 制御フレームワークとして評価します。

具体的な内容としては以下の項目があります。

- 難読化技術の機能および考慮事項の要約
- 一般的なシステム構成管理の依存の特定
- アプリケーションのライフサイクル ワークフローおよびプロセス要件
- 開発、品質保証およびサポートの最優良事例
- 企業向け難読化がいつ有効な IT 制御となるかについてのガイド
- 企業向け難読化の評価基準

AG-TECH CORP.

企業向け難読化の 3 つの特質

企業向けの難読化は 3 つの明確な方法で測定することができます。それは、変換処理コレクション(テクノロジ)、開発ワークフロー(プロセス)、および、ソース コードの管理外の配布によって発生するリスクを削減する手段(コントロール)です。

テクノロジ

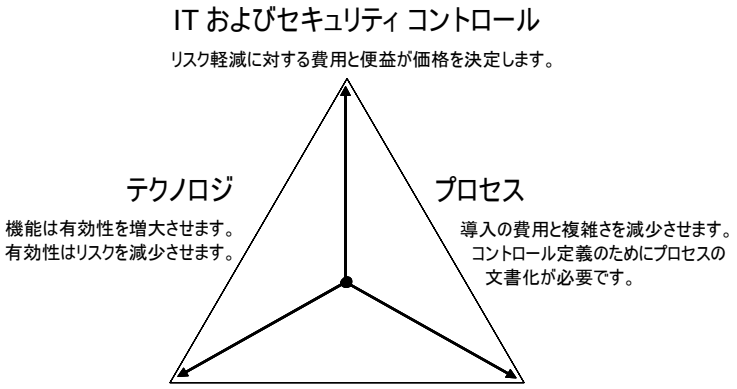
難読化は、1 つのバイナリから別のバイナリを生成するような変換のポートフォリオと定義できます。結果のバイナリは、人間が理解したり、あるいは逆コンパイラにより正確かつコンパイル可能なソース コードを出力することが実質的に非常に困難なものとなります。

プロセス

難読化は、アプリケーションのライフサイクルに組み込まれるプロセスと定義することができます。プロセス定義には、役割の適用範囲や IDE 統合要件のほかに、開発者の管理、継続的統合のサポート、ビルド設定と制御の定義、パッチ管理、品質制御、フィールド サポート統合に対応する特定の使用条件が含まれます。

コントロール

難読化は、ソース コードの管理外の配布に由来するリスクを管理するための補完的制御であると定義できます。これらのリスクには、システム攻撃の機会の増大、知的財産の損失、使用やその他の計量の実施を迂回することによる収益の損失が含まれます。



難読化ツールがいかに効果的であるかは、その計算機能および生産機能が立証します。難読化ツールは、難読化技術がより複雑で精巧になるほど、より効果的であることは明らかです。効果的な難読化ツールは、リスクを大幅に減少させます。

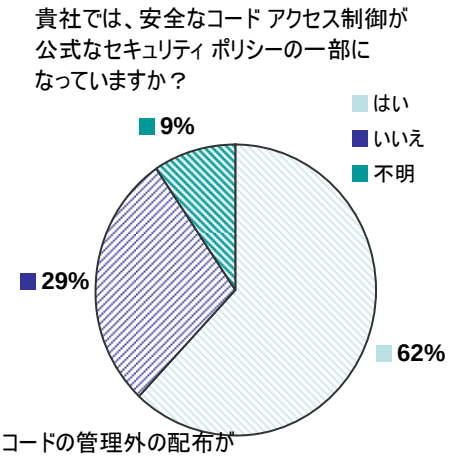
難読化プロセスは、広範な開発ライフサイクルおよび基礎をなす開発プラットフォームに統合されることにより、複雑さとコストを削減し、ひいてはリスクを削減します。管理の不十分なコンポーネントがほかの明確に定義されたプロセス内にある場合は、管理に費用がかかる上、それがどのような利益を約束していようと、その正当性が証明されることはほとんどありません。費用便益バランスの主要部分としての役割に加え、真正のリスク制御としての役割においては、難読化プロセスの文書が必要とされます。

難読化は、バイナリへのアクセス権はあるが、オリジナルのソース コードへのアクセス権がないコミュニティに対するソース コードの配布を制御する際に使用されます。.NET および Java のような、中間コードからソース コードを抽出することが簡単でよく知られている環境において、難読化はこのようなソース コード アクセス制御のギャップに対する一般的な防御手段です。法令遵守の問題として、管理とセキュリティへの関心は増大し、COBIT、ITIL および ISO のような公式な制御フレームワークの選択や厳守が劇的に増加しています。制御フレームワークに関係なく、各 IT およびセキュリティ制御の文書化は、法令遵守の基本的な第一歩です。難読化プロセスの理解と文書化、および当該プロセスの効果を明言する能力が必須です。

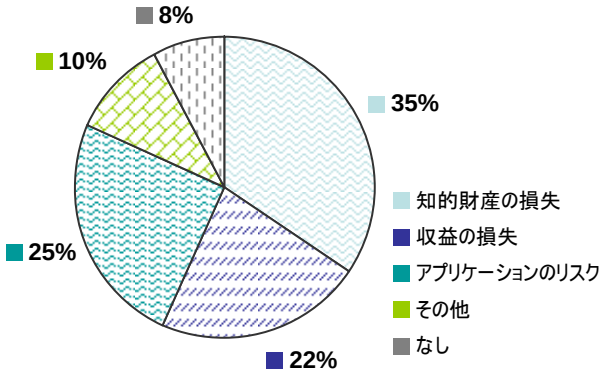
難読化が公式にそのように定義されているかどうかに関わらず、制御は難読化が果たす役割であり、増加するリスクおよび導入コストを計算に入れた上でのリスク回避の価値の費用便益分析は、成功への決定的な指標です。

世界的規模の 2000 法人だけでなく中堅、中小企業も含めた国際企業 3000 社に対する最近の調査では、以下のように報告されています。

- ソース コードのアクセス制御は、公式なセキュリティ ポリシーの一貫性のあるコンポーネントであると、回答者の 62% が断言し、否定したのはわずか 29% でした。
- ソース コードの配布が管理下にない場合、これに直接関連する 3 つの明確なリスクがあります。知的財産の損失は、その他の 2 つのカテゴリより若干高い頻度で言及されますが、収益の損失およびアプリケーションのセキュリティ脆弱性がさらされる脅威も明らかに重要です。

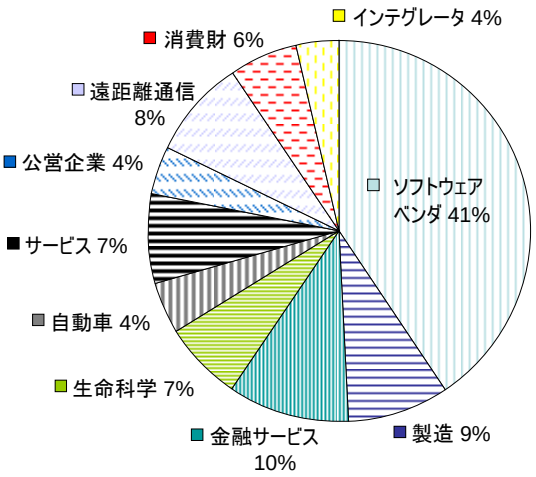


ソース コードの管理外の配布によって発生する周知のリスク、および、バイナリからのソースコードの抽出のハードルを劇的に下げた .NET と Java プラットフォームの導入を考慮すると、これらの環境で難読化が一般的な補完的制御として出現してきたことは驚くに値しません。



2 回目の調査では、2000 社がアプリケーション開発プロセス内に難読化プロセスを採用したことが確認されました。

予想どおり、難読化は、ソフトウェア開発に大きく依存する産業で最も多く採用されていることが調査により明らかになりました。ソフトウェアベンダ、金融サービス、製造業および遠距離通信業が多く採用した産業です。しかしながら、ソフトウェア開発と同様に、難読化がすべての産業に及んでいることは注目に値します。



難読化は広範な種類のリスクに対していつどこにでもある(ユビキタスな)反応として出現しています。

以下の章では、組織が難読化の必要性を判断するのを助けることに主眼を置いて、難読化の 3 つの特質について探り、効果的かつ能率的にリスクを減少させるために使用できる、企業向け難読化の選択基準の一貫したセットを提供します。

難読化のテクノロジ

難読化の最終目的は、「バイナリのリバース エンジニアリングによって、ソース コードへ不正アクセスする」という、特定の種類のアクセス制御違反を阻止することです。難読化は、バイナリを独自のデータ型として表すというユニークな要求と、従来のアクセス制御方法に対し潜在的に重大な問題を生み出すという要求に適応しながら、この目的を達成するように設計されています。

その効果を上げるため、難読化は、アプリケーションの動作やアクセス制御方針を変更することなく、リバース エンジニアリングを実質的により困難にする必要があります。実際のところ、難読化とは、これら 2 つの特徴を併せ持つ要求に適合する一連の技術を表す一般的な用語なのです。

コントロール	バイナリ アクセスとの関係
限定された読み取り/実行アクセス	• バイナリを見つけたり実行することができません。バイナリを実行する必要のあるグループに対してはこの方法を無効にします。 • 役割と証拠に基づくセキュリティ構造は、バイナリが制御されている、または配布されない状況では、効果的に読み取り/実行アクセスを制限できるかもしれません。
暗号化	• 伝送とストレージのみに有効 – バイナリは実行するためには復号化する必要があります。復号化によりバイナリは元の状態になります。 • 暗号化はその対象に関する高度な防御を提供します。
難読化	✓ アクセス制御設定とは無関係であるため、バイナリの実行を制限しません。 ✓ 伝送中および実行中、ディスク上で有効かつ永続的です。 ✓ 難読化技術の有効性は、変換の種類によってさまざまです。

難読化の方法

難読化は、人間や逆コンパイラがプログラムの機能を理解するのに使用するコンテキストを削除します。以下の「切符は何枚買えばよいですか？」という問いに答える 2 種類の同一メッセージについて考察します。

「明瞭な」電子メール	擬似「難読化された」電子メール
From:Mark To:Bill 件名:RE:切符は何枚買えばよいですか？ Bill、既に 15 枚買ったので、あと、必要なのは 7 枚です。 約束どおり、今夜あなた方みんなと会うのを楽しみにしています。	To:Bill From:Mark 7

どちらのメッセージも配信されて Bill の質問に答えていますが、「難読化された」方はそのコンテキストが取り除かれています。不正な読み手には、これが質問に対する答えであることや、Bill が 7 枚の切符を買うように頼まれていること、既に 15 枚の切符が購入済みであること、グループ全員がその晩会合を開くことなどはわかりません。

わかりやすく言えば、元の状態と動作を維持しながら、情報を取り除き構造を変更する複数の変換を適用することにより、コードを人間や逆コンパイラにほとんど理解不能にすることが、難読化の機能です。

代表的な難読化技術

以下の技術が、同等の動作をする 1 つ以上のバイナリ セットを生成するために、ソース コードではなくコンパイル済みのバイナリ群に適用されます。

識別子の名前変更

クラス、インターフェイス、メソッド、フィールド、メソッド パラメータ、ジェネリック タイプ パラメータなどのプログラム識別子の名前が変更され、名前の持つすべての意味が取り除かれます。たとえば、"GetPayroll" は "a" になります。さらに、高度な名前変更機能では、各識別子のスコープを解析し、重複利用されない複数の識別子間で名前を再利用することにより、新しい名前を大規模にオーバーロードします。つまり、使用されているどの "a" も、実際にはまったく異なる識別子が同一の新しい名前に変更された結果であり、あらゆる識別子が個別に分析されるため、この方法はセキュリティを強化します。

制御フローの難読化

高レベルなソース フローが、オリジナル ソースを厳密に反映する一連の命令にコンパイルされます。制御フローの難読化は、新しいシーケンスがオリジナル ソースへ逆コンパイルされないよう意図的に副作用を使用して、オリジナルの命令シーケンスを論理的に同等なシーケンスに変換します。逆コンパイラは多くの場合間違ったコードを出力するか、クラッシュ、または例外で停止します。

メタデータの除去

アプリケーションの実行にはすべてのメタデータが必要なわけではありません。たとえば、.NET プラットフォームでは、いくつかのプロパティ、イベント、メソッド パラメータ名、およびカスタム属性は取り除くことができます。実行に影響を与えず開発プロセスに関連する情報および開発者の意図を削除します。

文字列の暗号化

アプリケーション全体を暗号化することにはよく知られた制限事項がありますが、難読化の際の出力バイナリ内の文字列を暗号化することは、難読化に対するアプローチに効果的な層を 1 つ追加します。暗号の解除は、あらゆる文字列読み込み命令の直後に解除メソッドの呼び出しを埋め込み、実行時にスタック上にある暗号化文字列を「明瞭な」文字列に置換することで行います。これにより、暗号化の便宜を提供しつつ実行時の強力な保護をも提供しています。

システム構成

以下のコンポーネントは、ビルド プロセス内で挿入することができる企業向け難読化機能の基本要素を表します。ほかの構成も確かに可能ですが、この方法は複数のライブラリ、アセンブリ、パッチ生成および分散型開発ワークグループに適応させるのに十分な柔軟性があることが立証されています。

入力プログラム
.NET exe または DLL (MSIL)

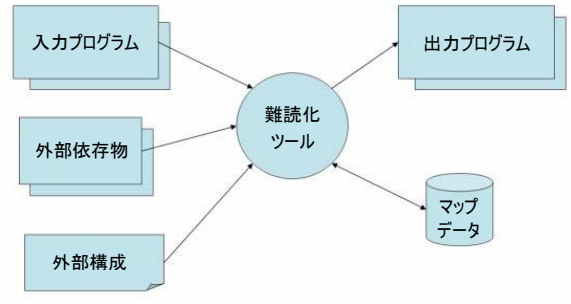
外部依存物
依存する DLL

外部構成
XML 構成ファイル

難読化ツール
コンパイル後保護システム

出力プログラム
よりサイズが小さく、リバース エンジニアリング困難な .NET exe または DLL (MSIL)

ビルド マップ
名前変更がどのように行われ、どのような属性が適用されたかについての情報を含む XML ファイル



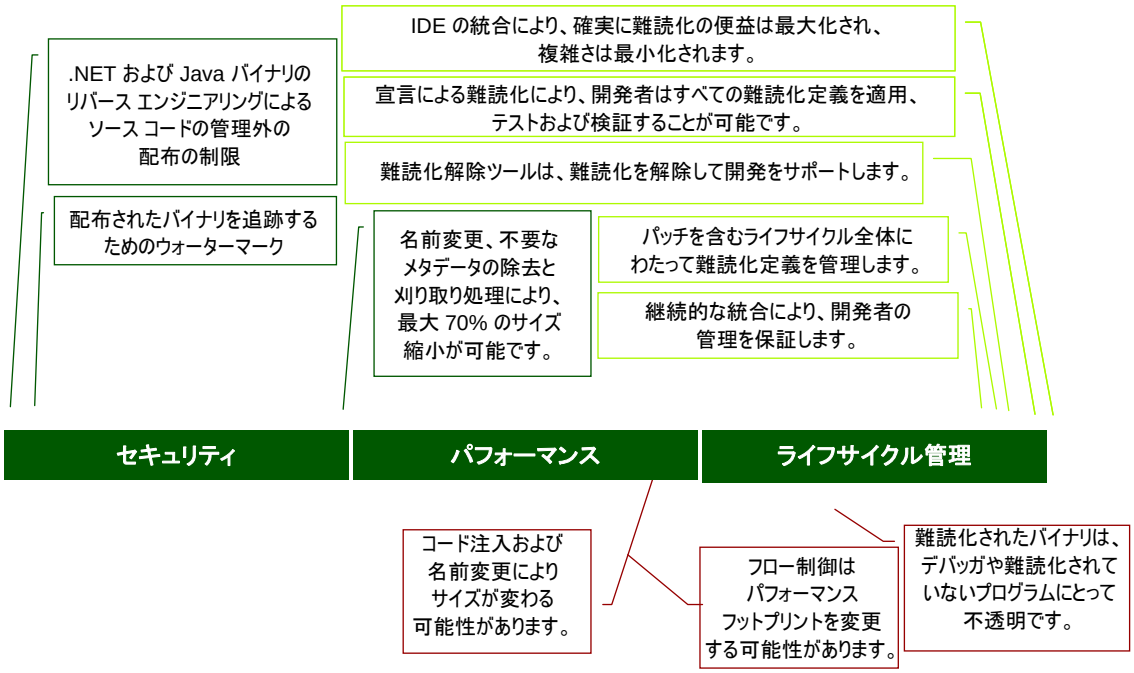
難読化の副作用、含意および最優良事例

難読化の変換処理は、意図的に、バイナリの見え方が大きく変わるようにしています。効果的な難読化は、重要なセキュリティとパフォーマンス上の利点を提供します。ただし、難読化されたバイナリに区別はなく、「悪しき人々と悪しきプログラム」に不透明であるのと同様「善き人々と善きプログラム」にも不透明です。

難読化されたバイナリを完全なアプリケーション開発のライフサイクルに含めることでもたらされる結果を予想し対応することによって、確実に難読化の便益は最大化され、関連する複雑さやリスクは最小化されるでしょう。

ソース コード配布と難読化に関わるコントロールのオプションとリスクは複雑です。概要説明のため、次の図に難読化のセキュリティ、パフォーマンスおよびライフサイクル管理上の便益、密接な関係、および潜在的な副作用についてまとめてあります。濃い緑色の部分は結果の好ましい機能を示し、赤色は特定の難読化ソリューションの選択または実装の前に査定するべき注意を示しています。明るい緑色は開発のライフサイクルに難読化を組み込むことの複雑さとリスクを補完または最小にする機能を表しています。黄色の特性は、主に、企業向け難読化手法と難読化ユーティリティとを区別するものです。

機能および結果については、この図の後のセクションで説明します。



デバッグ

難読化の目的は、バイナリを元のソース コードへ結び付けるのをできる限り困難にさせることです。開発、QA およびサポートで難読化されたバイナリを解くことを可能にする、明確で信頼できる「難読化解除」ツールがなければ、デバッグ処理は難読化によって困難にされるか妨害すらされかねません。

パフォーマンス

難読化は、アプリケーションのパフォーマンス フットプリントを変化させる可能性があります。制御フローの難読化は、きめ細かく調整されたアルゴリズムの品質を場合によっては下げることがあります。逆に、識別子の名前変更は、アプリケーション全体のサイズ(したがって、読み込み時間)を減少させる可能性があります。開発者は、アプリケーション内できめ細かいレベルで選択的にテストを行って、予期しない副作用の可能性を排除しながら、難読化の最大の便益を確実に達成できるようにする変換処理を適用するための手段を必要とします。

サイズ

難読化はアプリケーションのサイズも変更することがあります。文字列の暗号化手法の組み入れはアプリケーションの全体的なサイズを増大させるでしょう。しかし、難読化の最も一般的な結果は、アプリケーションのサイズを減少させることです。識別子の名前を短くすることに加え、難読化ツールは、アプリケーショ

ンとサード パーティ製のライブラリに含まれているけれども一度も呼び出されない部分を「刈り取り」ます。刈り取り処理および識別子の名前変更の効果的な使用により、一般的にアプリケーションのサイズは 30 ～ 40%、時には最大 70% も削減されます。

パッチ管理

名前の変更記録やその他の記録は難読化処理ごとに独自である可能性がありますが、将来のビルド時に互換性を確保するため、これらは繰り越される必要があります。このことは、異なる時期と場所でビルドおよび難読化されるパッチおよびコンポーネントにも適用されます。適切な難読化ユーティリティを使用した IDE への効果的な統合により、開発ライフサイクルとこのタッチ ポイントの管理が簡素化されます。

最優良事例

開発ライフサイクルに難読化を取り入れる最も効果的なアプローチは、適度な認識によって、ツールと処理の拡張部分を結び付けることです。このセクションには、洗練された開発組織がいかに効果的であるかを理解するための最優良事例が要約してあります。

宣言による難読化

開発者は誰よりも自分たちのコードをよく知っています。意図されざる、あるいは期待されていない副作用を持つことなく、確実にソース コードを正しく難読化する最も効果的かつ効率的な方法は、開発者がどの難読化変換処理を、コードのどこに適用すべきか(あるいは適用すべきでないか)を指定(宣言)することを可能とする一連の属性を提供することです。原理的には、これらの属性は開発者の使用する IDE によって認識され、追跡、認証および共有されます。たとえば、.NET Framework のバージョン 2.0 はまさにこの目的にかなう属性を提供しています。この手法の利点には以下のようなものがあります。

- モジュールの個々の開発者は、名前変更、文字列の暗号化、制御フローの難読化などの変換をコード内のどこに適用するかを指定することができます。
- ビルド時の構成ファイルを除去することができます。
- コンポーネントの提供者と再利用可能なコンポーネントの開発者は、その他の人が使用できるように難読化要件をコードに埋め込むことができます。

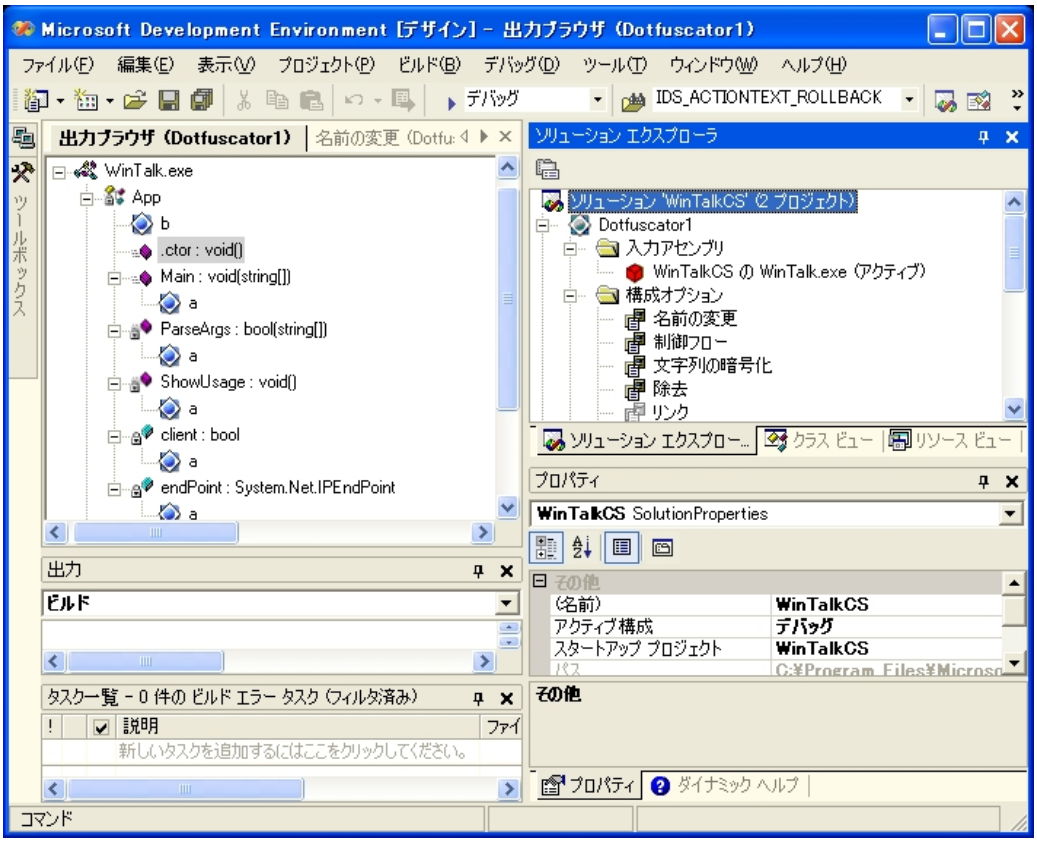
分散化された安全なデバッグ

開発者、QA およびサポートの人員にはすべてバイナリをデバッグする機会があります。実際上、開発プロセスの中でバイナリのデバッグを行わないのは、一般的に製造または組み立て段階のみです。難読化ソリューションは、完全な難読化ソリューションやビルド環境へのアクセスを必要としない、難読化されたバイナリの分散化デバッグをサポートする、費用効率の高い軽量で低維持費のユーティリティを提供する必要があります。難読化解除ユーティリティは、難読化されたアプリケーションのためのデバッグ シンボル ファイルを受け入れ、この情報を使用して、難読化されたアセンブリを 1 行ずつ「解読」していきます。この方法の利点は以下のようになります。

- 既存の開発、品質およびサポートの機能およびプロセスは中断されません。
- ビルド環境は、通常サポートされないデバッグ活動によって危険にさらされることはありません。
- 重い難読化ソリューションを広範に配置、管理、ライセンスする必要がありません。

IDE 統合

統合された難読化ユーティリティは、統合開発環境の一部として動作します。たとえば、Visual Studio ソリューションはネイティブ プロジェクト タイプとして認識され、入力ファイルを 1 つ以上のほかの Visual Studio プロジェクト(C# や VB.NET プロジェクトなど)から受け取り、すべての依存関係と権限を自動的に解決できるはずです。



この画面は、統合難読化機能を使用する IDE の例を示しています。以下の機能が表示されています。

- 難読化ユーティリティを .NET プロジェクト(C#、VB.NET など)に接続する機能
- 新しい名前を表示する機能 ー ほとんどのメソッド名が "a" に変更されていることに注目してください。
- 出力ログを表示する機能

IDE 統合の利点は以下のとおりです。

- ビルド工程エラーの減少
- デバッグおよびパッチ管理をサポートするための、直前の難読化マップのトラッキング機能の向上
- 自動化、プロセスの透明性、および品質の向上

パッチ管理

パッチ管理は、統合アプリケーション環境を管理する企業の開発チームが特に関心を持つ機能です。名前の割り当て記録を難読化時に生成することにより、以後の難読化時に同じ API の名前が同じ難読化名となるようにできます。またアクセス対象となるポイントの名前変更はビルド間で一貫性があるため、部分的なビルドが可能となります。この手法の利点には以下のようなものがあります。

- パッチの生成および配布プロセスが中断されません。
- 既に配布済みで変更されていないモジュールは難読化や配布を再度行う必要がありません。

継続的な統合

継続的に統合開発手法を組み入れてきた組織は、単体テストを行う開発者のために難読化手順を含めるべきです。これは、開発者が宣言による難読化を利用していて、特定の難読化変換を適用した際の影響を評価しようとする場合に特に当てはまります。この手法の利点には以下のようなものがあります。

- 開発者は、自分のコードを難読化した場合の影響と有用性について、その作業結果を調べる前に評価することができます。
- ビルドやリンク、およびその他の依存対象を、開発ライフサイクルの初期段階に確認および解決できます。

その他の機能と利点

難読化機能には、洗練された静的なバイナリ解析が必要です。また、効果的な難読化に必要とされる技術は、難読化と並行して適用可能な多数の機能も必要とします。そのような機能には、以下のようなものがあります。

不要コードの除去

不要コードの除去は、使用されていないコードの識別と削除です。難読化分析の一部として、不要な要素を判別することも可能です。さらに、圧縮または不要コードの除去プロセスは、未使用クラス、メソッド、インスタンス変数、デザイン時メタデータ、および実際の MSIL を削除することができ、より小さなアプリケーションを作成できます。多くのアプリケーションで不要コードの除去によるサイズ縮小効果は著しいものであり、最大 70% までの縮小が期待できますが、一般的にはサイズ縮小効果は 30% から 40% 程度になります。

ウォーターマーク

ソフトウェアのウォーターマーク処理は、顧客 ID や著作権情報をソフトウェア アプリケーションに隠して、ソフトウェアの所有者を特定したり、海賊版のコピーのオリジナルを追跡するのに使用することができます。ソフトウェアのウォーターマーク処理は、楽曲、映画、画像など、その他のデジタル コンテンツのウォーターマークと同じようなものです。難読化ツールは、まさにその本質から、その操作の検出または変更が困難であるため、ウォーターマークを挿入するのに特に効果的なツールです。

リンクおよびパッケージ化

難読化ツールのオプションを使用すると、複数の入力アセンブリを 1 つの出力アセンブリに結合することができ、配布とインストールを容易にします。

企業向けの難読化プロセス

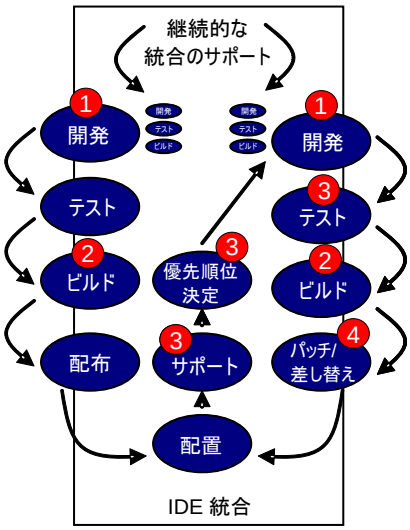
難読化プロセスは難読化機能をアプリケーションの開発ライフサイクル、具体的に言うと開発、テスト、統合、製造、アプリケーション サポート、および開発時のパッチ管理をサポートするサイクル内に統合します。

アプリケーション ライフサイクル統合

バイナリと共にソース コードを配布しない組織は、システム攻撃、知的財産の盗難、およびライセンスの施行や計量機能の迂回行為による利益損失のリスクを減少させるためにバイナリを難読化します。

難読化は、適切に適用すれば、ソース コードの管理外の配布によって起こるリスクを低減させるための 1 手段となります。

このページの図は、明確なアプリケーション開発プロセスの中で、リスクを減らすために、熟慮した難読化ソリューションが保守する必要のある複数のタッチ ポイントを示しています。



❶ 宣言による難読化制御

作成したコードを最も熟知しているのはその開発者です。XML で定義した属性を使用することで、どの変換を、どのように、アプリケーション内のどこに適用するかを指定することができます。宣言による難読化は精度を任意のレベルに設定できるため、最大の難読化レベルを最小の副作用で行うことを可能としています。

❷ 難読化変換の設定と適用

サプライチェーン マネジメント管理者またはビルド管理者が最終的な変換の設定と実行を行います。これらは、単体テストおよび一連の統合処理のサポートを含めるように拡張可能であるべきです。

❸ 効率的なデバッグのための難読化解除

難読化されたバイナリを「逆変換」可能な分散機能。この機能はオリジナルのソースコードへ戻すための有効な関連付けを提供します。デバッグ行為がビルドしたマシン上に制限されていては、あまり効率的であるとは言えません。

❹ 増分難読化の変換処理

過去に難読化および分散化されたバイナリとの互換性を維持しつつ、ソースへのアクセスを抑止するような、パッチの難読化機能。

好適な IDE との統合および今日の分散作業環境をサポートする配置の柔軟性は、効果的な難読化プロセスにおける決定的な要素です。

補完的制御としての難読化

あらゆる産業の組織は、直面する多くの調整、管理およびその他の法令遵守の責務を満たすため、難読化を彼らの IT 制御業務に組み入れています。

難読化の最終目的は、許可されない個人が理解したり変更することが実質的に困難なプログラムを作り出すことです。難読化は、Java および .NET 環境ではよく知られている、ソース コードに対するアクセス制御のギャップを埋めるための「補完的」な制御です。

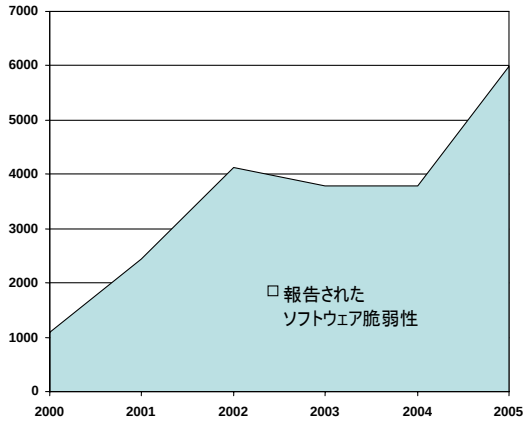
難読化の必要性の判断

ソース コードを知る必要があるのは誰か

情報へのアクセス権は、「知る必要性」を基準とし、「最小の権限」のアクセス制御運営ポリシーの一部として付与するということが、ほぼ普遍的なコンセンサス[1]です。Java や .NET のような現代の環境におけるバイナリからのソース コード抽出の容易さを考えると、いまや「ソース == バイナリ」は事実上トートロジー(同意反復)なのです。考えるべき問題点は、ソース コードを「誰が知るべきか」であり、知るべきコミュニティが、対応するバイナリへの読み取りアクセス権を持つコミュニティより小規模なものであるかどうかです。極端な実例の 1 つに、オープン ソース コミュニティがあります。設計上ソース コードが広く利用可能な場合、難読化は明らかに何の価値もありません。逆に、多くのソフトウェア ベンダや金融サービス業者、製造業者、防衛産業業者などはソフトウェアを頼りにその使用(アクセス、支払い等)の計量やプライバシーの保護、処理の継続性の確保を行い、独占的なビジネス ルールやプログラミング ロジックなどをソース コードに埋め込んでいます。これらはすべて一般的に物質的価値を表し、これらの損失および機能不全は物質的リスクを表します。

アプリケーション セキュリティの階層化手法

バイナリが広く配布されている場合には、SQL インジェクション攻撃の受けやすさ、バッファ オーバーフローなど、アプリケーションの脆弱性について関心が高まります。報告される脆弱性の数は、CERT/CC 統計[2]によれば過去 6 年間でほぼ 600% に増大しました。



(可能な場合に脆弱性を除去する)防衛プログラミングがおそらく最も効果的な制御である一方、難読化は未知の(または未検出の)脆弱性から保護するための追加のセキュリティ層として使用されます。階層化セキュリティは広く受け入れられている最良の方法です。銀行は金庫をビル内で鍵のかけられたドアの後ろに置き、プロの警備員によって守り、さらに警察によって監視されることによって守り続けてきました。

制御下にないソース コード配布によって生じる事態の特性を知る

ソース コードのアクセス制御は以下の事柄から生じるリスクを軽減するように設計されます。

- システム攻撃および情報窃盗が成功する可能性の高いコーディングの暴露

[1] 内部監査人協会の指針であるだけでなく、COBIT、ITIL および COSO 制御フレームワークにおいても同様です。

[2] CERT(www.cert.org)は、ネットワーク システムでの攻撃を阻止するために適切なテクノロジおよびシステム管理が使用されること、被害を抑えること、攻撃の成功や事故、障害にもかかわらず、重要なサービスの継続を確実にすることに専念している組織です。

- 企業価値の減少、競争の激化、法的防衛費用の増加を招く知的財産の公表
- ソフトウェア ライセンスの施行、あるいは製品およびサービスを制御する計量機能を回避し、収益や収益性を減少させるような「その場での」アプリケーションの変更

投資収益率の見地から見れば、制御を実装する費用とリスクの差異が、制御を実装しない場合の費用およびリスクより意味を持つ程度に低い場合に、制御は正当であると認められます。潜在的な損害やさまざまなリスクの発生の可能性については、ケース バイ ケースで確定する必要があります。

障害を最小限にするための「注意義務」の実証

リスク管理とセキュリティ プログラムは投資収益以上のものを提供します。これらは顧客、従業員および投資家に対する広範な義務の一部です。この広範な義務を軽んずることは責務の不履行と解釈され、場合によっては重大な障害をもたらします。監査役、監視委員および法廷は、組織が法令遵守とセキュリティのプログラムをどれほど効果的に予防、修復、監視および継続して改善しているかを計るため、行動の基準に注意を払います。リスクや障害を軽減するための効果的で持続するプログラムを明確に示すことのできる組織は、これらを減少させ、時には回避することすら可能です。

当然払うべき注意というのは、以下のように徐々に高い防壁を立てていくことによって実証されます。

I. 攻撃側に付け入る隙を与えない

II. 悪意のある者を妨害する

III. 攻撃側の熟練を阻止する

IV. 攻撃成功を特定し、処罰可能とする

難読化は上記 I を達成する方法であり、これに失敗した場合、難読化の難度を II から IV まで増大させます。

なぜ難読化の制御を含めるのか？

- セキュリティに対する階層型の手法は広く認められた最良の方法です。
- 難読化ソリューションを実装する費用は、アプリケーション開発への投資のうちのほんのわずかです。
- システム攻撃、知的財産の盗難および収益損失から発生する操作および財務上の規制されるリスクは壊滅的である可能性があります。

効果的に実装された難読化プロセスは、Java および .NET バイナリからソースにアクセスすることによって発生する物的リスクに対して、少ない費用であまり世話のかからない制御を提供します。

企業の難読化評価基準

実際には、あらゆる産業の組織が難読化をアプリケーション開発ライフサイクル プロセスに組み入れています。アプリケーションのセキュリティ、IT 管理とリスク管理プログラムは、ソース コードの管理外の配布に対する保護の必要性を強く認識させました。このような認識の高まりが、広範なセキュリティ、IT 管理およびリスクに基づく第一歩の中で、難読化を効果的に使用するために何が必要かについて、より包括的な理解も生み出しました。

あらゆる組織は、これらの問題を管理するために適切な処理と制御のセットを開発して維持していく必要があります。この広範な要求は根本的に普遍的なものですが、アプリケーション セキュリティに「1 つのサイズがすべてに適合する」ソリューションはありません。それぞれの組織は、適切なプロセス、コントロールおよびテクノロジのセットを決定するためにその組織に特有の必要性を評価する必要があります。

以下の 5 つの高レベルの質問を使用すれば、企業の要求に適切かつ効果的に適合する難読化ソリューションとするための、重要な特性を捉えることができます。

難読化ソリューション プロバイダへの質問

1. 難読化ソリューションは、貴社のアプリケーション ライフサイクルおよび継続的統合フレームにどのように適合しますか？
   - 開発者は制御を失うことなく最大の難読化を確実にするために、コードにマーク付けすることができますか？
   - 難読化されたバイナリに対するデバッグおよびパッチ リリースのソリューションは何ですか？
   - サポート、業務、QA だけでなく提携開発プロセスにも適合する、分散化され、かつ安全な配置モデルはありますか？
2. どのような難読化、圧縮、およびウォーターマーク技術を利用できますか？
   - それらは独自のものですか？特許を取得していますか？
3. 複数のプラットフォームをサポートしていますか？
   - .NET および Java はサポートしていますか？そのテクノロジは IDE に統合されていますか？
4. そのソリューションは、対象となる組織の規模や部門配置に適合するように構成されていますか？
5. どのようなサポート、アップグレード、および品質保証が約束されていますか？

これらの質問に対する「正しい」答えは、そのソリューションがどこでどのように配置されるかによって明らかに異なります。程度の差はあれ、まさに同一の理由から、さらにほかの答えが与えられるでしょう。このような受入基準および価値の可変性は、長期間にわたり、組織をまたいで、また単一の組織内で適用されます。

明らかになってくるアプリケーションの潜在的寿命と異種の開発習慣を考慮すると、難読化ソリューションのプロバイダは、持続的な技術のリーダーシップ、柔軟なプロセスおよび制御能力、そして信頼できるサポート インフラストラクチャを確保できるよう、製品を開発し、ビジネスを体系付け、戦略を調整することが特に重要です。

まとめ:難読化ツールと煙探知機

実証的証拠は疑う余地がありません。何万もの開発グループが開発過程に難読化を含めるという決定を下してきました。利益損失、操作の混乱、および知的財産の損失が、ソース コードの管理外の配布から起こる現実の脅威です。このことが、これらのリスクについてなぜもっと聞こえてこないのか、また、潜在的な損害がそんなにも大きいのであれば .NET および Java は現代の開発プラットフォームとしてなぜ最速の成長を遂げ続けているのか、という論点を巧みに避けています。

難読化は家庭内の煙探知機に似たものです。

答えは、潜在的な損害の論争の中ではなく、発生の可能性の中にあります。発生する可能性が低い災難をもたらす脅威は新しい事象ではなく、この種のリスクを管理する立証済みの手法があります。低費用で保守の手間がかからず、随所で目にする煙探知機は、費用が高く発生の可能性が低いリスクを効果的に管理する方法の日常的な例です。

住居の火災は生命と財産の両方を脅かしますが、比較的稀な出来事です。煙探知機は、損失を減らすという証明付きの能力と、比較的発生する可能性の低いことに見合った低費用および最低限の保守処理を組み合わせることを提案しています。

難読化ソリューションの費用は、それが防ぐ潜在的損失に対し、ほんの一部にしか相当しません。

ほとんどの人々が煙探知機なしで一生を送ることができるかもしれませんが、適用範囲が広く着実なやり方でこのリスクを軽減することによって、社会はより良いものとなります。それが、有効な煙探知システムの証明なしで家を販売することがしばしば不法となることの理由です。

Java および .NET におけるバイナリの難読化は、ソース コードが広く配布されることのない環境においてはアプリケーション開発プロセスの主要部分であるはずです。できることなら、どの開発チームも難読化が提供する保護を必要としないことを望みます。しかし、いくつかのチームは難読化によって利益を得、この簡単な手順を省いたチームの費用が実際に非常に厳しいものとなる可能性があることもわかっています。

火事の危険を避けるために家に住むのを防止するのは不合理ですが、わかっているリスクを最小限にするために簡単な安全策をとらないことも無責任であり、法の管轄区域によっては違法です。

.NET および Java 環境は、バイナリからのソース コードの抽出という簡単に管理できるリスクを補って余りある、はるかに多くの利点を提供します。

もちろん、ホテルやレストランでは、損失および増加する発生の公算が共に高い可能性があります。結果として要求されるものが高ければ高いほど、処罰もそれに比例して高くなります。

ソース コードの管理外の配布には、産業区分によって分けられるさまざまなリスクの程度があります。ソフトウェア ベンダ、金融サービス業者、遠距離通信業者、および製造業者などの、アプリケーションによって収益を生み出し、ビジネスの継続性を保証している上、そのアプリケーション自体が独自の知的財産となっているビジネスでは、抱えるリスクがより大きくなり、障害発生時にはそれに比例したより高い要求と厳しい処罰を伴います。

著者について

Gabriel Torok は PreEmptive Solutions, LLC の社長であり、書籍の著者、および全国規模のカンファレンスにおける講演者です。Gabriel はアプリケーションのセキュリティに関するさまざまな記事を .NET Developers Journal、MSDN Magazine、Visual Studio Magazine The Code Project などに発表しています。彼のブログは http://www.dotnetjunkies.com/WebLog/obfuscator に掲載されています。PreEmptive Solutions は Java および .NET のコード セキュリティ ツールを製造しています。PreEmptive 社の Dotfuscator ツールのライト バージョンが Microsoft の Visual Studio 2005 にバンドルされています。Gabriel Torok への連絡は gtorok@preemptive.com から行えます。

Sebastian Holst は現在 PreEmptive Solutions の販売およびマーケティング担当の副社長で、販売、製品管理およびマーケティングの責任者です。Sebastian はまた、Open Compliance and Ethics Group (www.oceg.org)の技術プログラムの責任者も務めています。Ethics Group は NPO 団体で、組織が統制、法令遵守およびリスク管理活動を調整してビジネスの効率を高め整合性を推進することを目的として掲げています。Sebastian は著名な技術解説者で、Gilbane Report(www.gilbane.com)の編集主任として多数のホワイト ペーパーおよび Web サービスから仮想リポジトリに関するトピックの記事を出版してきました。Sebastian は、効率的な企業管理、リスクおよび法令遵守管理を促進するように計画された The Compliance Consortium(www.thecomplianceconsortium.org)の社長および共同設立者であり、5 年間にわたって W3C の顧問委員でした。Sebastian はアプリケーションのセキュリティ、企業データベース、コンテンツ管理、XML およびビジネス インテリジェンスの技術と標準を 15 年間開発してきました。Sebastian へは sebastian@preemptive.com から連絡を取ることができます。

PreEmptive Solutions について

PreEmptive は、アプリケーションの価値を高める使命を持って 1996 年に設立され、Dotfuscator および DashO 難読化ツール ファミリを世に送り出してきました。2,000 を超える顧客企業、100 か国以上での 20,000 件の登録済みインストールに加え、Microsoft の 6,000,000 シート以上の Visual Studio へのバンドルによって、PreEmptive Solutions は誰もが認めるマーケット リーダーであり、ソース コード保護、アプリケーション セキュリティおよび IT ガバナンスが深刻であるあらゆる企業にとって明確な選択肢となっています。

PreEmptive Solutions の詳細については、電子メール solutions@preemptive.com または電話 +1 216 732 5895 でお問い合わせください。

AG-TECH CORP.
Smart Software, Smarter Deployment
株式会社エージーテック
本社 東京都千代田区神田錦町 1-21-1 昭栄神田橋ビル 3F
TEL:03-3293-5300 FAX:03-3293-5270
URL http://www.agtech.co.jp/

545-G9-001