Application Protection
Why bother?

(...and, no, this is not a rhetorical question)

**Why** should a developer (or parent organization) bother to protect their applications? Given the fact that PreEmptive Solutions builds application security and risk management software, you might think that we are somehow being snarky and rhetorical – but, please be assured, we are not.

The only way to answer such a question is to first know what it is that you need protection from. If the answer is something like “to protect against reverse engineering or tampering,” that is not a productive answer. The answer needs to consider what damage is likely to follow if/when reverse engineering or tampering should occur. Are you looking to protect sensitive data? Prevent piracy? Secure Intellectual Property (IP)? AGAIN – not good enough – the real answer is going to have to be tied to lost revenue, regulatory penalties, operational disruption resulting financial or other damage, etc. Unless and until you can answer these questions – it is impossible to appropriately prioritize your response to these risks.

If you feel this is too pedantic or too academic, then (and forgive me for saying this) you are not the person who should be making these kinds of decisions. If, on the other hand, you’re not sure how to answer these kinds of questions – but you understand (even if only in an intuitive way) the distinction between managing risks (damage) versus preventing events that can increase risk – then I hope the following distillation of how to approach managing the unique risks that stem from developing in .NET and/or Java (managed code) will be of value.
**First consideration:** managed code is easy to reverse engineer, monitor and modify by design – and there are plenty of legitimate scenarios where this is a good thing.

Your senior management needs to understand that reverse engineering and executable monitoring and manipulation is well understood and widely practiced.

Therefore, if this common practice poses any material risks to your organization, your organization is compelled to take steps to mitigate those risks – of course, if this basic characteristic of managed code does not pose a material risk – no additional steps are needed (nor should they be recommended).

**Second consideration:** reverse engineering, debugging and monitoring tools don’t commit crimes – criminals do; but criminals have found many ways to commit crimes with these powerful development utilities.

In order to be able to recommend an appropriate strategy, a complete list of threats is required – simply knowing that IP theft is ONE threat is not sufficient – if the loss of sensitive data poses an incremental threat – this qualitatively distinct risk must also be captured.
**Third consideration:** Which of the incident types above are relevant to your specific needs? How important are they? How can you objectively answer these kinds of questions?

Risk management is a mature discipline with well-defined frameworks for capturing and describing risk categories; DO NOT REINVENT THE WHEEL. How significant (material) a given risk may be is defined entirely by the relative impact on well-understood risk categories. The ones listed above are commonly associated with application tampering, monitoring and reverse engineering - but these are not universal nor is the list exhaustive.
Fourth consideration: How much risk is too much? How much risk is acceptable (what is your tolerance for risk)? …and what options are available to manage (control) these various categories of risk to keep them within your organization’s “appetite for risk?”

Tolerance (or appetite) for risk is NOT a technical topic – nor are the underlying risks. For example, an Android app developed by 4 developers as a side project may only be used by a small percentage of your clients to do relatively inconsequential tasks – the developers may even be external consultants – so the app itself has no real IP, generates no revenue, and is hardly visible to your customer base (let alone to your investors). On the other hand, if the result of a counterfeit version of that app results in client loss of data, reputation damage in public markets, and regulatory penalties – the trivial nature of that Android really won’t have mattered.

In other words, even if the technical scope of an application may be narrow, the risk – and therefore the stakeholders – can often be far reaching.

Risk management decisions must be made by risk management professionals – not developers (you wouldn't want risk managers doing code reviews would you?).

When the weight of the likelihood and impact of an “incident” occurring is deemed to be intolerably high, then the resulting risk must either be avoided entirely (stop developing in managed code in this example) or, when avoidance is not an option, the risk must be reduced to a tolerable level through the use of a “control.”

To be effective, a control must reduce the combined weight of likelihood and impact of an incident occurrence to a “tolerable” level.

Controls do not eliminate risks – controls make risks tolerable.
**Fifth consideration:** what controls are available specifically to help manage/control the risks that stem from managed code development?

Controls to mitigate risks stemming from the use of managed code include obfuscation to lower the likelihood of an incident occurrence (a preventative control) and tamper detection and defense as well as application monitoring and analytics to reduce the impact should an incident occur (through faster detection and real-time remediation).

To be effective, the control must combine technology to obfuscate and/or monitor applications, processes that detail how to consistently use the technology, and policies that dictate when to invoke these processes – thus ensuring consistent and effective risk mitigation.

Obfuscation is a portfolio of transformations that can be applied in any number of permutations – each with its own protective role and its own side effects.

Tamper detection and defense as well as regular feature and exception monitoring also have their own flavors and configurations.

Machine attacks, human attacks, attacks whose goal is to generate compliable code versus those designed to modify specific behaviors while leaving others in tact all call for different combinations of obfuscation, rooted device defense (for mobile), tamper defense, and alerts.

The goal is to apply the minimum levels of protection and monitoring required to bring identified risks levels down to an acceptable (tolerable) level. Any protection beyond that level is “over kill.” Anything less is wasted effort. …and this is why mapping all activity to a complete list of risks is an essential first step.
Sixth consideration: the cure (control) cannot be worse than the disease (the underlying risk).
In other words, the obfuscation, anti-debugger, anti-emulator, anti-root and tamper defense solutions cannot be more disruptive than the risks these technologies are designed to manage.

Focusing on the incremental risks that introducing obfuscation, anti-debugger, tamper defense, and other real-time detection and response controls can introduce is an essential part of the process. The following questions are often important to consider (this is a representative subset – not a complete list):

* Complexity of configuration
* Flexibility to support build scenarios across distributed development teams, build farms, etc.
* Debugging, patch scenarios, extending protection schemes across distinct components
* Marketplace, installation, and other distribution patterns
* Support for different OS and runtime frameworks
* Digital signing, runtime IL standards compliance, and watermarking workflows
* Mobile packaging (or other device specific requirements)
* For commercial products, vendor viability (will they be there for you in 3 years) and support levels (dedicated trained team? response times?)

So when should you protect your applications?

*Only when your organization has well-defined risks that are unacceptably high (financial, operational, compliance …) AND the recommended risk management controls (technology + process + policy) reduce risk levels to acceptable limits WITHOUT introducing unacceptable incremental risks or expenses.*