PreEmptive Solutions

アプリケーションの保護は必要なのでしょうか？

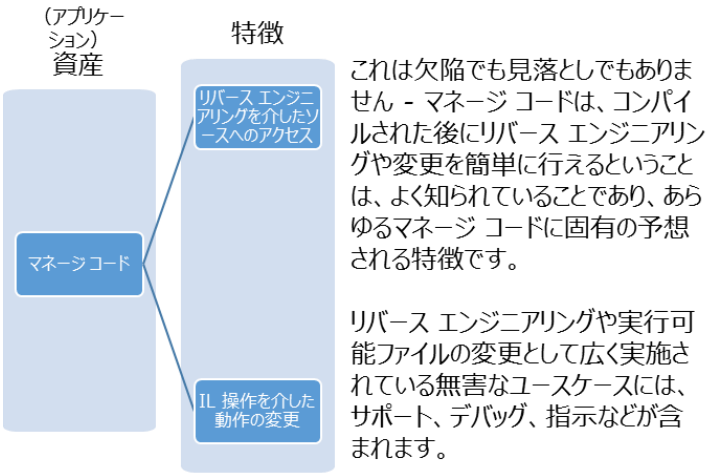
開発者（またはその所属組織）は、なぜ自身のアプリケーションをわざわざ保護しようとするのでしょうか？PreEmptive Solutions 社がアプリケーションのセキュリティやリスク管理のソフトウェアを構築しているという事実を考えれば、これは皮肉めいた、大げさな問いかけのように思われるかもしれませんが、そうではありません。

そのような質問に答えるための唯一の方法は、まず、何から保護する必要があるかを知ることです。「リバース エンジニアリングや改ざんから保護する」などと思われたのであれば、それは有意義な答えとは言えません。答えには、悪いことが起こった場合に、どのような問題が発生するかについての考慮が必要です。あなたが目を向けているのは、著作権侵害を防ぐことですか？知的財産の盗難ですか？これらも十分とは言えません。真の答えは、財政的な打撃やその他の損害をもたらす、収益の損失、運営の混乱に関わるものでなければなりません。これらの質問に答えられない限り、そのリスクへの対応について、適切に優先順位を付けることは不可能です。

ここまでで、細かいことにうるさいだとか非実用的すぎると思われる方は、（これを言って気を悪くされたら申し訳ありませんが）この種の意思決定に携わるべきではありません。その一方で、このような質問の答え方はわからないが、（直観的であっても）リスク（損害）の管理と、リスクを増大させる可能性のある事象の防止との違いを理解している方には、以下に示す .NET や Java（マネージ コード）での開発から生じる独自のリスクを管理するための的確な取り組み方法がお役に立つことを願っています。

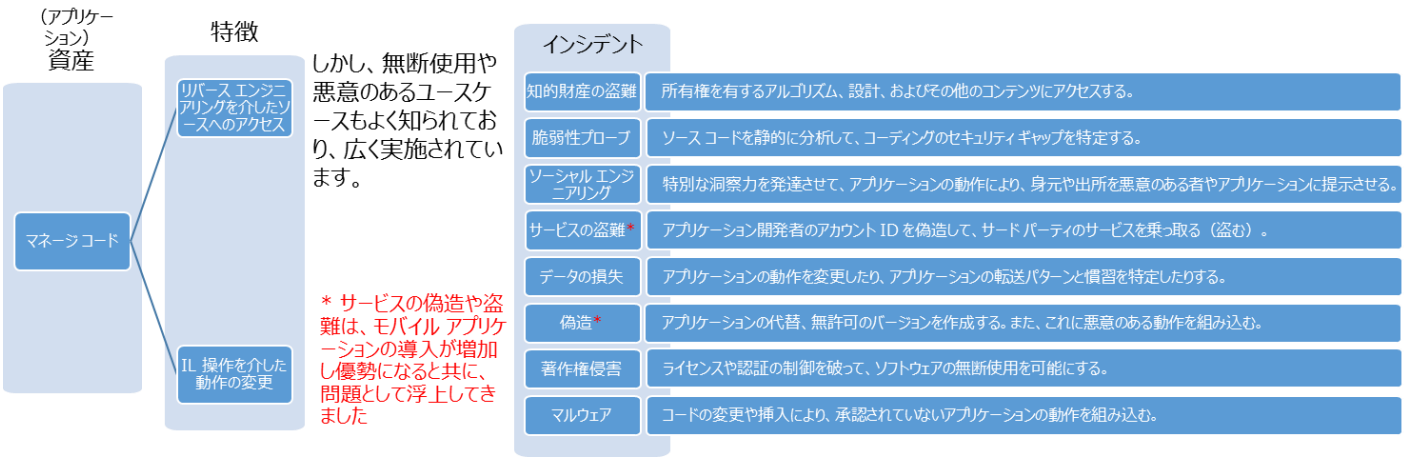
第 1 の考慮点:マネージ コードのリバース エンジニアリングや変更は、意図的に簡単になっています。これが利点となっている多くのシナリオが存在します。

リバース エンジニアリングや実行可能ファイルの操作はよく知られており、広く実施されているものであることを、上級管理者は理解する必要があります。



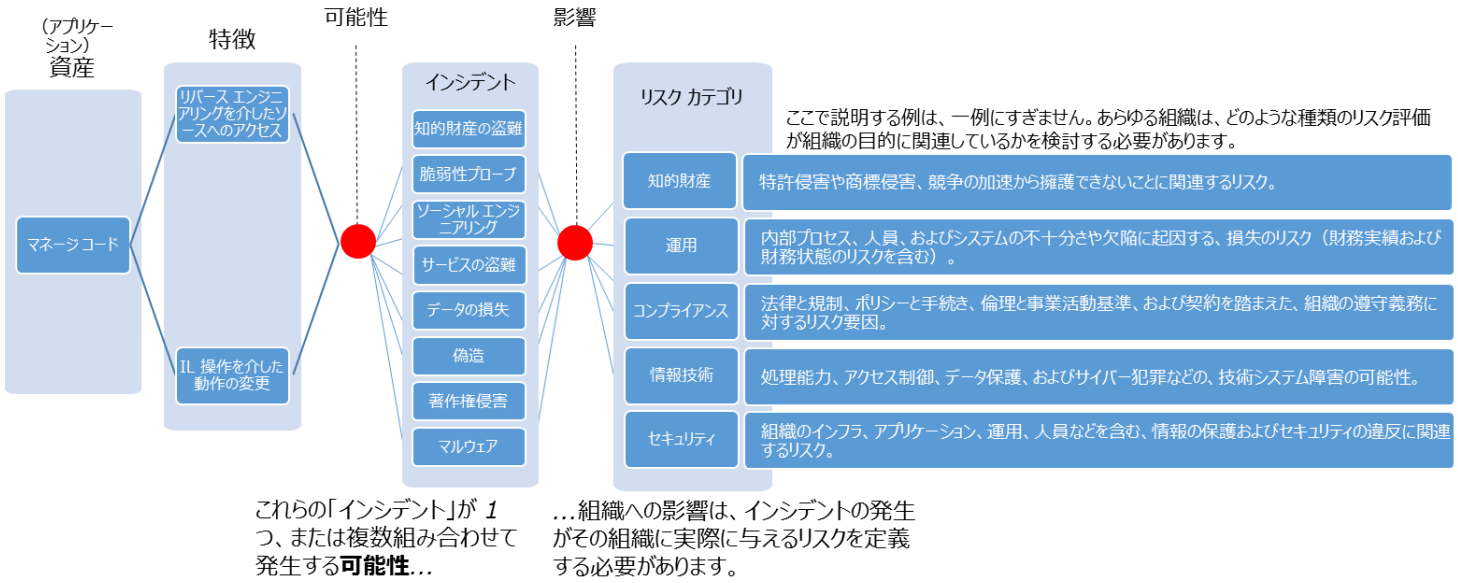
したがって、この一般的な方法が組織に物質的なリスクをもたらす場合は、これらのリスクを軽減するための措置を取らざるを得なくなります。もちろん、マネージ コードのこの基本特性が物質的リスクをもたらさないのであれば、措置を講じる必要はありません(また推奨もしません)。

第 2 の考慮点:リバース エンジニアリング ツールに罪はありません。犯罪者が罪を犯すのです。しかし、犯罪者は、リバース エンジニアリング(および他のカテゴリ)ツールを用いた犯罪方法を数多く見つけてきました。



適切な戦略を提言できるようにするためには、すべての脅威をリストアップする必要があります。知的財産の盗難が 1 つの脅威であることを知っているだけでは不十分です。偽造アプリケーションの流通により脅威が高まるのであれば、これも脅威の対象として捉える必要があります。

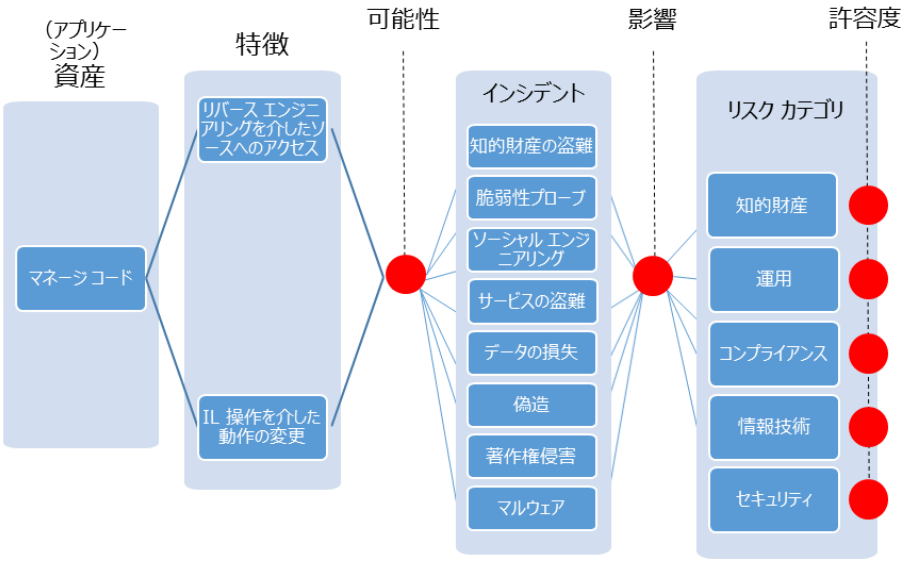
第 3 の考慮点: 上記のインシデントの種類のうち、どれがご自身のニーズと関係していますか？それらはどのくらい重要でしょうか？このような質問に対し、客観的に答えるにはどのようにしたらよいのでしょうか？



リスク管理は、リスク カテゴリを捕捉し記述するためのフレームワークが明確に定義されている成熟した統制です。わざわざ一から作り直す必要はありません。特定のリスクがどれくらい重要になるかは、よく知られているリスク カテゴリの相対的な影響度によって定義されます。上に挙げたリスク カテゴリは、一般的にアプリケーションのリバース エンジニアリングや改ざんに関係しています。しかし、これらは普遍的なものではなく、また、完全に網羅されたリストでもありません。

第 4 の考慮点：どれくらいのリスクは手に負えませんか？どれくらいのリスクなら受け入れられますか（リスク許容度）？そして、これらさまざまなカテゴリのリスクを管理（統制）し、それらを所属する組織の「リスク選好度」の範囲内に保つために、どのようなオプションを利用できるでしょうか？

リスクの許容度（または選好度）は技術的な話題ではありません。また、潜在的リスクでもありません。たとえば、4 人の開発者がサイド プロジェクトとして開発した Android アプリがあるとします。これは、ごく一部のクライアントが比較的重要度の低いタスクを実行するためにしか使用されないかもしれません。この開発者は外部のコンサルタントである可能性もあります。そのため、アプリ自体は実質的な知的財産を持たず、収益も生み出さず、顧客ベース（ましてや投資者）ではほとんど見えません。一方、そのアプリの偽造版によって、クライアント データの損失、公的な市場における評判の毀損、および規制の罰則が生じるような場合、実のところ、そのアプリの規模が小さいかどうかは問題ではありません。



「インシデント」発生の可能性と影響の重要度が**許容できない**ほど高いと見なされる場合は、結果として生じるリスクを完全に回避する（この例では、マネージ コードで開発するのを止める）か、あるいは回避が選択肢にない場合は、「統制」を用いることにより、リスクを許容レベルにまで削減する必要があります。

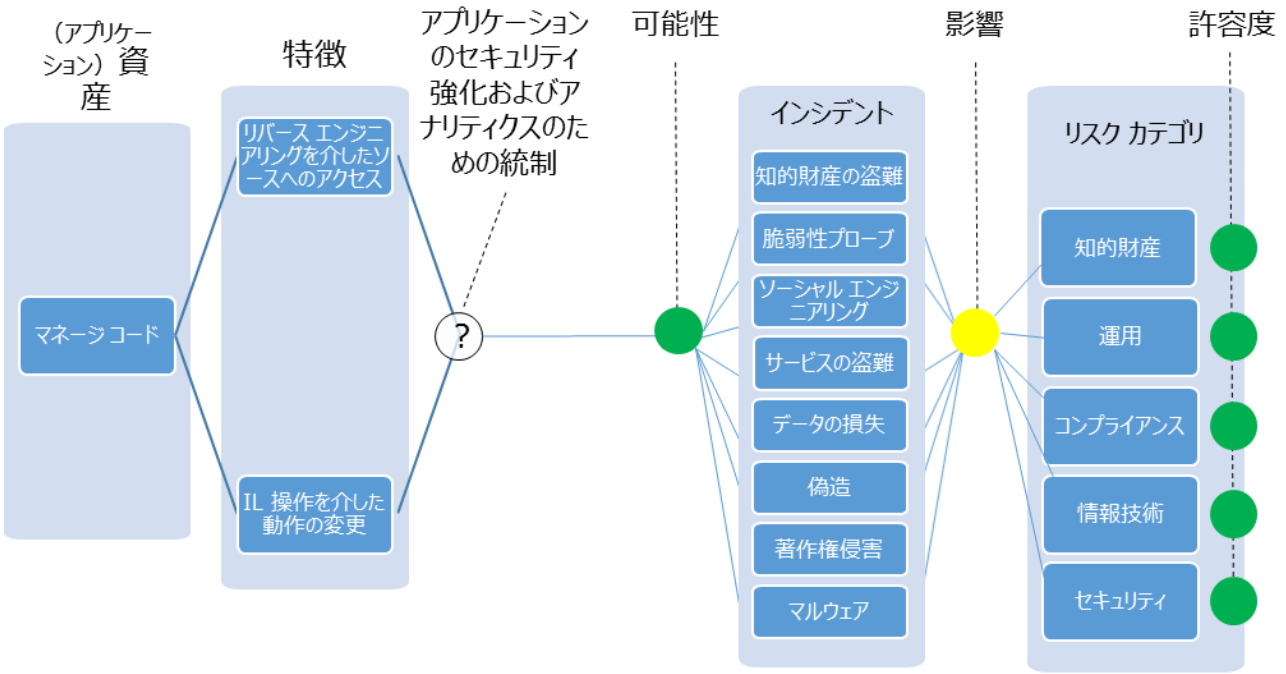
統制が有効であるためには、インシデント発生の可能性と影響を合わせた重要度が、「**許容**」レベルまで軽減される必要があります。

統制によりリスクは排除されません。統制はリスクを許容できるものにします。

つまり、アプリケーションの技術的範囲が狭くても、リスク、ひいては利害関係者が広範囲に及んでいることはよくあります。

リスク管理の決定は、開発者ではなく、リスク管理の専門家によって行われる必要があります（リスク管理者にコード レビューをしてもらいたくはないですよね？）。

第 5 の考慮点：マネージ コードの開発から生じるリスクを管理/統制するために、具体的にどのような統制を利用できますか？



マネージ コードの使用から生じるリスクを軽減する統制には、インシデントが発生する**可能性**を下げる**難読化**（予防的統制）や**改ざんの検出および防御**だけでなく、**アプリケーションの監視とアナリティクス**も含まれます。これらは、より速い検出とリアルタイムでの修復により、インシデント発生による影響を低減します。

統制を有効にするには、アプリケーションを難読化および監視する**技術**、その技術を一貫して使用する方法を詳しく述べる**プロセス**、これらのプロセスを呼び出すタイミングを決定する**ポリシー**を組み合わせる必要があります。このようにして、リスク軽減を**有効**で**一貫性のある**ものとします。

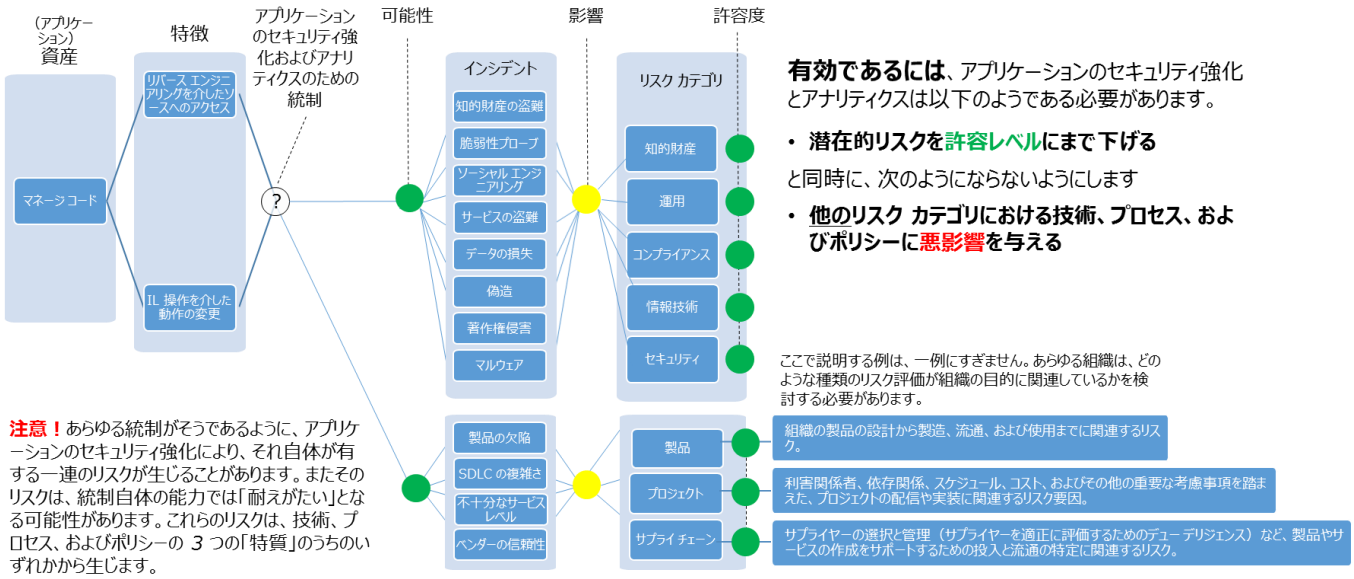
難読化は、いくつの順列にでも適用できる、変換のポートフォリオです。順列はそれぞれ独自の保護的役割を持ち、独自の副作用を伴います。

改ざん検出および防御だけでなく、通常の機能や例外の監視も独自の特色や構成を有しています。

コンピューター攻撃、人為的な攻撃、コンパイル可能なコードを生成することを目的とする攻撃への対抗は、特定の動作は変更するが、それ以外の動作は難読化、改ざん防御、およびアナリティクスのさまざまな組み合わせのためにすべてそのままとする（変更しない）ような設計です。

目標は、特定されたリスク レベルを受け入れ可能（許容）レベルまで引き下げるために必要な、最低限のレベルの保護と監視を適用することです。そのレベルを超えた保護は「過剰」であり、レベルを満たさなければ、努力が無駄になります。このような理由から、第一段階として不可欠なのが、すべての活動をリスクの完全なリストに割り当てることです。

第 6 の考慮点：治療（統制）が病気（潜在的リスク）を悪化させてはなりません。つまり、難読化と改ざん防御のソリューションは、これらの技術が管理することを目的としているリスクよりも破壊的であってはなりません。



難読化、改ざん防御、およびアナリティクスを導入することによって増加するリスクに焦点を当てると、多くの場合、次のような課題の検討が重要になります（これは一部の代表的なものであり、完全なリストではありません）。

* 構成の複雑さ
* 分散した開発チーム全体にわたるシナリオの構築や、ファームの構築などをサポートする柔軟性
* 異なる構成要素全体にわたるデバッグ、パッチのシナリオ、保護スキームの拡張
* マーケットプレース、インストール、およびその他の配布パターン
* さまざまな OS やランタイム フレームワークのサポート
* デジタル署名、ランタイム IL 標準準拠、および透かしのワークフロー
* モバイル パッケージ（またはその他デバイス固有の要件）
* アナリティクスの場合は、プライバシー、接続性、帯域幅、パフォーマンスなどをめぐる追加の問題がある
* 市販の製品の場合は、ベンダーの存続性（3 年後にも存在しているか）、サポート レベル（訓練を受けた専門チームであるか？応答時間は？）

では、どんな場合にアプリケーションを保護すべきでしょうか？

これは、組織において、許容できない高いリスク（財政、運営、コンプライアンスなど）を明確に定義しており、かつ、許容できないリスクや費用を引き受けることなく、推奨されるリスク管理コントロール（技術＋プロセス＋ポリシー）によって、リスク レベルを受け入れられる限度まで引き下げられる場合のみです。