PreEmptive Solutions

Common Practices to Manage Uncommon Risk

There is no data security without application security
Risks that aren't managed consistently cannot be managed effectively

PreEmptive Solutions

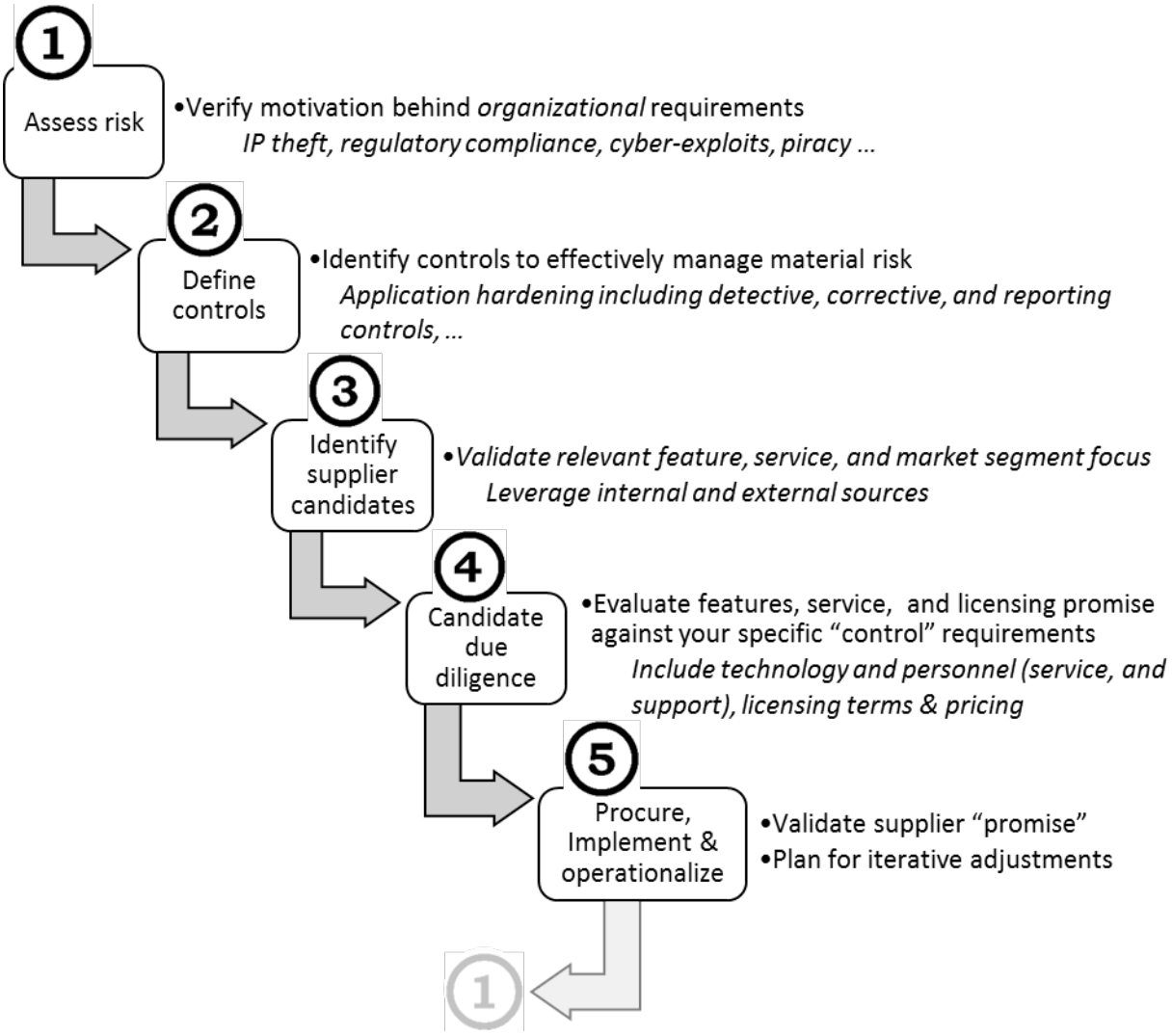
Contents

PreEmptive Solutions

Introduction

Applications saturate every aspect of our professional, cultural, societal and personal lives. Whether public or private, legitimate or criminal - organizations are rapidly learning to manage the risks that come with benefits of our digital age.

This collection of resources is organized around this journey of discovery, effective tactics, and the continuous improvement required to keep pace – not just with technological advances – but with the evolving regulatory, legislative, economic, criminal and geopolitical actors and the opportunities and threats that they represent.



Privacy, safety, and disclosure regulatory obligations often trigger special – and often material – risk factors. This is especially true for organizations that develop those applications. Examples are included here.

The cost, complexity, and risk factors that controls themselves can bring are also critical to understand. Specific considerations and a template for efficient and consistent project planning are featured.

If you would like to see additional reference material produced, please contact us at marketing@preemptive.com.

GDPR Liability: Software Development and The New Law

The GDPR is comprehensive; its impact is far reaching, and the penalties for infringement are severe (*up to €20 million or 4% of global annual revenue, whichever is higher*).

In short, no impacted business can afford to ignore The GDPR. As the May 2018 deadline looms, organizations find themselves scrambling to be "GDPR ready" – *but what exactly does that mean?*

We've simplified the GDPR legalese (while preserving the links to the original regulation) to help answer this question from a development perspective. If there is just one point to take away from this paper, it's that the GDPR is much more than an IT or operational responsibility.

If you're following the GDPR and your organization develops software (directly or through partners – for internal use or external use), this white paper is written for you.

GDPR Roles

The GDPR is organized around the notion of Controllers and Processors and the responsibilities and liabilities they share.

Responsibilities

- A Controller determines the "why" and the "how" of processing personal data.
- A Processor (or processors as the case may be) processes personal data for the Controller

(CHAPTER 1, General provisions, Article 4 Definitions)

Liabilities

The GDPR states that a person who has suffered any kind of damage (material or non-material) from a GDPR **infringement** has the right to compensation.

More to the point, processing systems that do not meet GDPR requirements (and therefore infringe) trigger GDPR liability for every user whose data is processed.

(CHAPTER VIII Remedies, liability and penalties, Article 82.2 Right to compensation and liability). The cost of a single GDPR incident is too high for anyone to ignore. An infringing processing system has the potential to generate thousands – if not millions – of these incidents.

With this potential exposure, do processing system **developers** have *any special obligations*?

Processing System Obligations

The GDPR mandates that processing systems include "appropriate" technical safeguards. For the GDPR, "appropriate" would consider factors like the **state-of-the-art** of hacking techniques and their corresponding countermeasures at

any given time (*implying an ongoing commitment to track and keep pace with developments in this area*), the cost of safeguard implementations (*time, money, other risks*), as well as the relative likelihood and severity of any given class of data breach occurring.

(CHAPTER IV Controller and processor, Section 1 General obligations, Article 25 Data protection by design and by default)

In this sense, the GDPR is consistent with well-understood risk management practices that call for proportionate risk mitigation investments. For a discussion of these basic risk concepts in the context of application development, see The Six Degrees of Application Risk.

The GDPR amplifies these basic concepts and, by implication, expands the working definition of "infringement."

Processing System Infringement

The GDPR places a special importance on "ensuring ongoing confidentiality, integrity, availability and resilience of processing systems and services."

In other words, the *GDPR deliberately carves out obligations for the processing system implementer – not just for the owners and caretakers of the data that flows through those systems*.

The GDPR goes on to state that special care must be taken in both assessing and proactively mitigating processing risks stemming from

- Unlawful destruction, loss, or alteration of personal data, and from
- Unauthorized disclosure of, or access to personal data transmitted, stored or otherwise processed.

(CHAPTER IV Controller and processor, Section 2 Security of personal data, Article 32 Security of processing)

GDPR Processing System Assessment

Extrapolating directly from the GDPR text, we can see that Controllers and Processors are responsible for implementing processing systems that

- Are secure, resilient, and reliable (trusted),
- Include controls to protect against unlawful and/or unauthorized access or disclosure of personal data, AND
- Include "state of the art" (up-to-date) countermeasures against current attack techniques.

The "appropriate technical and organisational measures" standard used throughout the GDPR needs to be extended to ensure that bespoke (custom) software includes the required GDPR safeguards.

GDPR Software Development Assessment

A Controller or Processor that develops components of a processing system must ensure that the code they write does not violate the GDPR obligations list above.

The development organization must be able to demonstrate that it has not – and will not – release software with commonly known, well-understood or otherwise avoidable software gaps or vulnerabilities.

With a notion of what GDPR compliance means for development organizations – how do development organizations get "GDPR ready" efficiently, effectively, and reliably?

For more on this topic, please refer to Application Development and the GDPR: Three Tenets for Effective Compliance.

Application Development and the GDPR: Three Tenets for Effective Compliance

According to the official EU GDPR website, http://www.eugdpr.org, "The EU General Data Protection Regulation (GDPR) is the most important change in data privacy regulation in 20 years."

This may well be true. The GDPR includes unprecedented penalties connected to data breaches, it reaches across international borders, and it targets both data owners and 3rd party service providers that process/manage that data.

While data governance inside IT and DevOps organizations have (justifiably) been the primary focus of GDPR compliance efforts, *application development organizations should also recognize that they have been put on notice as well*.

If your software might, perhaps even at some point in the future, process EU personal data (whether or not your company is the organization running that software) – you and/or your clients will also likely be subject to GDPR obligations and potential penalties.

Organizations that fall into this very wide net should consider the following GDPR tenets:

1. Development organizations can be held accountable for data breaches where attackers capitalized on avoidable software gaps or vulnerabilities.

A personal data breach, as defined by the GDPR, includes data damage, loss, or unauthorized access resulting from application tampering, monitoring, or vulnerability exploit.

The GDPR personal data breach definition includes "the unlawful alteration, loss, unauthorized disclosure of, or access to, personal data transmitted, stored or otherwise processed" (formatting added here for emphasis).

Many data breaches begin with an application vulnerability exploit (elevation of privileges for example) or application tampering (bypassing identity or other security checks using a debugger in a production setting to manipulate app data or runtime logic for example). In both of these examples, an attacker is able to subvert the controls and restrictions that an application would normally impose.

Recommendation: *These risks and their corresponding mitigating controls need to be included in GDPR assessments and, as appropriate, remediation processes. This would apply to both software developed in-house and to supplier risk assessments when software is licensed or used as a service.*

2. 100% vulnerability free applications 100% of the time is an unattainable standard.

Exploiting application vulnerabilities to gain unauthorized control over private data is a widely recognized, common attack technique.

In an ideal world, development would release vulnerability-free applications that were also immune to native and

managed debugger hacks, profilers and reverse-engineering tools. We do not live in an ideal world.

Secure coding practices informed by subsequent static analysis and security testing are often effective in striving for this ideal, but even in the best case scenarios, can never guarantee a vulnerability-free application. Further, secure coding practices do not address risks stemming directly from unauthorized debugging, tampering, or reverse engineering hacks (since these do not rely upon vulnerability exploits for success).

Recommendation: *Controls to prevent vulnerability discovery and exploitation in production settings are necessary compliments to those that minimize the likelihood that vulnerabilities are introduced in the first place.*

3. Application hardening is a recognized control to minimize risks stemming from unauthorized use of debuggers to compromise production applications (and, by extension, the data that flows through them).

In June of 2017, 400 development organizations were asked if they had controls in place to mitigate these kinds of production attacks on their applications.

1. 51% reported having preventative controls
2. 35% reported having detective and defensive controls, and
3. 23% reported having reporting controls.*

* It is also worth noting that the percentages across all categories were higher for development organizations serving manufacturing, financial, and healthcare industries. In short, independent of GDPR requirements, these kinds of controls are widely deployed.

Recommendation: Application hardening can play a vital role in an effective GDPR compliance program and should be evaluated for inclusion within existing application and cybersecurity control frameworks. Further, as the survey responses show, application hardening is generally known to be effective against these kinds of risks and, as such, may be considered by regulators and the courts to be “reasonable” precautions that should - by implication - be in place.

Next Steps

PreEmptive Solutions publishes risk assessment and project implementation templates to help enterprises and System Integrators evaluate and, when appropriate, implement application hardening GDPR controls.

President Obama signs the Defend Trade Secrets Act, May 11, 2016.

Defend Trade Secrets Act codifies "open season" on app reverse engineering

Application hardening and the doctrine of "contributory negligence"

Enjoying unprecedented bipartisan support (Senate 87-0 and the House 410-2), the DTSA bill expands trade secret protection across the US and substantially increases penalties for criminal misconduct – and what could go wrong with that?

After all, according to the Commission on the Theft of American Intellectual Property, **the theft of trade secrets costs the economy more than $300 billion a year**. …and, thanks in large part to technology, trade secrets have never been easier move, to copy, and to steal. In fact, in their 5 year strategic plan, the FBI labeled trade secrets as "one of the country's most vulnerable economic assets" precisely because they are so transportable.

…and nothing in today's world is more mobile than application software

If you were to assume that this bill has been custom-tailored to protect the trade secrets embedded in application software - you would be in good company

In her most recent blog post praising the Defend Trade Secrets Act, Michelle K. Lee, Under Secretary of Commerce for Intellectual Property and the current USPTO Director writes, "No matter the industry, whether telecommunications or biotechnology, traditional or advanced manufacturing or **software**, trade secrets are an essential driver of innovation and need to be afforded proper protections." … "Trade secret owners now also have the same access to federal courts long enjoyed by the holders of other types of IP."

Do software developers really now "enjoy the same access to federal courts?" Sort of – maybe – or maybe not.

Without special care, **Application owners have been stripped of every protection granted under the Defend Trade Secrets Act (DTSA)**.

The DTSA applies exclusively to "valuable" information that is both "secret" and has been "stolen" (the legal term is "acquired through Improper Means").

Alert: The DTSA explicitly excluded reverse engineering as "improper" means. The DTSA states that Improper Means DOES NOT include "reverse engineering, independent derivation, or any other lawful means of acquisition."

Is this an oversight? Did the legal staff of the Senate Judiciary Committee (who authored this bill) accidentally use this overloaded development term?

The answer is an unequivocal no – the exclusion of reverse engineered software is intentional and by design.

In a private briefing on Capitol Hill with senior legal counsel inside the Senate Judiciary Committee (the agenda was

encryption that day – not trade secrets) – we asked this question directly – "Did the committee intentionally include language that would exempt any intellectual property that could be accessed via reverse engineering of applications?" The answer was unequivocal. "Yes." "If I can see your IP with a reverse engineering tool – it's mine."

Is every algorithm, process, and data that flows through your software officially free for the taking?

No – it's not nearly that dire.

First – whether or not your IP is covered under this law – obfuscating .NET, Android, Java, or iOS applications make reverse engineering much harder. Code obfuscation will prevent – or at least reduce the number of times that your IP is lifted through reverse engineering.

Can application obfuscation be used to extend the protections of the DTSA to include application software in a court of law?

"Reasonable Efforts" and "The Doctrine of Contributory Negligence"

How do you ensure employees don't publicize your textual and image-based trade secrets (and exempt these from protection as well)?

You make sure employees know that they are secret through clear markings, communication, and education – and you secure relevant documents with physical and electronic locks. These are called "affirmative steps" that demonstrate concrete efforts to preserve confidentiality.

Failure to take these kinds of reasonable efforts lead to The Doctrine of Contributory Negligence.

This "doctrine" captures conduct that falls below the standard to which one should conform for one's own protection. When you fall below this standard, courts will often treat your information as public – and, to the extent you rise above that standard – courts are typically more willing to accept both the secret nature and the value of the IP in question.

Unfortunately, applications are not documents - and so standard "electronic and physical locks" do not apply.

However, code obfuscation does apply here. Obfuscation *is* a well-understood, widely practiced, and recognized practice to prevent reverse engineering. Code obfuscation does not guarantee absolute secrecy – but it is unquestionably recognized as a "reasonable step" to preserve secrecy – it's a lock on a front door that sends an unmistakable message to anyone who approaches – if I'm obfuscated – keep out.

Will development organizations who fail to include basic code obfuscation fall prey to the ominous sounding "Doctrine of Contributory Negligence?"

Can application obfuscation send a clear enough message to the courts to bring back trade secret theft protection under the newly minted Defend Trade Secrets Act?

The courts are still working through these issues.

In the meantime, be sure to take reasonable precautions to protect your trade secrets – whether they sit inside software or the data that your software processes.

Smart Cars Demand Smart Code

The following excerpts are from The key principles of vehicle cyber security for connected and automated vehicles (https://www.gov.uk/government/publications/principles-of-cyber-security-for-connected-and-automated-vehicles/the-key-principles-of-vehicle-cyber-security-for-connected-and-automated-vehicles) authored by the UK Centre for the Protection of National Infrastructure on August 9, 2017.

The excerpted content has been selected to highlight the following important elements that run throughout the "key principles:"

- There is an obligation to sustain "state of the art" threat awareness and mitigating controls versus the traditional "reasonable" effort,
- The linkage of data and code and hardware into a single risk framework, and
- The shared obligations across the entire supply chain

Guidance

Principle 1 - organisational security is owned, governed and promoted at board level

Principle 1.4
All new designs embrace security by design. Secure design principles are followed in developing a secure ITS/CAV system, and all aspects of security (physical, personnel and cyber) are integrated into the product and service development process.

Principle 2 - security risks are assessed and managed appropriately and proportionately, including those specific to the supply chain

Principle 2.1
Organisations must require knowledge and understanding of current and relevant threats and the engineering practices to mitigate them in their engineering roles.

Principle 2.2
Organisations collaborate and engage with appropriate third parties to enhance threat awareness and appropriate response planning.

Principle 2.3
Security risk assessment and management procedures are in place within the organisation. Appropriate processes for identification, categorisation, prioritisation, and treatment of security risks, including those from cyber, are developed.

Principle 2.4
Security risks specific to, and/or encompassing, supply chains, sub-contractors and service providers are identified and managed through design, specification and procurement practices.

Principle 3 - organisations need product aftercare and incident response to ensure systems are secure over their lifetime

Principle 3.1
Organisations plan for how to maintain security over the lifetime of their systems, including any necessary after-sales support services.

Principle 3.2
Incident response plans are in place. Organisations plan for how to respond to potential compromise of safety critical

assets, non-safety critical assets, and system malfunctions, and how to return affected systems to a safe and secure state.

Principle 3.3
There is an active programme in place to identify critical vulnerabilities and appropriate systems in place to mitigate them in a proportionate manner.

Principle 3.4
Organisations ensure their systems are able to support data forensics and the recovery of forensically robust, uniquely identifiable data. This may be used to identify the cause of any cyber, or other, incident.

Principle 4 - all organisations, including sub-contractors, suppliers and potential 3rd parties, work together to enhance the security of the system

Principle 4.1
Organisations, including suppliers and 3rd parties, must be able to provide assurance, such as independent validation or certification, of their security processes and products (physical, personnel and cyber).

Principle 4.3
Organisations jointly plan for how systems will safely and securely interact with external devices, connections (including the ecosystem), services (including maintenance), operations or control centres. This may include agreeing standards and data requirements.

Principle 4.4
Organisations identify and manage external dependencies. Where the accuracy or availability of sensor or external data is critical to automated functions, secondary measures must also be employed.

Principle 5 - systems are designed using a defence-in-depth approach

Principle 5.1
The security of the system does not rely on single points of failure, security by obscuration or anything which cannot be readily changed, should it be compromised.

Principle 5.2
The security architecture applies defence-in-depth and segmented techniques, seeking to mitigate risks with complementary controls such as monitoring, alerting, segregation, reducing attack surfaces (such as open internet ports), trust layers / boundaries and other security protocols.

Principle 6 - the security of all software is managed throughout its lifetime

Principle 6.1
Organisations adopt secure coding practices to proportionately manage risks from known and unknown vulnerabilities in software, including existing code libraries. Systems to manage, audit and test code are in place.

Principle 8 - the system is designed to be resilient to attacks and respond appropriately when its defences or sensors fail

Principle 8.1
The system must be able to withstand receiving corrupt, invalid or malicious data or commands via its external and internal interfaces while remaining available for primary use. This includes sensor jamming or spoofing.

Principle 8.2
Systems are resilient and fail-safe if safety-critical functions are compromised or cease to work. The mechanism is proportionate to the risk. The systems are able to respond appropriately if non-safety critical functions fail.

Application Risk Management Survey Summary Report

In June of 2017, 397 developers completed an Application Risk Management survey. The respondents represented 55+ industries and 100+ countries. The survey gathered information about their development organizations' risk management priorities and mitigation strategies.

How to Use This Report

Development organizations can use the survey results to benchmark their own practices by industry, application type, development organization size, etc.

The summary information presented here identifies application, organizational, and industry-specific considerations that can – and should – be considered by every development organization serious about sustaining an effective application risk management program.

For organizations interested in comparing their own specific practices against the full data set, please contact AppRiskMgmt@preemptive.com. You will be provided with a link to an online questionnaire and a benchmark analysis will be delivered to your attention following the completion of the questionnaire.

Key Survey Results

Risk Profiles

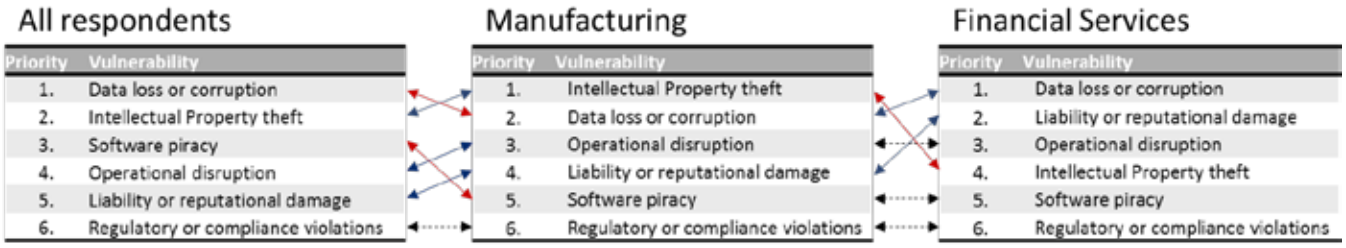
1. PROFESSIONAL APPLICATION DEVELOPMENT ORGANIZATIONS GENERALLY AGREE ON THE "SHORT LIST" OF COMMON APPLICATION DEVELOPMENT VULNERABILITIES.

   The top 6 application development vulnerabilities

Application Development Vulnerabilities
Data loss or corruption
Intellectual Property theft
Liability or reputational damage
Operational disruption
Regulatory or compliance violations
Software piracy

2. ...BUT DIVERGE QUICKLY ON THEIR RELEVANCE, PRIORITIZATION, AND MITIGATION STRATEGIES.

   The top 6 application development vulnerabilities prioritized by materiality (severity & relevance)



Vulnerability priority ranking compared across all developers, manufacturing, and financial services

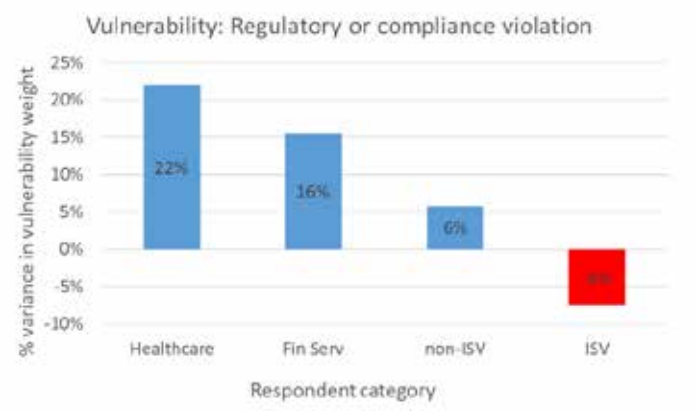
Manufactures rank IP theft as a greater threat than the general development community did and significantly higher than financial service company developers.

Financial Service companies placed data loss or corruption as the highest priority vulnerability with liability or reputational damage as the second highest priority vulnerability to manage.

3. THE MATERIALITY OF EACH VULNERABILITY ALSO VARIED WIDELY ACROSS DEVELOPMENT SEGMENTS.

   Development organizations may align on the relative prioritization of vulnerabilities within a group, but they can still place markedly different weights on any individual vulnerability.

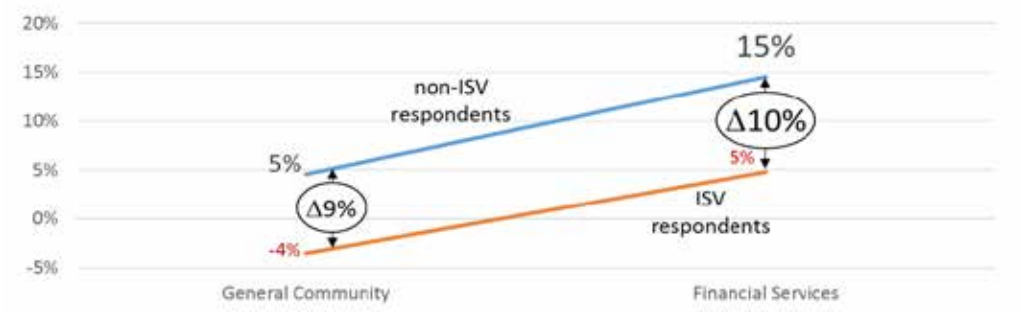
   As an example, regulatory or compliance violations were typically placed at the bottom of the prioritized vulnerability lists. However, the relative importance of regulatory compliance across different development communities varied significantly.



Healthcare and financial services development teams placed a greater weight on regulatory and compliance risk versus the general development community.

ISVs (independent Software Vendors) placed a lower emphasis on regulator risk than did the general non-ISV development community.

4. ISVs SHARE SOME UNIQUE TRAITS AMONGST THEMSELVES, BUT THEY ARE OFTEN MOST INFLUENCED BY THE CUSTOMERS THEY SERVE.



Comparing ISV and non-ISV Risk Tolerance (Appetite)

Non-ISV's have a 9% higher investment in mitigating risks than their ISV counterparts (which is 5% higher than the overall average investment).

Financial Services development organizations have a 10% higher investment in mitigating risks than the general non-ISV community.

ISV's whose applications are specifically developed for the Financial Services industry, like the financial industry developers themselves, have a ~10% higher than the general ISV community.

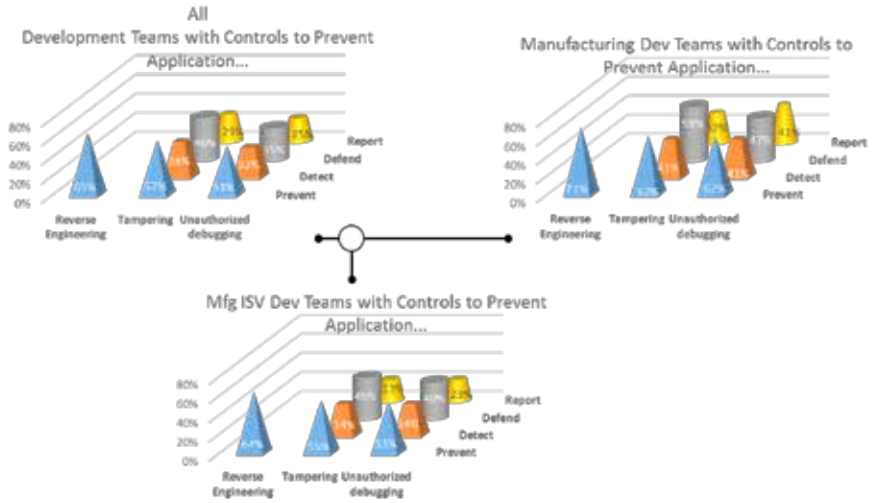
Financial Services ISV's have assumed the risk mitigation profile of their target users' risk profile.

Risk Mitigation Controls and Technologies

Survey respondents also indicated to what extent their development organizations have implemented specific controls to mitigate their application development risk.

Three gaps were assessed; Reverse engineering, tampering, and unauthorized debugging (to view and modify data, logic, and privileges).

Four classes of controls were measured; preventative, detective, active defense, and reporting.



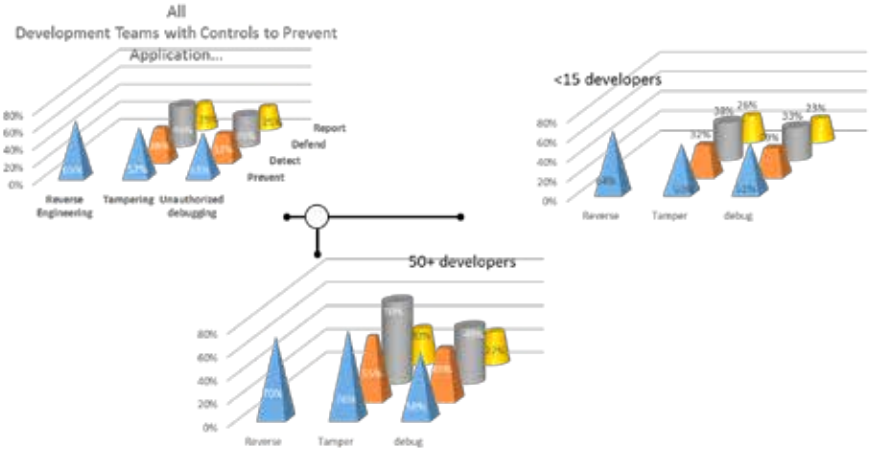
Application control adoption comparison between all respondents, manufacturers, and ISVs developing software for manufacturers.

The majority of development organization have controls to prevent reverse engineering, application tampering, and unauthorized debugging. Preventative controls are a well-understood, common practice.

Manufacturers are most likely to implement application risk management controls across all classes and, in addition to preventative controls, defending against application tampering is also a common, well-understood practice.

ISVs developing software for manufacturers, as a group, appear to be out of step with the priorities of their target industry. As supplier risk management grows in importance, many of these ISVs may find themselves squeezed out but more security conscious competitors.

Development organization size major predictor of control investments



Development organizations greater than 50 are roughly more than 2X more likely to be investing in detective and defensive controls than those organizations with less than 15 developers.

Have questions about this research? Want to learn more about your peers and how you measure-up? Contact Sales@preemptive.com and we will be delighted to work with you.

Demographics

Some more information about the 397 respondents.

Applications being developed (multiple selections permitted)

For internal business use	47.86%
To support our partners and supply chain	22.42%
For sale and/or subscription use	53.65%
Embedded inside or in support of some form of equipment	15.11%

Development organization size

Answer Choices	Responses
1-5 developers	59.32%
6-15 developers	14.70%
16-50 developers	13.91%
51+ developers	12.07%

Development platform and language (multiple selections permitted)

.NET desktop	81.11%
.NET backend server components	53.90%
Modern .NET surfaces including UWP, .NET Core	29.97%
Xamarin for Android	15.87%
Xamarin for iOS	14.11%
Other Xamarin	7.05%
Cloud-based apps and services	21.16%
Non-mobile Java	7.30%
Android	22.42%
iOS objective C	10.33%
JavaScript	28.46%
Other (please specify) Responses	6.80%

Risk management maturity (risk priorities and controls are established…)

In an ad hoc and occasionally reactive manner	40.86%
Within a framework approved by management	30.35%
Within a framework formally established as organizational policy	16.34%
Within a framework formally established as organizational policy that is regularly updated based on the application of risk management processes to changes in business/mission requirements and a changing threat and technology landscape.	12.45%

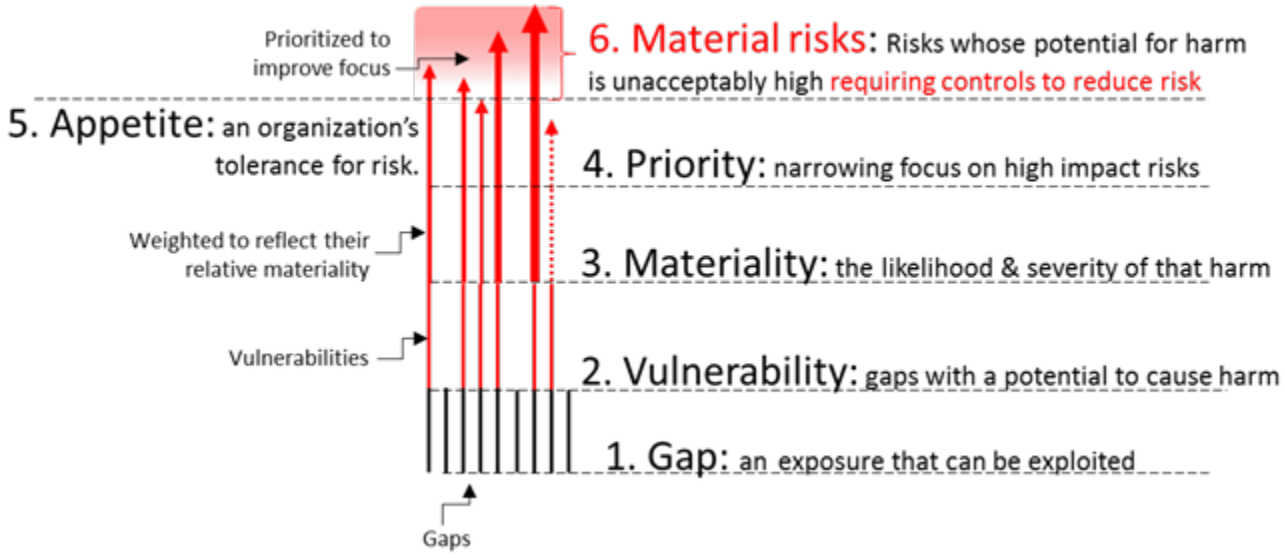
The Six Degrees of Application Risk

Cyber-attacks, evolving privacy and intellectual property legislation, and ever-increasing regulatory obligations are now simply "the new normal" – and the implications for development organizations are unavoidable; application risk management principles must be incorporated into every phase of the development lifecycle.

Organizations want to work smart – not be naïve – or paranoid. Application risk management is about getting this balance right. How much security is enough? Are you even protecting the right things?

The six degrees of application risk offer a basic framework to engage application stakeholders in a productive dialogue – whether they are risk or security professionals, developers, management, or even end users.

With these concepts, organizations will be in a strong position to take advantage of the following risk management hacks (an unfortunate turn of a phrase perhaps) that reduce the cost, effort, complexity, and time required to get your development on the right track.



Six Degrees of Application Risk

The following commonly used (and related) terms provide a minimal framework to communicate application risk concepts and priorities.

1. **Gaps** are (mostly) well-understood behaviors and characteristics of an application, its runtime environment, and/or the people that interact with the application. As an example, .NET and Java applications (managed applications) are especially easy to reverse-engineer. This isn't an oversight or an accident that will be corrected in the "next release." Managed code, by design, includes significantly more information at runtime than its C++ or other native language counterparts – making it easier to reverse-engineer.
2. **Vulnerabilities** are the subset of Gaps that, if exploited, can result in some sort of damage or harm. If, for example, an application was published as an open source project – one would not expect that reverse engineering an instance of that application would do any harm. After all, as an open source project, the source code would be published alongside the executable. In this case, the Gap (reverse engineering) would NOT qualify as a Vulnerability.
3. **Materiality** is the subjective (but not arbitrary) assessment of how likely a vulnerability will be exploited combined with the severity of that exploitation. The likelihood of a climate-changing impact of a meteor hitting earth in the next 3 years is significantly lower than the likelihood of an electrical fire in your home. This distinction outweighs the fact that a meteor impact will obviously do far more harm than a single home fire. This is why we, as

individuals, invest time and money preventing, detecting, and impeding electrical fires while taking no preemptive steps to mitigate the risks of a meteor collision.

4. **Priority** ranking of vulnerabilities helps to ensure that our limited resources are most effectively allocated. Vulnerabilities are not all created equal and, therefore, do not justify the same degree of risk mitigation investment. Life insurance is important – but medical insurance typically is seen as "more material" justifying greater investments.
5. **Appetite** for risk is another a subjective (but not arbitrary) measure. Appetite is synonymous with tolerance. Organizations cannot eliminate risk – but each organization must identify those vulnerabilities whose combined likelihood and impact are simply unacceptable. Some sort of action is required to reduce (not eliminate) those risks to bring them to within tolerable levels. Health insurance does not reduce the likelihood of a health-related incident – it reduces some of the harm that stems from an incident when it occurs. While many individuals have both life and health insurance, there are many who feel that they can tolerate living without life insurance but cannot tolerate losing health insurance.
6. **Material risks** are those vulnerabilities whose risk profile are intolerably high. Material risks are, by definition, any vulnerability that merits some level of investment to bring either its likelihood and/or its impact down to within tolerable levels. Ideally, once all risk management controls are in place, there are no "intolerable risks" looming.

Applying the Six Degrees of Application Risk

Extending these concepts into the development process, at a high level, translate into the following activities:

- Inventory relevant "gaps" across your development and production environments
- Identify the vulnerabilities within the collection of gaps
- Assess and prioritize according to your organization's notions of materiality
- Agree on a consistent definition of your organizations tolerance for these vulnerabilities (appetite)
- Identify the vulnerabilities that present a material risk
- Select and implement controls to mitigate these risks
- Measure, assess, and correct on an ongoing (periodic) basis

Simple right?

Effective Application Risk Management Hacks

Incorporating any new process or technology into a mature development process is, in and of itself, a risky and potentially expensive proposition.

> *The threat of increasing development complexity or cost, or compromising application quality or user experience is often motivation enough to maintain the status quo.*

Avoid unnecessary waste and risk – follow-the-leaders

There is an old saying in risk management that "you don't have to be the fastest running from the bear – you just don't want to be the slowest." Hackers mostly attack targets of opportunity and regulators and the courts typically look for "reasonable" and "appropriate" controls.[1] It is often much more efficient to benchmark and adapt the practices of your peers rather than develop your own risk management and security practices from the ground-up. There are many sources from which to choose.

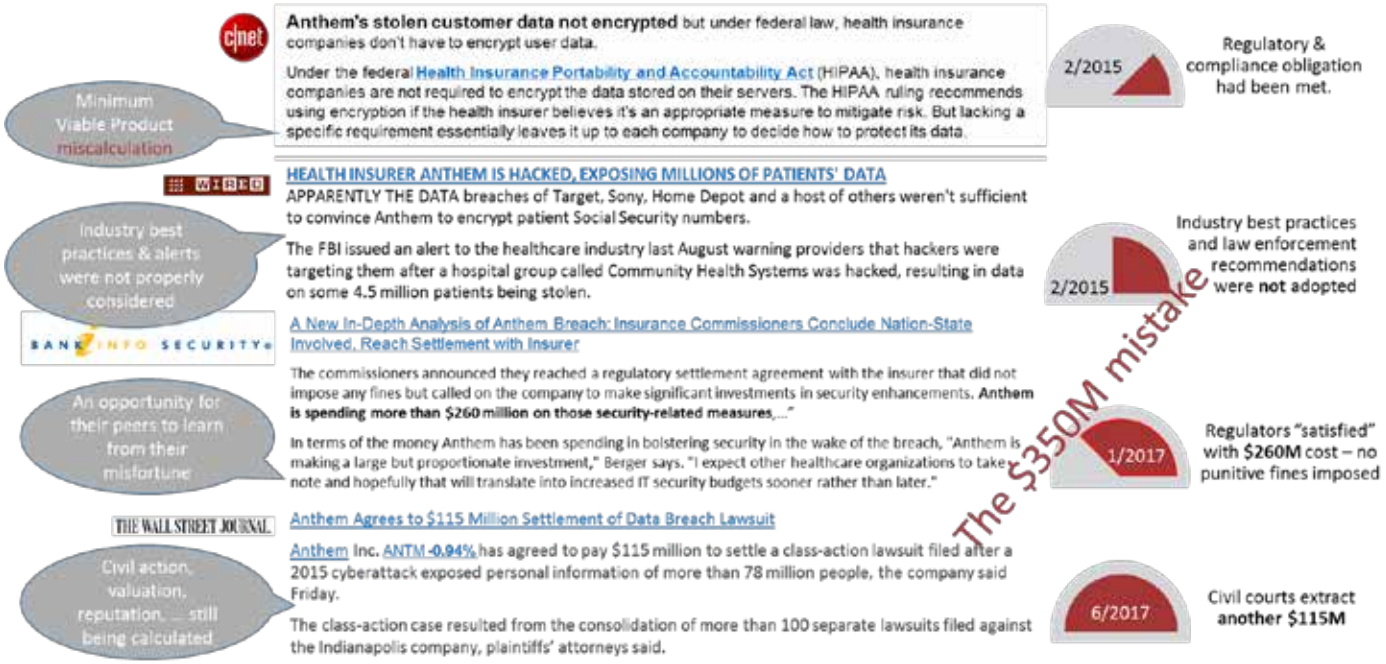
- Benchmark your practices against your organization's
  - peers (similar organizations)

[1] Note the sudden rise of the "state-of-the-art" standard in lieu of the usual "reasonable" standard.

  - customers (their risks are often, by extension, your risks)
  - suppliers (they are experts in their specialty and/or may pose a risk if they do not live up to your appetite for risk)
- Embrace well-understood and common practices
  - Adopt an accepted a standard or open risk management framework.
  - Monitor regulatory and legislative developments
  - Track relevant breaches and exploits and the aftermath

Punishing the victim: the Anthem data breach

The following timeline of the Anthem data breach illustrates clearly the risks and penalties that come with under-managing application and data management risk. While complying with relevant standards, Anthem's failure to stay current with cyber threats cost them over $350M USD.



PreEmptive Solutions Application Risk Management

Background & Guiding Principles

PreEmptive's product portfolio, roadmap, and overall mission is best understood in the context of the broader discipline of risk management in general.

To pursue application development opportunities, organizations must concurrently address application threats and requirements through a mix of preventative, detective, responsive, and reporting controls.

Effective controls must consistently and efficiently mitigate and manage material risks such that the residual risk, after controls have been applied, fall within acceptable limits (appetite for risk).

For over 20 years, Preemptive Solutions has delivered market-leading software to manage risks stemming from application reverse engineering, unauthorized monitoring, tampering, and other relevant and emerging categories of application abuse.

Effective Application Risk Management

Consistency and efficiency requires a sustained investment in the following categories. On behalf of our users, PreEmptive Solutions continuously invests in the following areas:

Effective feature set aligned with control categories

Effective risk management supports all four control "dimensions."

Controls			
Preventative	**Detective**	**Responsive**	**Reporting**
Obfuscation • Renaming (patented) • Control flow (configurable) • String encryption • Linking & pruning • Metadata stripping…	Anti-tamper Anti-debugging Expiry Rooted (Android)	Real-time defenses Pre-packaged exceptions, Randomized crashes Custom method calls	Packet transmission to arbitrary destinations and/or existing portals, e.g. App Insights, Google Analytics. Alerts include offline caching, encryption, etc.

Quality

As "the last step" before digital signing and distribution, quality issues that may arise have the potential to have catastrophic impact on deployment and production application service levels.

Testing	Validation
Rigorous automated and manual testing process refined over 15 years supporting .NET, Java, Android, iOS, …	With over 500,000 Dotfuscator Community Edition users, every release is immediately hardened against millions of scenarios.
As an embedded component inside Visual Studio (since VS2003), Dotfuscator is additionally subjected to Microsoft's regression, security, and code review processes	With over 5,000 commercial enterprise clients, Dotfuscator Professional's scale, manufacturing integration, and enterprise protection features are immediately validated against the world's most demanding development organizations.

Timeliness

Three factors drive release cycles for PreEmptive Solutions application protection and risk management products; *the latter two are unique to the larger security and risk management category*.

1. **New product features and accrued bug fixes:** this is typically the sole driving force for new software product releases.
2. **Updates to OS, runtime, and specialized runtime frameworks:** delayed support for new formats and semantics would result in delays in developer support for those platforms or will force poor risk management practices on the platforms that most likely need protection most of all.
3. **Emergence of new threats and malicious patterns and practices:** as with anti-virus software, bad actors are constantly searching for ways to circumvent security controls. Without consistent tracking of this activity and timely updates to react to these developments, application security technology can quickly be rendered as obsolete.

As a case in point, in the past 24 months, PreEmptive Solutions has released 28 product updates in the past 24 months.

Low Friction

In order to be effective and consistently applied, the configuration and implementation of proactive, detective, and corrective controls cannot require excessive time or expertise. Specific areas where PreEmptive Solutions invests to reduce development and operational friction include:

- Automated detection and protection of common programming frameworks, e.g. WPF, Universal Applications, Spring, etc.
- Custom rule definition language to maximize protection across complex programming patterns at scale
- Specialized utilities to simplify debugging of hardening applications.
- Automated deployment: support for build farms, dynamically constructed virtual machines, command line integration, MSBuild, Ant, etc. come standard with PreEmptive Solutions' professional SKUs.
- Cross-assembly hardening to extend protection strategies across distributed components and for components built in different locations and at different times.
- Support for patch and incremental hardening to minimize and simplify updates to hardened application components.

Responsive Support

Should critical issues arise, live support can prove to be the difference between applications shipping on time or suffering last-minute and unplanned delays. PreEmptive Solutions has dedicated, live support staff available to all of our clients.

Vendor Viability

Applications can live in production for years – and with extended application lifecycles comes the requirement to secure these applications across evolving threat patterns, runtime environments, and compliance obligations. PreEmptive Solutions created the obfuscation category and has been the leading application hardening solution provider for over 15 years. Our clients include government agencies, financial institutions, leading manufacturers, healthcare and medical device manufacturers, aerospace, and, in fact, every other mission critical industry segment in today's modern economy.

PreEmptive Solutions

Application Hardening Implementation Project Plan

Contents

Before you begin

This document addresses scenarios across a broad set of platforms (.NET, Xamarin, Unity, Java, Kotlin, Android) and references corresponding PreEmptive Solutions technologies and products (Dotfuscator, DashO, and PreEmptive Protection for iOS) [1]. There are also a growing number of specialized, platform-specific editions of this document, e.g. Xamarin only. These editions are shorter and simpler but with a limited scope. For information on these platform-specific versions, contact sales@preemptive.com.

How to use this document

This document is a template for a high-level project plan to harden one or more applications using PreEmptive Protection products. This template helps the project planner to:

1. *Understand PreEmptive Protection features and requirements*. By reading through this template, you'll get an overview of how PreEmptive Protection works and how it fits into your development processes. For a more thorough treatment of any particular product, please refer to the corresponding product documentation, e.g. Dotfuscator or DashO.

[1] While also appropriate in many respects for *PreEmptive Protection for iOS* (PPiOS), specific notes for iOS protection will be included in the future.

2. *Plan and schedule the initial implementation*. By answering a short series of questions and completing the forms in this template, you and your organization will be well-prepared to implement an effective application hardening process in a timely, predictable and efficient manner.

3. *Execute*. A completed implementation plan provides the structure and information to deliver a scalable, sustained, and automated application hardening process properly calibrated to your specific application risk management priorities and requirements.

Using this template is not required.

For those situations where you simply need to harden an application as quickly as possible, the default configuration settings for each of our products are designed to be appropriate for most scenarios and most organizations. The integration process is well-documented and supported by live, dedicated support professionals.

There are, however, many important reasons to adopt a methodical approach to hardening your applications.

1. There are often cost/benefit tradeoffs to be made between the level of protection offered by a particular control and the cost of implementing that control (e.g. complexity of implementation, potential for performance impact, challenges related to debugging, etc.). The defaults provide reasonably-strong protection with a minimum potential for side-effects.

   This may be especially important for external regulations or other obligations that explicitly require specific controls, documentation, breach notification protocols, etc.

2. Controls that trigger new application behavior require additional planning and communication. Some of the available controls detect, respond, and report on production incidents, and must be configured to do so in a consistent and effective way. Configuring these controls – which are disabled by default – requires decision making from product owners and outside stakeholders as they affect application behavior and organizational incident response. New behaviors often need to be communicated to other teams, e.g. product support.

   A documented process is especially valuable when application hardening is part of a larger risk or compliance management program, where documentation, audit capability, and communication are also essential.

3. Technology platforms, development environments, and PreEmptive's products each have their own idiosyncratic properties. As such, the effort required to integrate and configure PreEmptive Protection can vary. Planning the implementation process will minimize otherwise avoidable implementation missteps.

4. A consistent implementation methodology helps to ensure that risks are managed consistently. Organizations establish, at the highest levels, their tolerance for risk (a risk appetite) and that tolerance is expressed through a framework, policies, and recognized controls.

Using this document will help you deliver a successful integration project and a working application, while providing an appropriately balanced level of protection, without avoidable delays or issues.

To get started:

1. Ensure you (and any other implementation personnel) have access to the PreEmptive Protection software, so you can start to get familiar with it.
2. Work with your implementation team to go through the document, answering each question and filling in all the blanks. **The bold sentences** represent places where you should update the value; feel free to use them as-is or write whatever is appropriate.
3. Schedule and execute the plan, as described in the finished document.

And as always, please contact PreEmptive Solutions at support@preemptive.com if you have issues or need assistance!

When you are ready to begin, you can delete this section from the document.

Introduction

This document is an application hardening project plan that integrates Dotfuscator or DashO into an application's development process and build pipeline.

PreEmptive Solutions' products "harden" applications to prevent reverse engineering, tampering, and unauthorized monitoring, and to help secure the data that passes through those applications.

Depending on the scenario, application hardening is used for Intellectual Property protection and a variety of application controls including anti-piracy, anti-counterfeiting, data privacy, operational resilience, trade secret protection, and more. These controls, in turn, often play a significant role in regulatory compliance obligations including PCI, GDPR, and HIPAA and in meeting statutory requirements such as the DTSA (Defend Trade Secret Act).

Each PreEmptive Protection product has its own unique user interface and features, but there are many common components and their basic usage model is consistent. They each:

- …operate on compiled binaries (executables, DLLs, Jar files, APK files, etc.).
- …use a configuration file to specify what they should do to each binary.
- …can also be configured via the use of in-code attributes or annotations.
- …have a component that integrates into the application build process, so that protection can be integrated automatically during each build.
- …have a user interface that can be used to generate and edit the configuration file.
- …can transform the binaries in multiple different ways:
    - Renaming obfuscation
    - Control Flow obfuscation
    - String Encryption obfuscation
    - Remove unused code
    - Link (or merge) binaries together
    - Watermark binaries
    - Inject code that performs state "checks" at runtime for e.g. debugging, tampering, product expiration, etc.
    - Inject code that responds when the state checks are triggered, with built-in actions and/or by calling custom application code

This document provides an outline for an initial application hardening implementation for a previously unprotected application. It includes:

- personnel assignments and roles
- a risk assessment
- configuration details about each of the aspects of PreEmptive Protection
- an implementation plan
- an estimated schedule for the initial implementation

Personnel

This document assumes personnel are available who can perform the roles listed below. It is not necessary for each role to have a distinct person; multiple roles could be performed by the same person. Alternately, a single role might be spread across a group of people.

The following roles are assumed:

- **Product Owner**: someone with authority to sign off on product functionality, and to give input into a product-based risk analysis. The may also communicate product functionality changes to secondary stakeholders. They will typically spend 2-8 hours on the project.
    - **For this project, the Product Owner will be:**
- **Project Manager**: someone who plans the project schedule, allocates people and work, and tracks progress. They will typically spend 4-8 hours on the project.
    - **For this project, the Project Manager will be:**
- **Developer(s)**: someone who can modify the source code of the application. A developer is not necessarily required to use PreEmptive Protection; see below for details. If a developer is required, they will typically spend anywhere from a day to a week or more on the project, implementing custom functionality.
    - **For this project, the Developer(s) will be:**
- **Build Expert(s)**: someone who can modify the build process of the application. They will typically spend anywhere from a day to a week or more on the project, depending on the application complexity and aggressiveness of the protection configuration.
    - **For this project, the Build Expert(s) will be:**
- **Tester(s)**: someone who can test the application and verify its correct behavior. They will typically spend anywhere from a day to a week or more on the project, depending on the testing difficulty of the application and the aggressiveness of the protection configuration.
    - **For this project, the Tester(s) will be:**

Application Hardening Risk Assessment

PreEmptive Solutions' application hardening options are highly configurable. It is not always clear how aggressively hardening controls should be applied. Consider the following:

- There are often cost/benefit tradeoffs to be made between the level of protection offered by a particular control and the cost of implementing that control (e.g. complexity of implementation, potential for performance impact, challenges related to debugging, etc.).
- Controls that trigger new application behavior require additional planning and communication. Some controls detect, respond, and report on production incidents, and must be configured to do so in a consistent and effective way. Configuring these controls – which are disabled by default – requires decision making from product owners and outside stakeholders, as they affect

application behavior and organizational incident response management. There may also be a requirement to communicate these new behaviors to other teams, e.g. product support.

- Technology platforms, development environments, and PreEmptive's products each have their own idiosyncratic properties. As such, the effort required to integrate and configure PreEmptive Solutions products vary.

To make appropriate implementation decisions, an organization must:

1. Reach a consensus on the material threats relevant to the application and the organization.
2. The likelihood and cost of an occurrence should one occur.
3. The organization's tolerance (appetite) for these risks.
4. Which controls to employ to mitigate those risks – based in large part on control effectiveness and the cost and risks inherent in the implementation of a control.

To develop this understanding, perform a risk assessment that includes representation from the relevant application stakeholder personas. Even a brief meeting of the appropriate parties, just to discuss these topics, can go a long way to help the implementation team understand how to prioritize and invest.

This document will not cover all the details of how to do a formal risk assessment. For more discussion on application risk assessment, see The Six Degrees of Application Risk.

As a quick start, consider these ideas:

- Ensure you have the right people available for the assessment. Risk assessments extend well into the business and legal domain, and people from outside departments may have important input to provide.
- Consider many risk categories, including: intellectual property theft, piracy, data theft, unauthorized access, operational disruption, fraud, privacy breach, compliance violations, reputational damage, device compromise, vulnerability discovery, and so on.
- Your organization may already have policies and controls that apply to this application; be sure to involve stakeholders who would know of them and can decide whether they are relevant and appropriate.
- When thinking about risk mitigation, brainstorm beyond just technical solutions – consider procedural changes, legal approaches, training, etc.
- When thinking about technical mitigations, consider passive controls such as obfuscation and active controls such as detective capabilities, defensive measures, and/or reporting of telemetry.

The output of the risk assessment should be a shared understanding of the topics above. A summary of that understanding should be documented here.

Implementation Decisions

Before beginning an implementation, the project team should have already answered the following three high level questions:

1. Will in-code attributes/annotations be used to configure PreEmptive Protection?
2. Will runtime state checks be injected into the application, and if so, how will the application respond to those state checks at runtime?
3. What are the specific security features, obfuscation transformations, and runtime features to be deployed.

The sections below provide detail about each of these decisions.

In-code Attributes/Annotations

PreEmptive Protection is most commonly configured entirely through a config file, but most settings that map to specific code locations can be configured with in-code attributes (.NET) or annotations (Java). If developers are available to do the configuration, using attributes/annotations may be desirable because it will be easier to keep the configuration in sync with source code changes over time. However, it can also slow down the initial configuration process because updating the attributes/annotations will require rebuilding the application.

Even if in-code attributes are used, a config file is always required for settings that aren't code-specific.

For this application, configuration will be <entirely config-file based> <primarily done through in-code attributes>.

Additional resources for Dotfuscator: Custom Attribute Reference, Declarative Obfuscation

Additional resources for DashO: Annotations

Runtime State Checks

All "Checks" in PreEmptive Protection are executed at specific times and places in your application's lifecycle, specifically wherever you configure PreEmptive Protection to inject the Checks. It is typically appropriate to inject Checks at or near application startup, as well as in other places throughout the app lifecycle, especially around sensitive areas of the code.

All Checks in PreEmptive Protection share a common set of behaviors that can be configured when each Check is triggered. Each of those behaviors requires some forethought.

First, PreEmptive Protection can inject code that automatically performs pre-defined actions such as exiting the app, throwing an exception, or hanging.

Pre-defined action details will be shown in the "Specific Protections" table, below.

Second, Checks can call methods (or set fields) in the application. These calls are commonly used to change application behavior and/or to use third-party analytics platforms.

To use custom behavior, application code will have to be modified and added. Therefore this is only an option if developers are available to change the code.

This application <will> <will not> use custom behavior.

This application <will> <will not> send telemetry to <third-party analytics>.

<Details will be shown in the "Specific Protections" table, below.>

An important consideration with any Check-triggered behavior (built-in or custom) is the appropriateness of the behavior. For example, if a Debug Check is triggered, it might be appropriate to exit the app, or to limit the application's feature set. It is probably not appropriate to wipe all the application data, though, as there are potential legitimate scenarios where a debugger could be attached to a production application.

To that end, you may wish to deploy custom actions across two releases. In the first release, focus on generating telemetry and configure moderate responses that do not cause serious interruptions to valid users. Then once you are comfortable with the way you have configured the Checks and the way they are performing in production, update the application to perform more-substantial responses.

In both releases, it may be necessary to communicate the behavioral changes to other teams within the organization, e.g. the customer support team may need to be aware of the behavior and how to help customers who are experiencing it (because of e.g. false positives or malware).

Additional resources for Dotfuscator: Checks, Check Telemetry, Application Notification, Check Actions

Additional resources for DashO: Checks

Specific Protections

Area	Decisions	Additional Resources
Renaming obfuscation	Renaming obfuscation changes the human-readable names that are embedded in the application (like "getUserName") into meaningless names (like "a"). Renaming is the most common form of obfuscation, and it is highly effective, but it has the potential to break functionality of an application if a symbol is renamed and a reference to that symbol isn't. In most cases, PreEmptive Protection can automatically identify name-based references and automatically rename the reference, such as in simple reflection, standard XAML, or manifest files. But more-complex references cannot be statically analyzed, and therefore must not be renamed. PreEmptive Protection attempts to identify all such more-complex references and automatically exclude appropriate code from Renaming, but it is impossible to do this in all scenarios. Because of this, it is important to plan time in the project schedule for functional testing after Renaming is applied. It is also important to decide how aggressively to rename symbols. By default, PreEmptive Protection uses relatively safe settings, but this means that some symbols that could be renamed are probably skipped. To increase the protection provided by Renaming, the first step is usually to disable Library Mode (if you aren't building a library). From there, additional options can be configured to improve the strength of Renaming, at the expense of increased risk of functional issues. **For this application, our level of Renaming will be: <none> <with Library Mode> <without Library Mode> <as high as possible>.** If Renaming is enabled, PreEmptive Protection will generate a "map file" that preserves a record of the original and replacement names. This file will be necessary for converting any production stack traces back to their original names, and is necessary for any	Dotfuscator: Renaming Library Mode Renaming Options Map File DashO: Renaming Libraries Renaming Options Map File

	future incremental obfuscation. It should be preserved as a release artifact. **For this application, we will preserve Renaming map files by: <fill in>.**	
Control Flow obfuscation	Control Flow obfuscation changes the algorithmic structure of the code without changing the actual result of the code, to make it harder to understand what the code is doing. In most cases Control Flow obfuscation is low-risk. It is on by default, with the highest settings. In performance-sensitive areas of the code, though, Control Flow obfuscation can degrade performance, so if that is a concern then performance should be tested and/or performance-sensitive areas of the code should be excluded from Control Flow obfuscation. **For this application, we will <leave Control Flow at its defaults> <test performance and exclude areas from Control Flow as necessary>.** Another consideration for Control Flow, for Dotfuscator, is that certain obfuscation transforms will not run correctly on Mono (even though they are fine on the CLR). If this is a concern, you should configure Dotfuscator to use Mono-compatible transforms. **For this application, we will <only use Mono-compatible transforms> <use all transforms>.**	Dotfuscator: Control Flow Control Flow Exclusions Global Options (Mono-compatible transforms) DashO: Control Flow Control Flow Options
String Encryption obfuscation	String Encryption obfuscation turns static strings like an API key into jumbled strings that have to be converted back into their original value at runtime, so that the string values can't be found by searching the application binary. String Encryption obfuscation is off by default, and must be enabled for particular areas of the code. String Encryption changes how strings are accessed, so it has an effect on performance, especially when strings are retrieved in tight loops. For heavily GUI-oriented apps that don't have performance-sensitive areas, String Encryption can typically be enabled across the entire codebase. For other app scenarios, String Encryption should be avoided in performance-sensitive areas, but still	Dotfuscator: String Encryption String Encryption Editor DashO: String Encryption String Encryption Options Map File Custom Encryption

	enabled broadly to avoid drawing attention to sensitive strings. If String Encryption is enabled broadly, application performance should be tested to identify any performance changes. **For this application, String Encryption will <not be used> <be used in specific sensitive areas> <be used globally>. Performance testing <is not necessary> <will be required>.** If String Encryption is enabled in DashO it will generate a "map file" that preserves a record of which decrypters were used and where they were injected. This file will be necessary for future incremental obfuscation, and it should be preserved as a release artifact. Dotfuscator handles String Encryption differently, and does not require String Encryption map files. **For this application, we will preserve String Encryption map files by: <fill in>.**	
Debug Check	Debug Check will identify when a debugger is attached to the application. Debug Check is not available on all platforms. Please see the documentation for details. **Debug Check <will> <will not> be used.** **Debug Check will be injected at <startup> <other places>.** **Debug Check telemetry <will><will not> be sent to <third-party analytics>.** **When Debug Check triggers, the application will <pre-defined action> <and/or> <custom-developed behavior>.**	Dotfuscator: Debugging Check DashO: Debug Checks and Responses
Tamper Check	Tamper Check will identify when an application has been tampered on disk. Tamper Check is not available on all platforms. Please see the documentation for details. **Tamper Check <will> <will not> be used.**	Dotfuscator: Tamper Check Tamper Testing DashO: Tamper Check

	Tamper Check will be injected at <startup> <other places>. **Tamper Check telemetry <will><will not> be sent to <third-party analytics>.** **When Tamper Check triggers, the application will <pre-defined action> <and/or> <custom-developed behavior>.**	
Root Check	Root Check will identify when a mobile device has been "rooted" or is running in certain insecure configurations. Root check is only available for Android applications processed by DashO. **Root Check <will> <will not> be used.** **Root Check will be injected at <startup> <other places>.** **Root Check telemetry <will><will not> be sent to <third-party analytics>.** **When Root Check triggers, the application will <pre-defined action> <and/or> <custom-developed behavior>.**	DashO: Root Check
Shelf Life Check	Shelf Life Check will identify when a time-limited application is nearing or past its end-date. Shelf Life Check is not available on all platforms. Please see the documentation for details. **Shelf Life Check <will> <will not> be used.** **Shelf Life Check will be injected at <startup> <other places>.** **Shelf Life Check telemetry <will><will not> be sent to <third-party analytics>.** **When Shelf Life Check triggers, the application will <pre-defined action> <and/or> <custom-developed behavior>.**	Dotfuscator: Shelf Life Check DashO: Shelf Life Check
Additional Checks	PreEmptive Solutions periodically adds new Checks to the PreEmptive Protection products. Please check the	Dotfuscator: Understanding Checks

	latest documentation for your product to see if any additional Checks are available. **Additional Check <will> <will not> be used.** **Additional Check will be injected at <startup> <other places>.** **Additional Check telemetry <will><will not> be sent to <third-party analytics>.** **When Additional Check triggers, the application will <pre-defined action> <and/or> <custom-developed behavior>.**	DashO: Checks Overview
Removal (unused code)	PreEmptive Protection can remove unused code from your application. This helps reduce binary size and reduces the attack surface of the application. Removal works automatically by identifying the entry points of the application and statically analyzing all the code that is reachable from those points. Static analysis cannot always find all used code, however, because of e.g. dynamic reflection. Because of this, Removal has the potential to break application functionality, and it is therefore important to plan time in the project schedule for functional testing after Removal is applied. It is also important to decide how aggressively to remove code. As with Renaming, Library Mode affects Removal, preventing removal of public code elements. Both PreEmptive Protection products have additional options that allow for increasing or decreasing the aggressiveness of Removal, at the expense of increased risk of functional issues. Dotfuscator and DashO each use different default settings for Removal; it is important to confirm that the settings are what you want them to be, as part of the setup of PreEmptive Protection. **For this application, our level of Removal will be: <none> <with Library Mode> <without Library Mode> <as much as possible>.**	Dotfuscator: Pruning Removal Editor DashO: Removal Removal Options
Linking (merging)	Linking (Dotfuscator) and merging outputs (DashO) is a way to combine multiple inputs into a single output. This can simplify deployment scenarios and make it somewhat harder for an attacker to understand the structure of the application.	Dotfuscator: Linking Linking Editor DashO:

	Linking/merging defaults to off and must be enabled and configured, to be used. Linking/merging is not typically necessary for applications that will undergo further packaging after obfuscation (e.g. Xamarin, APK). **For this application, we <will> <will not> link/merge the outputs together.**	Output Options
Watermarking	Watermarking is a way of embedding a custom string into an application's structure, such that it can't be easily spotted by an attacker. The watermark can be extracted from the application to identify a particular build. **For this application, we <will> <will not> use Watermarking.**	Dotfuscator: Watermarking PreMark Editor Extracting a Watermark DashO: PreMark PreMark Options

Implementation Plan

An initial implementation of PreEmptive Protection typically follows this process:

1. **Initial risk assessment**
    a. If not already completed, this typically requires a meeting of the team and stakeholders, and follow-up communication.
2. **Project planning**
    a. If not already completed, work through the "Implementation Decisions" section of this document with the Product Owner and other key members of the team. Document the results and communicate them to the team, external stakeholders, and any other affected parties (e.g. product support).
    b. This may only require a single meeting, but if custom runtime behavior or third-party analytics will be used, additional time will be required for defining those details and documenting them.
3. **Provision the software for the development and/or build team.**
    a. If not already completed, this is typically a simple install and license verification. This shouldn't require a time allocation in the schedule, unless the software hasn't been purchased yet.
4. **First integration and "get it to work" (on a dev/build-expert machine).**
    a. Take a pre-existing build artifact and configure PreEmptive Protection to process it, using default/minimal settings as much as possible. (For an extremely-simple starting point, disable everything except Control Flow.) Verify that basic application functionality is correct, and that you understand how PreEmptive Protection fits into the build process.
    b. Depending on the target application platform and the time required for each build, this process can take anywhere from an hour to a few days. Desktop, server, and standard mobile platforms are usually easiest. Platforms with extensive packaging complexity (e.g. WAR) or with multiple output platforms (e.g. Xamarin) typically take the most work. Large applications typically take longer to process, so the cycle time (configure -> build -> test -> repeat) is longer.
5. **Provision the software in the CI (Release) build environment.**
    a. The complexity of this step depends largely on the complexity of your build environment. The PreEmptive Protection software needs to be deployed to the build hosts, and licensing has to be configured correctly.
    b. In simple setups this is an hour's work; more-complex setups should be estimated by the Build Experts.
6. **Integration into CI (Release) build.**

a. Configure PreEmptive Protection to be automatically run on every (release) build of the application, typically via one of the build-tool integrations (e.g. MSBuild, Gradle, command-line). Configure any pre- or post-build events, and any signing required. Commit the PreEmptive Protection config file to source control, so it can be executed in the CI (Release) build.

b. Most of the complexity of this process will have been discovered in step #2, but there may still be significant work remaining.

c. Depending on the target application platform and the time required for each build, this process can take anywhere from an hour to a few days, for the same reasons as #2(b).

7. Iterate:

a. Develop any custom functionality.

i. If Checks will be injected and if they will trigger custom behavior, that custom behavior should be implemented now.

ii. The time required for this step depends heavily on the complexity of the application and the complexity of the custom behavior. Estimate this phase as you would any other small software work.

b. Tailor the configuration.

i. Incrementally increase the level of protection: enable additional transforms, configure the transforms to provide stronger protection, and adjust the Includes and Excludes as desired.

ii. Typically each round of this step is quick: make the desired change, commit it, and kick off a new build. Additional time may be required for reading documentation and understanding configuration options. Time will be required for the actual build to run.

c. Test and verify functionality and/or performance.

i. Depending on the configuration change made, different types of testing are required.

ii. Renaming, Removal, and Linking require broad functional testing.

iii. Control Flow and String Encryption may require targeted performance testing.

iv. Injected Checks require targeted functional testing to ensure the Checks are triggering appropriately and that when they are triggered, the desired behaviors are observed. (PreEmptive Protection includes tools to help with this testing.)

v. Watermarking requires testing that the watermark is present (via PreEmptive Protection tools).

vi. The time required for testing should be the same as your typical testing process. Assume at least 3 iterations through this process, although there could be more depending on the aggressiveness of the configuration.

d. Resolve any issues.

i. Functional issues are typically caused by Renaming, Removal, or Linking. Examine the build warnings and use Inclusions and Exclusions to narrow down the source of the issue.

ii. Performance issues are typically caused by Control Flow or String Encryption. Try excluding those areas, or modifying the code to make it more efficient.

iii. The resolution process may be slow at first as you learn how to troubleshoot unexpected behavior, but it will speed up as you get familiar with the process.

8. First protected release.

To "turn up the dials" in future releases, repeat the cycle in step #4 as needed.

If future releases will continue to use the same protection settings then no additional work is required beyond the usual release testing. PreEmptive Protection will continue to protect your builds, automatically.

Implementation Timeline (estimated)

Based on the implementation decisions above, the estimated implementation timeline is as follows:

Example implementation timeline estimates

This timeline assumes the following conditions:

- *A desktop app with a 15-minute build cycle and a single build server.*
- *Functional and/or targeted performance test passes typically take a few hours.*
- *The team consists of one person in the role of Product Owner & Project Manager, two Developers who are also Build Experts, and one Tester.*
- *The Risk Assessment did not identify any areas that required extra attention.*
- *Configuration will be performed entirely through the config file.*
- *Renaming will be applied with Library Mode turned off, but otherwise no additional changes from the defaults. Map files will be preserved alongside existing build outputs, which are already preserved by existing infrastructure.*
- *Control Flow will be enabled with default settings.*
- *String Encryption will be enabled globally, and exploratory performance testing will be performed. Map files will be preserved alongside existing build outputs.*

- *Debug and Tamper Checks will be used, but their only action will be to send telemetry to an already-used third-party analytics platform. They will be injected at application startup, and at each place a new screen is loaded.*
- *Removal will be enabled with Library Mode turned off, but no other changes from the defaults.*
- *Linking and Watermarking will not be used.*

Based on those assumptions, the estimated timeline is:

1. Risk assessment: already complete
2. Project planning: already complete
3. Initial provisioning: already complete
4. First integration: 1-2 days
5. Build environment provisioning and initial CI integration: 1-2 days
6. Iteration of protection configuration:
    a. Start by enabling all selected protections except Checks, and get everything stable: 1-2 days per cycle; assume 2-3 cycles; total of 2-6 days
    b. Then inject the checks and test their behavior: 1 day per cycle; assume 2-3 cycles; total of 2-3 days

Total: 6-13 days (elapsed).