



BETA

SOFTWARE ASSURANCE MATURITY MODEL

V0.8 BETA

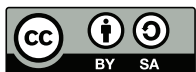


ACKNOWLEDGEMENTS

This work was originally created by Pravir Chandra (chandra@cognosticus.com) and funded through generous support from Fortify Software, Inc. This document is currently maintained and updated through the SAMM project (<http://www.opensamm.org>).

This work would not be possible without the support of many individual reviewers and experts that offered critical feedback. Thanks to Fabio Arciniegas, Jeff Payne, and Gunnar Peterson. Additionally, thanks to the Fortify team of Jonathan Carter, Brian Chess, Justin Derry, and Jeff Piper.

LICENSE



This work is licensed under the Creative Commons Attribution-Share Alike 3.0 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Contents

Introduction	2
❖ Motivation & Rationale	2
❖ Role Definitions	2
❖ Intended Audience	2
❖ Using the Maturity Model	3
Maturity Model	4
Alignment & Governance	
❖ Education & Guidance	6
❖ Standards & Compliance	10
❖ Strategic Planning	14
Requirements & Design	
❖ Threat Modeling	18
❖ Security Requirements	22
❖ Defensive Design	26
Verification & Assessment	
❖ Architecture Review	30
❖ Code Review	34
❖ Security Testing	38
Deployment & Operations	
❖ Vulnerability Management	42
❖ Infrastructure Hardening	46
❖ Operational Enablement	50
Roadmaps	54
❖ Independent Software Vendors	56
❖ Online Services Providers	58
❖ Financial Services Organizations	coming soon
❖ Government Organizations	coming soon
Case Studies	60
❖ VirtualWare - an Independent Software Vendor	62
❖ Moogle - an Online Services Provider	coming soon
❖ UniGroup - a Financial Services Organization	coming soon
❖ SocialLink - a Government Organization	coming soon

Introduction

Background information for the maturity model

MOTIVATION & RATIONALE

This maturity model for software assurance programs was built with several guiding principles and goals in mind.

First, a properly built assurance program requires disparate activities within an organization and changing an organization's behavior is hard. There must be well defined short-term goals at each stage of improvement and long-term goals must be defined incrementally to efficiently build a program without excessive overhead cost.

Second, achievement of improvements to an assurance program must be measurable and simple to understand. Overly nuanced or vague activity definitions make general-case usage of a maturity model difficult and error-prone.

Third, there is no single assurance program roadmap that suits all organizations. It should be built to suit the specific risks facing an organization based on the way software it built and utilized by the business. Thus, a maturity model should define improvement orthogonally to the type of business and demonstrate common usage scenarios with recommended roadmaps and case studies.

ROLE DEFINITIONS

Developers

Individuals performing low-level design and implementation of the software

Architects

Individuals performing high-level design work and large scale system engineering

Managers

Individuals performing day-to-day management of development staff

QA Testers

Individuals performing routine testing and pre-release verification of software

Security Auditors

Individuals with technical security knowledge related to software being produced

Business Owners

Individuals performing key decision making on software and its business requirements

Support Operations

Individuals performing customer support or direct technical operations support

INTENDED AUDIENCE

This document contains information relevant to many roles involved in software production, therefore can be used by a variety of individuals. The following recommendations are given to simplify getting started with SAMM

Managers, Business Owners, Security Auditors, Architects

For individuals seeking to understand how to lead a software assurance program and manage a program going forward, start by understanding the security Functions at a high-level by reading each one-page overview under the Maturity Areas. After this, it is probably most helpful to peruse the Roadmaps and the Case Studies relevant to your organization. As needed, refer back to the Maturity Areas section for details on specific Functions and Objectives.

Developers, Architects, QA Testers, Security Auditors, Support Operations

For individuals seeking to understand how to perform specific activities on the project level in order to improve provision of a security Function, select corresponding parts of the Objective and Activity definitions in the Maturity Areas section. As needed, refer to related Objectives and other resources for background and contextual information. To understand further improvement of the Function over time, readers should review the subsequent Objective(s) as well.

Using the Maturity Model

This maturity model describes a wide variety of activities in which an organization could engage to reduce security risks and increase assurance. At the highest level, there are four **Disciplines** that represent general types of activities:

Alignment & Governance **Requirements & Design** **Verification & Assessment** **Deployment & Operations**

Grouped under each Discipline are three **Functions** that each represent an area of security-related activities on which an organization *could* improve over time:

- | | | | |
|--------------------------|-------------------------|-----------------------|----------------------------|
| ◆ Education & Guidance | ◆ Threat Modeling | ◆ Architecture Review | ◆ Vulnerability Management |
| ◆ Standards & Compliance | ◆ Security Requirements | ◆ Code Review | ◆ Infrastructure Hardening |
| ◆ Strategic Planning | ◆ Defensive Design | ◆ Security Testing | ◆ Operational Enablement |

Each Function has three successive **Objectives** that define increasingly more sophisticated goals for the organization with respect to the given Function. An organization achieves an Objective by following the corresponding guidance in this document, and after that, they can work toward the next Objective under the given Function. Generally, the successive Objectives under each Function represent:

- | | | | |
|--|---|---|---|
| ❖ 0 : Implicit starting point with the Function unfulfilled | ❖ 1 : Initial understanding and ad hoc provision of Function | ❖ 2 : Increase efficiency and/or effectiveness of the Function | ❖ 3 : Comprehensive mastery of the Function at scale |
|--|---|---|---|

Most organizations will already be performing some of these activities, so an initial assessment should be conducted to map the organization's performance into the twelve Functions by assigning a score (0-3) for each based upon Objectives already complete. After this, an organization should iterate based on business drivers and organization-specific information by choosing which Functions to improve and then accomplishing the next successive Objective for each.

Maturity Model

This section defines the building blocks comprising the Maturity Model itself. Under each of the four high-level Disciplines there are three security Functions defining the breadth of security-related activities that could be performed. In turn, each Function has three levels of provision with corresponding activities, success metrics, personnel overhead, and associated costs.



Education & Guidance	6
❖ 1: Offer development staff access to resources around the topics of secure programming and deployment.	7
❖ 2: Educate all personnel in the software life-cycle with role-specific guidance on secure development.	8
❖ 3: Mandate comprehensive security training and certify personnel for baseline knowledge.	9
Standards & Compliance	10
❖ 1: Understand relevant governance and compliance drivers to the organization.	11
❖ 2: Establish security and compliance baseline and understand per-project risks.	12
❖ 3: Require compliance and measure projects against organization-wide standards and policies.	13
Strategic Planning	14
❖ 1: Establish unified strategic roadmap for software security within the organization.	15
❖ 2: Understand relative value of data and software assets and choose risk tolerance.	16
❖ 3: Align security expenditure with relevant business indicators and asset value.	17

Threat Modeling	18
❖ 1: Identify and understand high-level threats to the organization and individual projects.	19
❖ 2: Increase accuracy of threat assessment and improve granularity of per-project understanding.	20
❖ 3: Concretely tie compensating controls to each threat against internal and third-party software.	21
Security Requirements	22
❖ 1: Consider security explicitly during the software requirements process.	23
❖ 2: Generate security requirements from application-specific risks and abuse scenarios.	24
❖ 3: Mandate security requirements for all software projects and third-party dependencies	25
Defensive Design	26
❖ 1: Provide proactive design guidance to project teams and make perimeter security enhancements.	27
❖ 2: Build software with comprehensive security enhancements within the design process.	28
❖ 3: Measure proactive effort of project teams to make defensive design more efficient.	29

Architecture Review	30
❖ 1: Support ad hoc reviews of software architecture to ensure baseline mitigations for known risks.	31
❖ 2: Offer assessment services to review software design against comprehensive best practices for security.	32
❖ 3: Require assessments and validate artifacts to develop detailed understanding of protection mechanisms.	33
Code Review	34
❖ 1: Opportunistically find basic code-level vulnerabilities and other high-risk security issues.	35
❖ 2: Make code review during development more accurate and efficient through automation.	36
❖ 3: Mandate comprehensive code review process to discover both language-level and application-specific risks.	37
Security Testing	38
❖ 1: Enable testing process to perform basic security tests based on software requirements.	39
❖ 2: Make security testing during development more complete and efficient through automation.	40
❖ 3: Require application-specific security testing to ensure baseline security before deployment.	41

Vulnerability Management	42
❖ 1: Understand high-level plan for responding to vulnerability reports or incidents.	43
❖ 2: Elaborate expectations for response process to improve consistency and communications.	44
❖ 3: Improve analysis and data gathering within response process for feedback into proactive planning.	45
Infrastructure Hardening	46
❖ 1: Understand baseline operational environment for applications and software components.	47
❖ 2: Improve confidence in application operations by hardening the operating environment.	48
❖ 3: Validate application health and status of operational environment against known best practices.	49
Operational Enablement	50
❖ 1: Enable communications between development teams and operators for critical security-relevant data.	51
❖ 2: Improve expectations for continuous secure operations through provision of detailed procedures.	52
❖ 3: Mandate communication of security information and validate artifacts for completeness.	53

Education & Guidance

Education & Guidance function is focused on arming personnel involved in the software life-cycle with knowledge about application security. With improved access to information, project teams will be better able to proactively identify and mitigate the specific security risks that apply to your organization.

One major theme for improvement across the objectives is providing training for employees, either through instructor-led sessions or computer-based modules. As an organization progresses through these levels, a broad base of training is built by starting with developers and moving toward more roles throughout the organization, culminating with addition of role-based certification of staff to ensure comprehension of the material.

In addition to training, this function also requires pulling security-relevant information into guidelines that serve as reference information to staff. This builds a foundation for establishing a baseline expectation for security practices in your organization, and later allows for incremental improvement once usage of the guidelines has been adopted.

OBJECTIVES	Offer development staff access to resources around the topics of secure programming and deployment.	Educate all personnel in the software life-cycle with role-specific guidance on secure development.	Mandate comprehensive security training and certify personnel for baseline knowledge.
ACTIVITIES	<ul style="list-style-type: none"> ❖ Conduct technical security awareness training ❖ Build and maintain technical guidance 	<ul style="list-style-type: none"> ❖ Conduct role-specific application security training ❖ Utilize security coaches to enhance project teams 	<ul style="list-style-type: none"> ❖ Create application security support portal ❖ Establish role-based examination/certification
RESULTS	<ul style="list-style-type: none"> ◆ Increased developer awareness on the most common problems at the code level ◆ Maintain software with rudimentary security best-practices in place ◆ Set baseline for security know-how among technical staff ◆ Enable qualitative security checks for baseline security knowledge 	<ul style="list-style-type: none"> ◆ End-to-end awareness of the issues that leads to security vulnerabilities at the product, design, and code levels ◆ Build plans to remediate vulnerabilities and design flaws in ongoing projects ◆ Enable qualitative security checkpoints at requirements, design, and development stages ◆ Deeper understanding of security issues encourages more proactive security planning 	<ul style="list-style-type: none"> ◆ Efficient remediation of vulnerabilities in both ongoing and legacy code bases ◆ Quickly understand and mitigate against new attacks and threats ◆ Judge security-savvy of staff and measure against a common standard ◆ Establish fair incentives toward security awareness

EG **1**

EG **2**

EG **3**

Education & Guidance

EG

1

Offer development staff access to resources around the topics of secure programming and deployment.

ACTIVITIES

Conduct technical security awareness training

Either internally or externally sourced, conduct security training for technical staff that covers the basic tenets of application security. Generally, this can be accomplished via instructor-led training in 1-2 days or via computer-based training with modules taking about the same amount of time per developer.

Course content should cover both conceptual and technical information. Appropriate topics include high-level best practices surrounding input validation, output encoding, error handling, logging, authentication, authorization. Additional coverage of commonplace software vulnerabilities is also desirable such as the OWASP Top 10 for web applications, or a more appropriate modified list for software being developed for other paradigms (embedded devices, client-server applications, back-end transaction systems, etc.). Wherever possible, use code samples and lab exercises in the specific programming language(s) that applies.

To rollout such training, it is recommended to mandate annual security training and then hold courses (either instructor-led or computer-based) as often as required based on development head-count.

Build and maintain technical guidance

For development staff, assemble a list of approved documents, web pages, and technical notes that provide technology-specific security advice. These references can be assembled from many publicly available resources on the Internet. In cases where very specialized or proprietary technologies permeate the development environment, utilize senior, security-savvy staff to build security notes over time to create such a knowledge base in an ad hoc fashion.

Ensure management is aware of the resources and briefs oncoming staff about their expected usage. Try to keep the reference list small and up-to-date to avoid clutter and irrelevance. Once a comfort-level has been established, the technical references can be used as a qualitative checklist to ensure that the guidance has been read, understood, and followed in the development process.

RESULTS

- ◆ Increased developer awareness on the most common problems at the code level
- ◆ Maintain software with rudimentary security best-practices in place
- ◆ Set baseline for security know-how among technical staff
- ◆ Enable qualitative security checks for baseline security knowledge

SUCCESS METRICS

- ◆ >50% development staff briefed on security issues within past 1 year
- ◆ >75% senior development/architect staff briefed on security issues within past 1 year
- ◆ Launch technical guidance within 3 months of first training

COSTS

- ◆ Training course buildout or license
- ◆ Ongoing maintenance of technical guidance

PERSONNEL

- ◆ Developers (1-2 days/yr)
- ◆ Architects (1-2 days/yr)

RELATED OBJECTIVES

- ◆ Standards & Compliance - 2
- ◆ Security Requirements - 1
- ◆ Defensive Design - 1

RELATED COMPLIANCE

- ◆ PCI v1.1, section 6.5
- ◆ PCI v1.1, section 12.5.1
- ◆ PCI v1.1, section 12.6

Education & Guidance

Educate all personnel in the software life-cycle with role-specific guidance on secure development.

RESULTS

- ◆ End-to-end awareness of the issues that leads to security vulnerabilities at the product, design, and code levels
- ◆ Build plans to remediate vulnerabilities and design flaws in ongoing projects
- ◆ Enable qualitative security checkpoints at requirements, design, and development stages
- ◆ Deeper understanding of security issues encourages more proactive security planning

ADD'L SUCCESS METRICS

- ◆ >60% development staff trained within past 1 year
- ◆ >50% management/analyst staff trained within past 1 year
- ◆ >80% senior development/architect staff trained within past 1 year
- ◆ >3.0 Likert on usefulness of training courses

ADD'L COSTS

- ◆ Training library buildout or license
- ◆ Security-savvy staff for hands-on coaching

ADD'L PERSONNEL

- ◆ Developers (2 days/yr)
- ◆ Architects (2 days/yr)
- ◆ Managers (1-2 days/yr)
- ◆ Business Owners (1-2 days/yr)
- ◆ QA Testers (1-2 days/yr)
- ◆ Security Auditors (1-2 days/yr)

RELATED OBJECTIVES

- ◆ Vulnerability Management - 1
- ◆ Architecture Review - 2
- ◆ Defensive Design - 2

RELATED COMPLIANCE

- ◆ PCI v1.1, section 6.5
- ◆ PCI v1.1, section 12.5.1
- ◆ PCI v1.1, section 12.6
- ◆ PCI v1.1, section 12.9.4

ACTIVITIES

Conduct role-specific application security training

Conduct security training for staff that highlights application security in the context of each role's job function. Generally, this can be accomplished via instructor-led training in 1-2 days or via computer-based training with modules taking about the same amount of time per person.

For managers and requirements specifiers, course content should feature security requirements planning, vulnerability and incident management, threat modeling, and misuse/abuse case design.

Tester and auditor training should focus on training staff to understand and more effectively analyze software for security-relevant issues. As such, it should feature techniques for code review, architecture and design analysis, runtime analysis, and effective security test planning.

Expand technical training targeting developers and architects to include other relevant topics such as security design patterns, tool-specific training, threat modeling and software assessment techniques.

To rollout such training, it is recommended to mandate annual security awareness training and periodic specialized topics training. Course should be available (either instructor-led or computer-based) as often as required based on head-count per role.

Utilize security coaches to enhance project teams

Using either internal or external experts, make security-savvy staff available to project teams for consultation. Further, this coaching resource should be advertised internally to ensure that staff are aware of its availability.

The coaching staff can be created by recruiting experienced individuals within the organization to spend some percentage of their time (10% max) performing coaching activities. The coaches should communicate between one another to ensure they are aware of each other's area of expertise and route questions accordingly for efficiency.

While coaches can be used at any point in the software life-cycle, appropriate times to use the coaches include during initial product conception, before completion of functional or detailed design specification(s), when issues arise during development, test planning, and when operational security incidents occur.

Over time, the internal network of coaching resources can be used as points-of-contact for communicating security-relevant information throughout the organization as well as being local resources that have greater familiarity with the ongoing project teams than a purely centralized security team might.

Education & Guidance

EG 3

Mandate comprehensive security training and certify personnel for baseline knowledge.

ACTIVITIES

Create application security support portal

Building upon written resources on topics relevant to application security, create and advertise a centralized repository (usually an internal web site). The guidelines themselves can be created in any way that makes sense for the organization, but an approval board and straightforward change control processes must be established.

Beyond static content in the form of best-practices lists, tool-specific guides, FAQs, and other articles, the support portal should feature interactive components such as mailing lists, web-based forums, or wikis to allow internal resources to cross-communicate security relevant topics and have the information cataloged for future reference.

The content should be cataloged and easily searchable based upon several common factors such as platform, programming language, pertinence to specific third party libraries or frameworks, life-cycle stage, etc. Project teams creating software should align themselves early in product development to the specific guidelines that they will follow. In product assessments, the list of applicable guidelines and product-related discussions should be used as audit criteria.

Establish role-based examination/certification

Either per role or per training class/module, create and administer aptitude exams that test people for comprehension and utilization of security knowledge. Typically, exams should be created based on the role-based curricula and target a minimum passing score around 75% correct. While staff should be required to take applicable training or refresher courses annually, certification exams should be required biannually at a minimum.

Based upon pass/fail criteria or exceptional performance, staff should be ranked into tiers such that other security-related activities could require individuals of a particular certification level to sign-off before the activity is complete, e.g. an uncertified developer cannot pass a design into implementation without explicit approval from a certified architect. This provides granular visibility on an per-project basis for tracking security decisions with individual accountability. Overall, this provides a foundation for rewarding or penalizing staff for making good business decisions regarding application security.

RESULTS

- ◆ Efficient remediation of vulnerabilities in both ongoing and legacy code bases
- ◆ Quickly understand and mitigate against new attacks and threats
- ◆ Judge security-savvy of staff and measure against a common standard
- ◆ Establish fair incentives toward security awareness

ADD'L SUCCESS METRICS

- ◆ >80% staff certified within past 1 year

ADD'L COSTS

- ◆ Certification examination buildout or license
- ◆ Ongoing maintenance and change control for application security support portal
- ◆ Human-resources and overhead cost for implementing employee certification

ADD'L PERSONNEL

- ◆ Developers (1 day/yr)
- ◆ Architects (1 day/yr)
- ◆ Managers (1 day/yr)
- ◆ Business Owners (1 day/yr)
- ◆ QA Testers (1 day/yr)
- ◆ Security Auditors (1 day/yr)

RELATED OBJECTIVES

- ◆ Standards & Compliance - 2 & 3

RELATED COMPLIANCE




- ◆ PCI v1.1, section 2.2
- ◆ PCI v1.1, section 6.5
- ◆ PCI v1.1, section 12.5.1
- ◆ PCI v1.1, section 12.6
- ◆ PCI v1.1, section 12.9.4

Standards & Compliance

The Standards & Compliance function is focused on understanding and meeting external regulatory requirements while also driving internal security standards to meet those in a way that's aligned with the business purpose of the organization's software projects.

A driving theme for improvement within this function is focus on project-level audits that gather information about the organization's behavior in order to check that expectations are being met. By introducing routine audits that start out lightweight and grow in depth over time, organizational change is achieved iteratively.

In a sophisticated form, provision of this function entails organization-wide understanding of both internal standards and external compliance drivers while also maintaining low-latency checkpoints with project teams to ensure no project is operating outside expectations without visibility.

OBJECTIVES	Understand relevant governance and compliance drivers to the organization.	Establish security and compliance baseline and understand per-project risks.	Require compliance and measure projects against organization-wide standards and policies.
ACTIVITIES	<ul style="list-style-type: none"> ❖ Identify and monitor external compliance drivers ❖ Build and maintain compliance checklist 	<ul style="list-style-type: none"> ❖ Build internal standards for security and compliance ❖ Establish project audit practice 	<ul style="list-style-type: none"> ❖ Create compliance gates for projects ❖ Adopt solution for audit data collection
RESULTS	<ul style="list-style-type: none"> ◆ Increased assurance for handling third-party audit with positive outcome ◆ Alignment of internal resources based on priority of compliance requirements ◆ Timely discovery of evolving regulatory requirements that affect your organization 	<ul style="list-style-type: none"> ◆ Awareness for project teams regarding expectations for both security and compliance ◆ Business owners that better understand specific compliance risks in their product lines ◆ Optimized approach for efficiently meeting compliance with opportunistic security improvement 	<ul style="list-style-type: none"> ◆ Organization-level visibility of accepted risks due to non-compliance ◆ Concrete assurance for compliance at the project level ◆ Accurate tracking of past project compliance history ◆ Efficient audit process leveraging tools to cut manual effort
			

Standards & Compliance

SC 1

Understand relevant governance and compliance drivers to the organization.

ACTIVITIES

Identify and monitor external compliance drivers

While an organization might have a wide variety of compliance requirements, this activity is specifically oriented around those that either directly or indirectly affect the way in which the organization builds or uses software and/or data. Leverage internal staff focused on compliance if available.

Based on the organization's core business, conduct research and identify third-party regulatory standards with which compliance is required or considered an industry norm. Possibilities include the Sarbanes-Oxley Act (SOX), the Payment Card Industry Data Security Standards (PCI-DSS), the Health Insurance Portability and Accountability Act (HIPAA), etc. After reading and understanding each third-party standard, collect specific requirements related to software and data and build a consolidated list that maps each driver (third-party standard) to each of its specific requirements for security. At this stage, try to limit the amount of requirements by dropping anything considered optional or only recommended.

At a minimum, conduct research at least biannually to ensure the organization is keeping updated on changes to third-party standards. Depending upon the industry and the importance of compliance, this activity can vary in effort and personnel involvement, but should always be done explicitly.

Build and maintain compliance checklist

Based upon the consolidated list of software and data-related requirements from compliance drivers, elaborate the list by creating a corresponding response statement to each requirement. Sometimes called control statements, each response should capture the concept of what the organization does to ensure the requirement is met (or to note why it does not apply).

Since typical audit practice often involves checking a control statement for sufficiency and then measuring the organization against the control statement itself, it is critical that they accurately represent actual organizational practices. Also, many requirements can be met by instituting simple, lightweight process elements to cover base-line compliance prior to evolving the organization for better assurance down the road.

Working from the consolidated list, identify major gaps to feed the future planning efforts with regard to building the assurance program.

Communicate information about compliance gaps with stakeholders to ensure awareness of the risk from non-compliance.

At a minimum, update and review control statements with stakeholders at least biannually. Depending on the number of compliance drivers, it may make sense to perform updates more often.

RESULTS

- ◆ Increased assurance for handling third-party audit with positive outcome
- ◆ Alignment of internal resources based on priority of compliance requirements
- ◆ Timely discovery of evolving regulatory requirements that affect your organization

SUCCESS METRICS

- ◆ >1 compliance discovery meeting in past 6 months
- ◆ Compliance checklist completed and updated within past 6 months
- ◆ >1 compliance review meeting with stakeholders in past 6 months

COSTS

- ◆ Initial creation and ongoing maintenance of compliance checklist

PERSONNEL

- ◆ Architects (1 day/yr)
- ◆ Managers (2 days/yr)
- ◆ Business Owners (1-2 days/yr)

RELATED OBJECTIVES

- ◆ Strategic Planning - I

RELATED COMPLIANCE

- ◆

Standards & Compliance

Establish security and compliance baseline and understand per-project risks.

RESULTS

- ◆ Awareness for project teams regarding expectations for both security and compliance
- ◆ Business owners that better understand specific compliance risks in their product lines
- ◆ Optimized approach for efficiently meeting compliance with opportunistic security improvement

ADD'L SUCCESS METRICS

- ◆ >75% of staff briefed on internal standards in past 6 months
- ◆ >80% stakeholders aware of compliance status against internal standards

ADD'L COSTS

- ◆ Internal standards buildout or license
- ◆ Per-project overhead from compliance with internal standards and audit

ADD'L PERSONNEL

- ◆ Architects (1 days/yr)
- ◆ Managers (1 days/yr)
- ◆ Security Auditors (2 days/project/yr)

RELATED OBJECTIVES

- ◆ Education & Guidance - 1 & 3
- ◆ Strategic Planning - 2
- ◆ Security Requirements - 1 & 3
- ◆ Defensive Design - 3
- ◆ Code Review - 3
- ◆ Architecture Review - 3
- ◆ Infrastructure Hardening - 3

RELATED COMPLIANCE

- ◆

ACTIVITIES

Build internal standards for security and compliance

Beginning with a current compliance checklist, review regulatory standards and note any optional or recommended security requirements. Also, the organization should conduct a small amount of research to discover any potential future changes in compliance requirements that are relevant.

Augment the list with any additional requirements based on known business drivers for security. Often it is simplest to consult existing guidance being provided to development staff and gather a set of best practices.

Group common/similar requirements and rewrite each group as more generalized/simplified statements that meet all the compliance drivers as well as provide some additional security value. Work through this process for each grouping with the goal of building a set of internal standards that can be directly mapped back to compliance drivers and best practices.

It is important for the unified standard to not contain requirements that are difficult or excessively costly for project teams to comply. Approximately 80% of projects should be able to comply with minimal disruption. This requires a good communications program being set up to advertise the new internal standards and assist teams with compliance if needed.

Establish project audit practice

Create a simple audit process for project teams to request and receive an audit against internal standards. Audits are typically performed by security auditors but can also be conducted by security-savvy staff as long as they are knowledgeable about the internal standards.

Based upon any known business risk indicators, projects can be prioritized as the audit queue is triaged such that high-risk software is assessed sooner or more frequently. Additionally, low-risk projects can have internal audit requirements loosened to make the audit practice more cost-effective.

Overall, each active project should undergo an audit at least biannually. Generally, subsequent audits after the initial will be simpler to perform if sufficient audit information about the application is retained.

Advertise this service to business owners and other stakeholders so that they may request an audit for their projects. Detailed pass/fail results per requirement from the internal standards should be delivered to project stakeholders for evaluation. Where practical, audit results should also contain explanations of impact and remediation recommendations.

Standards & Compliance

SC 3

Require compliance and measure projects against organization-wide standards and policies.

ACTIVITIES

Create compliance gates for projects

Once an organization has established internal standards for security, the next level of enforcement is to set particular points in the project life-cycle where a project cannot pass until it is audited against the internal standards and found to be in compliance.

Usually, the compliance gate is placed at the point of software release such that they are not allowed to publish a release until the compliance check is passed. It is important to provide enough time for the audit to take place and remediation to occur, so generally the audit should begin earlier, for instance when a release is given to QA.

Despite being a firm compliance gate, legacy or other specialized projects may not be able to comply, so an exception approval process must also be created. No more than about 20% of all projects should have exception approval.

Adopt solution for audit data collection

Organizations conducting regular audits of project teams generate a large amount of audit data over time. Automation should be utilized to assist in automated collection, manage collation for storage and retrieval, and to limit individual access to sensitive audit data.

For many concrete requirements from the internal standards, existing tools such as code analyzers, application penetration testing tools, monitoring software, etc. can be customized and leveraged to automate compliance checks against internal standards. The purpose of automating compliance checks is to both improve efficiency of audit as well as enable more staff to self-check for compliance before a formal audit takes place. Additionally, automated checks are less error-prone and allow for lower latency on discovery of problems.

Information storage features should allow centralized access to current and historic audit data per project. Automation solutions must also provide detailed access control features to limit access to approved individuals with valid business purpose for accessing the audit data.

All instructions and procedures related to accessing compliance data as well as requesting access privileges should be advertised to project teams. Additional time may be initially required from security auditors to bootstrap project teams.

RESULTS

- ◆ Organization-level visibility of accepted risks due to non-compliance
- ◆ Concrete assurance for compliance at the project level
- ◆ Accurate tracking of past project compliance history
- ◆ Efficient audit process leveraging tools to cut manual effort

ADD'L SUCCESS METRICS

- ◆ >80% projects in compliance with internal standards as seen by audit
- ◆ <50% time per audit as compared to manual

ADD'L COSTS

- ◆ Buildout or license tools to automate audit against internal standards
- ◆ Ongoing maintenance of audit gates and exception process

ADD'L PERSONNEL

- ◆ Developers (1 days/yr)
- ◆ Architects (1 days/yr)
- ◆ Managers (1 days/yr)

RELATED OBJECTIVES

- ◆ Education & Guidance - 3
- ◆ Code Review - 2
- ◆ Security Testing - 2

RELATED COMPLIANCE

- ◆


Strategic Planning

The Strategic Planning function is focused on establishing the framework within an organization for a software security assurance program. This is the most fundamental step in defining security goals in a way that's aligned with the organization's real business risk.


By starting with lightweight risk profiles, a organization grows into more advanced risk classification schemes for application and data assets over time. With additional insight on relative risk measures, an organization can tune its project-level security goals and develop granular roadmaps to make the security program more efficient.

At the more advanced levels within this function, an organization draws upon many data sources, both internal and external, to collect metrics and qualitative feedback on the security program. This allows fine tuning of cost outlay versus the realized benefit at the program level.


OBJECTIVES	Establish unified strategic roadmap for software security within the organization.	Understand relative value of data and software assets and choose risk tolerance.	Align security expenditure with relevant business indicators and asset value.
ACTIVITIES	<ul style="list-style-type: none"> ❖ Estimate overall business risk profile ❖ Build and maintain assurance program roadmap 	<ul style="list-style-type: none"> ❖ Classify data and applications based on business risk ❖ Establish per-classification security goals 	<ul style="list-style-type: none"> ❖ Conduct periodic industry-wide cost comparisons ❖ Collect metrics for historic security spend
RESULTS	<ul style="list-style-type: none"> ◆ Concrete list of the most critical business-level risks caused by software ◆ Tailored roadmap that addresses the security needs for your organization with minimal overhead ◆ Organization-wide understanding of how the assurance program will grow over time 	<ul style="list-style-type: none"> ◆ Customized assurance plans per project based on core value to the business ◆ Organization-wide understanding of security-relevance of data and application assets ◆ Better informed stakeholders with respect to understanding and accepting risks 	<ul style="list-style-type: none"> ◆ Information to make informed case-by-case decisions on security expenditures ◆ Estimates of past loss due to security issues ◆ Per project consideration of security expense versus loss potential ◆ Industry-wide due diligence with regard to security



SP 1



SP 2



SP 3

Strategic Planning

SP

1

Establish unified strategic roadmap for software security within the organization.

ACTIVITIES

Estimate overall business risk profile

Interview business owners and stakeholders and create a list of worst-case scenarios across the organization's various application and data assets. Based on the way in which your organization builds, uses, or sells software, the list of worst-case scenarios can vary widely, but common issues include data theft or corruption, service outages, monetary loss, reverse engineering, account compromise, etc.

After broadly capturing worst-case scenario ideas, collate and select the most important based on collected information and knowledge about the core business. Any number can be selected, but aim for at least 3 and no more than 7 to make efficient use of time and keep the exercise focused.

Elaborate a description of each of the selected items and document details of contributing worst-case scenarios, potential contributing factors, and potential mitigating factors for the organization.

The final business risk profile should be reviewed with business owners and other stakeholders for understanding.

Build and maintain assurance program roadmap

Understanding the main business risks to the organization, evaluate the current performance of the organization against each the twelve Functions. Assign a score for each Function from 1, 2, or 3 based on the corresponding Objective if the organization passes all the cumulative success metrics. If no success metrics are being met, assign a score of 0 to the Function.

Once a good understanding of current status is obtained, the next goal is to identify the Functions that will be improved in the next iteration. Select them based on business risk profile, other business drivers, compliance requirements, budget tolerance, etc. Once Functions are selected, the goals of the iteration are to achieve the next Objective under each.

Iterations of improvement on the assurance program should be approximately 3-6 months, but an assurance strategy session should take place at least every 3 months to review progress on activities, performance against success metrics and other business drivers that may require program changes.

RESULTS

- ◆ Concrete list of the most critical business-level risks caused by software
- ◆ Tailored roadmap that addresses the security needs for your organization with minimal overhead
- ◆ Organization-wide understanding of how the assurance program will grow over time

SUCCESS METRICS

- ◆ >80% of stakeholders briefed on business risk profile in past 6 months
- ◆ >80% of staff briefed on assurance program roadmap in past 3 months
- ◆ >1 assurance program strategy session in past 3 months

COSTS

- ◆ Buildout and maintenance of business risk profile
- ◆ Quarterly evaluation of assurance program

PERSONNEL

- ◆ Developers (1 day/yr)
- ◆ Architects (4 days/yr)
- ◆ Managers (4 days/yr)
- ◆ Business Owners (4 days/yr)
- ◆ QA Testers (1 day/yr)
- ◆ Security Auditor (4 days/yr)

RELATED OBJECTIVES

- ◆ Standards & Compliance - 1
- ◆ Threat Modeling - 1
- ◆ Security Requirements - 2

RELATED COMPLIANCE

- ◆

Strategic Planning

Understand relative value of data and software assets and choose risk tolerance.

RESULTS

- ◆ Customized assurance plans per project based on core value to the business
- ◆ Organization-wide understanding of security-relevance of data and application assets
- ◆ Better informed stakeholders with respect to understanding and accepting risks

ADD'L SUCCESS METRICS

- ◆ >90% applications and data assets evaluated for risk classification in past 6 months
- ◆ >80% of staff briefed on relevant application and data risk ratings in past 6 months
- ◆ >80% of staff briefed on relevant assurance program roadmap in past 3 months

ADD'L COSTS

- ◆ Buildout or license of application and data risk categorization scheme
- ◆ Program overhead from more granular roadmap planning

ADD'L PERSONNEL

- ◆ Architects (2 days/yr)
- ◆ Managers (2 days/yr)
- ◆ Business Owners (2 days/yr)
- ◆ Security Auditor (2 days/yr)

RELATED OBJECTIVES

- ◆ Standards & Compliance - 2
- ◆ Threat Modeling - 2
- ◆ Architecture Review - 2

RELATED COMPLIANCE

- ◆

ACTIVITIES

Classify data and applications based on business risk

Establish a simple classification system to represent risk-tiers for applications. In its simplest form, this can be a High/Medium/Low categorization. More sophisticated classifications can be used, but there should be no more than seven categories and they should roughly represent a gradient from high to low impact against business risks.

Working from the organization's business risk profile, create project evaluation criteria that maps each project to one of the risk categories. A similar but separate classification scheme should be created for data assets and each item should be weighted and categorized based on potential impact to business risks.

Evaluate collected information about each application and assign each a risk category based upon overall evaluation criteria and the risk categories of data assets in use. This can be done centrally by a security group or by individual project teams through a customized questionnaire to gather the requisite information.

An ongoing process for application and data asset risk categorization should be established to assign categories to new assets and keep the existing information updated at least biannually.

Establish per-classification security goals

With a classification scheme for the organization's application portfolio in place, direct security goals and assurance program roadmap choices can be made more granularly.

The assurance program's roadmap should be modified to account for each application risk category by specifying emphasis on particular Functions for each category. For each iteration of the assurance program, this would typically take the form of prioritizing more higher-level Objectives on the highest risk application tier and progressively less stringent Objectives for lower/other categories.

This process establishes the organization's risk tolerance since active decisions must be made as to what specific Objectives are expected of applications in each risk category. By choosing to keep lower risk applications at lower levels of performance with respect to the Security Functions, resources are saved in exchange for acceptance of a weighted risk. However, it is not necessary to arbitrarily build a separate roadmap for each risk category since that can lead to inefficiency in management of the assurance program itself.

Strategic Planning

SP 3

Align security expenditure with relevant business indicators and asset value.

ACTIVITIES

Conduct periodic industry-wide cost comparisons

Research and gather information about security costs from intra-industry communication forums, business analyst and consulting firms, or other external sources. In particular, there are a few key factors that need to be identified.

First, use collected information to identify the average amount of security effort being applied by similar types of organizations in your industry. This can be done either top-down from estimates of total percentage of budget, revenue, etc. or it can be done bottom-up by identifying security-related activities that are considered normal for your type of organization. Overall, this can be hard to gauge for certain industries, so collect information from as many relevant sources as are accessible.

The next goal of researching security costs is to determine if there are potential cost savings on third-party security products and services that your organization currently uses. When weighing the decision of switching vendors, account for hidden costs such as retraining staff or other program overhead.

Overall, these cost-comparison exercises should be conducted at least annually prior to the subsequent assurance program strategy session. Comparison information should be presented to stakeholders in order to better align the assurance program with the business.

Collect metrics for historic security spend

Collect project-specific information on the cost of past security incidents. For instance, time and money spent in cleaning up a breach, monetary loss from system outages, fines and fees to regulatory agencies, project-specific one-off security expenditures for tools or services, etc.

Using the application risk categories and the respective prescribed assurance program roadmaps for each, a baseline security cost for each application can be initially estimated from the costs associated with the corresponding risk category.

Combine the application-specific cost information with the general cost model based on risk category, and then evaluate projects for outliers, i.e. sums disproportionate to the risk rating. These indicate either an error in risk evaluation/classification or the necessity to tune the organization's assurance program to address root causes for security cost more effectively.

The tracking of security spend per project should be done quarterly at the assurance program strategy session, and the information should be reviewed and evaluated by stakeholders at least annually. Outliers and other unforeseen costs should be discussed for potential affect on assurance program roadmap.

RESULTS

- ◆ Information to make informed case-by-case decisions on security expenditures
- ◆ Estimates of past loss due to security issues
- ◆ Per-project consideration of security expense versus loss potential
- ◆ Industry-wide due diligence with regard to security

ADD'L SUCCESS METRICS

- ◆ >80% of projects reporting security costs in past 3 months
- ◆ >1 industry-wide cost comparison in past 1 year
- ◆ >1 historic security spend evaluation in past 1 year

ADD'L COSTS

- ◆ Buildout or license industry intelligence on security programs
- ◆ Program overhead from cost estimation, tracking, and evaluation

ADD'L PERSONNEL

- ◆ Architects (1 days/yr)
- ◆ Managers (1 days/yr)
- ◆ Business Owners (1 days/yr)
- ◆ Security Auditor (1 days/yr)

RELATED OBJECTIVES

- ◆ Vulnerability Management - I

RELATED COMPLIANCE

- ◆

Threat Modeling

The Threat Modeling function is centered on identification and understanding the project-level risks based on the business function of the software being developed. From details about threats and likely attacks against each project, the organization as a whole operates more effectively through better decisions about prioritization of initiatives for security. Additionally, decisions for risk acceptance are more informed, therefore better aligned to the business.

By starting with simple attack trees and building to more detailed methods of threat inspection and weighting, an organization improves over time. Ultimately, a sophisticated organization would maintain this information in a way that's tightly coupled to the compensating factors and passthrough risks from external entities. This provides greater breadth of understanding for potential downstream impacts from security issues while keeping a close watch on the organization's current performance against known threats.

OBJECTIVES	Identify and understand high-level threats to the organization and individual projects.	Increase accuracy of threat assessment and improve granularity of per-project understanding.	Concretely tie compensating controls to each threat against internal and third-party software.
ACTIVITIES	<ul style="list-style-type: none"> ✦ Build and maintain per-application attack trees ✦ Develop attacker profile from software architecture 	<ul style="list-style-type: none"> ✦ Elaborate attack trees with application-specific data ✦ Adopt a weighting system for measurement of threats 	<ul style="list-style-type: none"> ✦ Explicitly evaluate risk from third-party components ✦ Elaborate attack trees with compensating controls
RESULTS	<ul style="list-style-type: none"> ✦ High-level understanding of factors that may lead to negative outcomes ✦ Increased awareness of threats amongst project teams ✦ Inventory of threats for your organization 	<ul style="list-style-type: none"> ✦ Granular understanding of likely threats to individual projects ✦ Framework for better tradeoff decisions within project teams ✦ Ability to prioritize development efforts within a project team based on risk weighting 	<ul style="list-style-type: none"> ✦ Deeper consideration of full threat profile for each software project ✦ Detailed mapping of assurance features to established threats against each software project ✦ Artifacts to document due diligence based on business function of each software project

TM 1

TM 2

TM 3

Threat Modeling

TM

1

Identify and understand high-level threats to the organization and individual projects.

ACTIVITIES

Build and maintain per-application attack trees

Based purely on the business purpose of each software project and the business risk profile (if available) identify likely worst-case scenarios for the software under development in each project team. Identify each worst-case scenario in one sentence and label these as the high-level goals of an attacker.

From each attacker goal identified, identify preconditions that must hold in order for each goal to be realized. This information should be captured in branches underneath each goal where each branch is either a logical AND or a logical OR of the statements contained underneath. An AND branch indicates that each directly attached child nodes must be true in order to realize the parent node. An OR branch indicates that any one of the directly attached child nodes must be true in order to achieve the parent node.

Review each current and, if available, historic functional requirement and augment the attack tree to indicate security failures relevant to each. Brainstorm by iteratively dissecting each node into branches indicating all the possible ways in which an attacker might be able to reach one of the goals. After initial creation, the attack tree for an application should be updated when significant changes to the software are made such that any of the nodes could be eliminated or new nodes added. This assessment should be conducted with senior developers and architects as well as one or more security auditors.

Develop attacker profile from software architecture

Initially, conduct an assessment to identify all likely threats to the organization based on software projects. For this assessment, consider threats to be limited to agents of malicious intent and omit other risks such as known vulnerabilities, potential weaknesses, etc.

Begin by generally considering external agents and their corresponding motivations for attack. To this list, add internal roles that could cause damage and their motivations for insider attack. Based on the architecture of the software project(s) under consideration, it can be more efficient to conduct this analysis once per architecture type instead of for each project individually since applications of architecture and business purpose will generally be susceptible to similar threats.

This assessment should be conducted with business owners and other stakeholders but also include one or more security auditors for additional perspective on threats. In the end, the goal is to have a concise list of threat agents and their corresponding motivations for attack.

RESULTS

- ◆ High-level understanding of factors that may lead to negative outcomes
- ◆ Increased awareness of threats amongst project teams
- ◆ Inventory of threats for your organization

SUCCESS METRICS

- ◆ >50% of project stakeholders briefed on the attack trees of relevant projects within past 12 months
- ◆ >75% of project stakeholders briefed on attacker profiles for relevant architectures

COSTS

- ◆ Buildout and maintenance of project artifacts for attack trees

PERSONNEL

- ◆ Business Owners (1 day/yr)
- ◆ Developers (1 day/yr)
- ◆ Architects (1 day/yr)
- ◆ Security Auditors (2 day/yr)
- ◆ Managers (1 day/yr)

RELATED OBJECTIVES

- ◆ Strategic Planning - 1
- ◆ Security Requirements - 2

RELATED COMPLIANCE

- ◆

Threat Modeling

Increase accuracy of threat assessment and improve granularity of per-project understanding.

RESULTS

- ◆ Granular understanding of likely threats to individual projects
- ◆ Framework for better tradeoff decisions within project teams
- ◆ Ability to prioritize development efforts within a project team based on risk weighting

ADD'L SUCCESS METRICS

- ◆ >75% of project teams with identified and rated threats
- ◆ >75% of project stakeholders briefed on the attack trees of relevant projects within past 6 months
- ◆ >50% of all security incidents already identified by project attack trees within past 12 months

ADD'L COSTS

- ◆ Project overhead from maintenance of attack trees and attacker profiles

ADD'L PERSONNEL

- ◆ Security Auditor (1 day/yr)
- ◆ Business Owner (1 day/yr)
- ◆ Managers (1 day/yr)

RELATED OBJECTIVES

- ◆ Strategic Planning - 2
- ◆ Defensive Design - 2

RELATED COMPLIANCE

- ◆

ACTIVITIES

Elaborate attack trees with application-specific data

From the established attack tree for each software project, elaborate the analysis to include technical details about the software's design and implementation. This should be done by considering several factors about the software. Since attack trees can be lengthy for large applications, split them if necessary, but ensure each piece still contains a copy of its root goal node and connecting nodes along the path.

Begin with the security-relevant assumptions that were considered in the design phase and augment the attack tree with nodes that account for each assumption failing. Based on the security requirements for the project, elaborate nodes on the attack tree that indicate necessary preconditions for failure of each security requirement. Additionally, account for any known security incidents related to the software project to ensure they are represented on the attack tree.

Finally, identify user roles and their corresponding capabilities to further detail insider attack scenarios captured in the attack tree. If available, use application-specific permissions matrices to fuel thinking about failure scenarios such as separation-of-duties concerns.

Adopt weighting system for measurement of threats

Based on the established attacker profiles, identify a rating system to allow relative comparison between the threats. Initially, this can be a simple high-medium-low rating based upon business risk, but any scale can be used provided that there are no more than 5 categories.

After identification of a rating system, build evaluation criteria that allow each threat to be assigned a rating. In order to do this properly, additional factors about each threat must be considered beyond motivation. Important factors include capital and human resources, inherent access privilege, technical ability, relevant goals on the attack tree(s), likelihood of successful attack, etc.

After assigning each threat to a rating, use this information to prioritize risk mitigation activities within the development life-cycle. Once built for a project team, it should be updated during design of new features or refactoring efforts.

Threat Modeling

TM 3

Concretely tie compensating controls to each threat against internal and third-party software.

ACTIVITIES

Explicitly evaluate risk from third-party components

Conduct an assessment of your software code-base and identify any components that are of external origin. Typically, these will include open-source projects, purchased COTS software, and online services which your software uses.

For each identified component, elaborate attacker profiles for the software project based upon potential compromise of third-party components. Based upon the newly identified attacker profiles, update software attack trees to incorporate any likely risks based upon new attacker goals or capabilities.

In addition to threat scenarios, also consider ways in which vulnerabilities or design flaws in the third-party software might affect your code and design. Elaborate your attack trees accordingly with the potential risks from vulnerabilities and knowledge of the updated attacker profile.

After initially conducted for a project, this must be updated and reviewed during the design phase or every development cycle. This activity should be conducted by a security auditor with relevant technical and business stakeholders.

Elaborate attack trees with compensating controls

Conduct an assessment to formally identify factors that directly prevent preconditions for compromise represented by the nodes on the attack tree. These mitigating factors are the compensating controls that formally address the direct risks from software. Factors can be technical features in the software itself, but can also be process elements in the development life-cycle, infrastructure features, etc.

Due to the logical relationships represented on an attack tree, each branch will either represent an AND or an OR. Therefore, by mitigating against just one precondition on an AND branch, the parent and all connected leaf nodes can be marked as mitigated. However, all child nodes on an OR node must be prevented before the parent can be marked as mitigated.

By reviewing the attack trees, identify compensating controls that maximize the number of nodes marked as mitigated. For any viable paths from the leaf nodes to a top-level goal, identify potential compensating controls for feedback into organizational strategy.

After initially conducted for a project, this must be updated and reviewed during the design phase or every development cycle. This activity should be conducted by a security auditor with relevant technical and business stakeholders.

RESULTS

- ◆ Deeper consideration of full threat profile for each software project
- ◆ Detailed mapping of assurance features to established threats against each software project
- ◆ Artifacts to document due diligence based on business function of each software project

ADD'L SUCCESS METRICS

- ◆ >80% of project teams with updated attack trees prior to every implementation cycle
- ◆ >80% of project teams with updated inventory of third-party components prior to every release

ADD'L COSTS

- ◆ Project overhead from maintenance of detailed attack trees and expanded attacker profiles
- ◆ Discovery of all third-party dependencies

ADD'L PERSONNEL

- ◆ Business Owners (1 day/yr)
- ◆ Developers (1 day/yr)
- ◆ Architects (1 day/yr)
- ◆ Security Auditors (2 day/yr)
- ◆ Managers (1 day/yr)

RELATED OBJECTIVES

- ◆ Security Requirements - 2 & 3




RELATED COMPLIANCE

- ◆

Security Requirements

The Security Requirements function is focused on proactively specifying the expected behavior of software with respect to security. Through addition of analysis activities at the project level, security requirements are initially gathered based on the high-level business purpose of the software. As an organization advances, more sophisticated techniques are used such as abuse-case modeling to discover new security requirements that may not have been initially obvious to development.

In a sophisticated form, provision of this function also entails pushing the security requirements of the organization into its relationships with vendors and then auditing projects to ensure all are adhering to expectations with regard to specification of security requirements.

OBJECTIVES	Consider security explicitly during the software requirements process.	Generate security requirements from application-specific risks and abuse scenarios.	Mandate comprehensive security requirements process for all software projects and third-party dependencies.
ACTIVITIES	<ul style="list-style-type: none"> ✦ Derive security requirements from business functionality ✦ Use guidelines, standards, and compliance resources 	<ul style="list-style-type: none"> ✦ Build and maintain abuse-case models per project ✦ Specify security requirements based on known risks 	<ul style="list-style-type: none"> ✦ Build security requirements into vendor agreements ✦ Expand audit program for security requirements
RESULTS	<ul style="list-style-type: none"> ✦ High-level alignment of development effort with business risks ✦ Ad hoc capturing of industry best-practices for security as explicit requirements ✦ Awareness amongst stakeholders of measures being taken to mitigate risk from software 	<ul style="list-style-type: none"> ✦ Detailed understanding of attack scenarios against business logic ✦ Prioritized development effort for security features based on likely attacks ✦ More educated decision-making for tradeoffs between features and security efforts ✦ Stakeholders that can better avoid functional requirements that inherently have security flaws 	<ul style="list-style-type: none"> ✦ Formally set baseline for security expectations from external code ✦ Centralized information on security effort undertaken by each project team ✦ Ability to align resources to projects based on application risk and desired security requirements
			

Security Requirements

SR

1

Consider security explicitly during the software requirements process.

ACTIVITIES

Derive security requirements from business functionality

Conduct a review of functional requirements that specify the business logic and overall behavior for each software project. After gathering requirements for a project, conduct an assessment to derive relevant security requirements. Even if software is being built by a third-party, these requirements, once identified, should be included with functional requirements delivered to vendors.

For each functional requirement, a security auditor should lead stakeholders through the process of explicitly noting any expectations with regard to security. Typically, questions to clarify for each requirement include expectations for data security, access control, transaction integrity, criticality of business function, separation of duties, uptime, etc.

It is important to ensure that all security requirements follow the same principles for writing good requirements in general. Specifically, they should be specific, measurable, and reasonable.

Conduct this process for all new requirements on active projects. For existing features, it is recommended to conduct the same process as a gap analysis to fuel future refactoring for security.

Use guidelines, standards, and compliance resources

Determine industry best-practices that project teams should treat as requirements. These can be chosen from publicly available guidelines, internal or external standards, or established compliance requirements.

It is important to not attempt to bring in too many best-practice requirements into each development iteration since there is a time tradeoff with design and implementation. The recommended approach is to slowly add best-practices over successive development cycles to bolster the software's overall assurance profile over time.

For existing systems, refactoring for security best practices can be a complex undertaking. Where possible, add security requirements opportunistically when adding new features. At a minimum, conducting the analysis to identify applicable best practices should be done to help fuel future planning efforts.

This review should be performed by a security auditor with input from business stakeholders. Senior developers, architects, and other technical stakeholders should also be involved to bring design and implementation-specific knowledge into the decision process.

RESULTS

- ◆ High-level alignment of development effort with business risks
- ◆ Ad hoc capturing of industry best-practices for security as explicit requirements
- ◆ Awareness amongst stakeholders of measures being taken to mitigate risk from software

SUCCESS METRICS

- ◆ >50% of project teams with explicitly defined security requirements

COSTS

- ◆ Project overhead from addition of security requirements to each development cycle

PERSONNEL

- ◆ Security Auditor (2 days/yr)
- ◆ Business Owner (1 days/yr)
- ◆ Managers (1 day/yr)
- ◆ Architects (1 day/yr)

RELATED OBJECTIVES

- ◆ Education & Guidance - 1
- ◆ Standards & Compliance - 2
- ◆ Architecture Review - 1
- ◆ Code Review - 1
- ◆ Security Testing - 1

RELATED COMPLIANCE

- ◆

Security Requirements

Generate security requirements from application-specific risks and abuse scenarios.

RESULTS

- ◆ Detailed understanding of attack scenarios against business logic
- ◆ Prioritized development effort for security features based on likely attacks
- ◆ More educated decision-making for tradeoffs between features and security efforts
- ◆ Stakeholders that can better avoid functional requirements that inherently have security flaws

ADD'L SUCCESS METRICS

- ◆ >75% of all projects with updated abuse-case models within past 6 months

ADD'L COSTS

- ◆ Project overhead from buildout and maintenance of abuse-case models

ADD'L PERSONNEL

- ◆ Security Auditor (2 days/yr)
- ◆ Managers (1 day/yr)
- ◆ Architects (2 days/yr)
- ◆ Business Owners (1 day/yr)

RELATED OBJECTIVES

- ◆ Threat Modeling - 1 & 3
- ◆ Strategic Planning - 1

RELATED COMPLIANCE

- ◆

ACTIVITIES

Build and maintain abuse-case models per project

Further considering the functional requirements of the software project, conduct a more formal analysis to determine potential misuse or abuse of functionality. Typically, this process begins with identification of normal usage scenarios, e.g. use-case diagrams if available.

For each usage scenario, generate a set of abuse-cases by starting with a statement of normal usage and brainstorming ways in which the statement might be negated, in whole or in part. The simplest way to get started is to insert the word “no” or “not” into the usage statement in as many ways as possible, typically around nouns and verbs. Each usage scenario should generate several possible abuse-case statements.

Further elaborate the abuse-case statements to include any application-specific concerns based on the business function of the software. The ultimate goal is for the completed set of abuse statements to form a model for usage patterns that should be disallowed by the software.

After initial creation, abuse-case models should be updated for active projects during the design phase. For existing projects, new requirements should be analyzed for potential abuse, and existing projects should opportunistically build abuse-cases for established functionality where practical.

Specify security requirements based on known risks

Explicitly review existing artifacts that indicate organization or project-specific security risk in order to better understand the overall risk profile for the software. When available, draw on resources such as the high-level business risk profile, individual application attack trees, findings from architecture, code review, security testing, etc.

In addition to review of existing artifacts, use abuse-case models for an application to serve as fuel for identification of concrete security requirements that directly or indirectly mitigate the abuse scenarios.

This process should be conducted by business owners and security auditors as needed. Ultimately, the notion of risks leading to new security requirements should become a built-in step in the planning phase whereby newly discovered risks are specifically assessed by project teams.

Security Requirements

SR 3

Mandate comprehensive security requirements process for all software projects and dependencies.

ACTIVITIES

Build security requirements into vendor agreements

Beyond the kinds of security requirements already identified by previous analysis, additional security benefits can be derived from third-party agreements. Typically, requirements and perhaps high-level design will be developed internally while low-level design and implementation is often left up to vendors.

Based on the specific division of labor for each externally developed component, identify specific security activities and technical assessment criteria to add to the vendor contracts. Commonly, this should be a set of activities from the Verification & Assessment Discipline covering the Architecture Review, Code Review, and Security Testing Functions.

The way in which the externally developed components are used

Modifications of agreement language should be handled on a case-by-case basis with each vendor since adding additional requirements will generally mean an increase in cost. The cost of each potential security activity should be balanced against the benefit of the activity as per the usage of the component or system being considered.

Expand audit program for security requirements

Incorporate checks for completeness of security requirements into routine project audits. Since this can be difficult to gauge without project-specific knowledge, the audit should focus on checking project artifacts such as requirements or design documentation for evidence that the proper types of analysis were conducted.

Particularly, each functional requirement should be annotated with security requirements based on business drivers as well as expected abuse scenarios. The overall project requirements should contain a list of requirements generated from best-practices in guidelines and standards. Additionally, there should be a clear list of unfulfilled security requirements and an estimated timeline for their provision in future releases.

This audit should be performed during every development iteration, ideally toward the end of the requirements process, but it must be performed before a release can be made.

RESULTS

- ◆ Formally set baseline for security expectations from external code
- ◆ Centralized information on security effort undertaken by each project team
- ◆ Ability to align resources to projects based on application risk and desired security requirements

ADD'L SUCCESS METRICS

- ◆ >80% of projects passing security requirements audit in past 6 months
- ◆ >80% of vendor agreements analyzed for contractual security requirements in past 12 months

ADD'L COSTS

- ◆ Increased cost from outsourced development from additional security requirements
- ◆ Ongoing project overhead from release gates for security requirements

ADD'L PERSONNEL

- ◆ Security Auditor (2 days/yr)
- ◆ Managers (2 days/yr)
- ◆ Business Owners (1 day/yr)

RELATED OBJECTIVES

- ◆ Threat Modeling - 3
- ◆ Standards & Compliance - 2

RELATED COMPLIANCE

- ◆

Defensive Design

The Defensive Design function is focused on proactive steps for an organization to build secure software by default. By building software in a way less likely to have downstream vulnerabilities, the overall security risk from software can be dramatically reduced.

Beginning from high-level design guidance based on best practices, an organization evolves toward consistently used design patterns for security functionality. Branching from this, activities also encourage project teams to better understand business logic and create more formal methods for ensure it is designed safely the first time.

As an organization evolves over time, sophisticated provision of this function entails organizations building reference platforms to cover the generic types of software they build. These serve as frameworks upon which developers can build custom software with less risk of vulnerabilities.

OBJECTIVES	Provide proactive design guidance to project teams and make perimeter security enhancements.	Build software with comprehensive security enhancements within the design process.	Measure proactive effort of project teams to make defensive design more efficient.
ACTIVITIES	<ul style="list-style-type: none"> ✦ Maintain list of recommended software frameworks ✦ Explicitly apply security principles to perimeter interfaces 	<ul style="list-style-type: none"> ✦ Build permission matrix for resource access ✦ Identify security design patterns from architecture 	<ul style="list-style-type: none"> ✦ Establish formal reference platforms ✦ Validate usage of frameworks, patterns, and platforms
RESULTS	<ul style="list-style-type: none"> ✦ Ad hoc prevention of unexpected dependencies and one-off implementation choices ✦ Stakeholders aware of increased project risk due to libraries and frameworks chosen ✦ Established protocol within development for proactively applying security mechanisms to a design 	<ul style="list-style-type: none"> ✦ Detailed mapping of assets to user roles to encourage better compartmentalization in design ✦ Reusable design building blocks for provision of security protections and functionality ✦ Increased confidence for software projects from use of established design techniques for security 	<ul style="list-style-type: none"> ✦ Customized application development platforms that provide built-in security protections ✦ Organization-wide expectations for proactive security effort in development ✦ Stakeholders better able to make tradeoff decisions based on business need for secure design

DD **1**

DD **2**

DD **3**

Defensive Design

DD 1

Provide proactive design guidance to project teams and make perimeter security enhancements.

ACTIVITIES

Maintain list of recommended software frameworks

Across software projects within the organization identify commonly used third-party software libraries and frameworks in use. Generally, this need not be an exhaustive search for dependencies, but rather focus on capturing the high-level components that are most often used.

From the list of components, group them into functional categories based on the core features provided by the third-party component. Also, note the usage prevalence of each component across project teams to weight the reliance upon the third-party code. Using this weighted list as a guide, create a list of components to be advertised across the development organization as recommended components.

Several factors should contribute to decisions for inclusion on the recommended list. Although a list can be created without conducting research specifically, it is advisable to inspect each for incident history, track record for responding to vulnerabilities, appropriateness of functionality for the organization, excessive complexity in usage of the third-party component, etc.

This list should be created by senior developers and architects, but also include input from managers and security auditors. After creation, this list of recommended components matched against functional categories should be advertised to the development organization. Ultimately, the goal is to provide well-known defaults for project teams.

Explicitly apply security principles to perimeter interfaces

During design, technical staff on the project team should use a short list of guiding security principles as a checklist against edge interfaces in the system. Typically, security principles include defense in depth, securing the weakest link, use of secure defaults, simplicity in design of security functionality, secure failure, balance of security and usability, running with least privilege, avoidance of security by obscurity, etc.

For each known perimeter interface, the design team should consider each principle in the context of the overall system and identify features that can be added to bolster security at that interface. Generally, these should be limited such that they only take a small amount of extra effort beyond the normal implementation cost of functional requirements and anything larger should be noted and scheduled for future releases.

While this process should be conducted by each project team after being trained with security awareness, it is helpful to incorporate more security-savvy staff to aide in making design decisions.

RESULTS

- ◆ Ad hoc prevention of unexpected dependencies and one-off implementation choices
- ◆ Stakeholders aware of increased project risk due to libraries and frameworks chosen
- ◆ Established protocol within development for proactively applying security mechanisms to a design

SUCCESS METRICS

- ◆ >80% of development staff briefed on software framework recommendations in past 1 year
- ◆ >50% of projects self-reporting application of security principles to design

COSTS

- ◆ Buildout, maintenance, and awareness of software framework recommendations
- ◆ Ongoing project overhead from analysis and application of security principles

PERSONNEL

- ◆ Architects (2-4 days/yr)
- ◆ Developers (2-4 days/yr)
- ◆ Security Auditors (2-4 days/yr)
- ◆ Managers (2 days/yr)

RELATED OBJECTIVES

- ◆ Education & Guidance - I

RELATED COMPLIANCE

- ◆

DD 2

Defensive Design

Build software with comprehensive security enhancements within the design process.

RESULTS

- ◆ Detailed mapping of assets to user roles to encourage better compartmentalization in design
- ◆ Reusable design building blocks for provision of security protections and functionality
- ◆ Increased confidence for software projects from use of established design techniques for security

ADD'L SUCCESS METRICS

- ◆ >80% of projects with updated permission matrix in past 6 months
- ◆ >80% of project teams briefed on applicable security patterns in past 6 months

ADD'L COSTS

- ◆ Buildout or license of applicable security patterns
- ◆ Ongoing project overhead from maintenance of permission matrix

ADD'L PERSONNEL

- ◆ Architects (2-4 days/yr)
- ◆ Developers (1-2 days/yr)
- ◆ Managers (1-2 days/yr)
- ◆ Business Owners (1 day/yr)
- ◆ Security Auditors (1-2 days/yr)

RELATED OBJECTIVES

- ◆ Education & Guidance - 2

RELATED COMPLIANCE

- ◆

ACTIVITIES

Build permission matrix for resource access

Based upon the business purpose of the application, identify user and operator roles. Additionally, build a list of resources by gathering all relevant data assets and application-specific features that are guarded by any form of access control.

In a simple matrix with roles on one axis and resources on the other, consider the relationships between each role and each resource and note in each intersection the correct behavior of the system in terms of access control according to stakeholders.

For data resources, it is important to note access rights in terms of creation, read access, update, and deletion. For resources that are features, gradation of access rights will likely be application-specific, but at a minimum note if the role should be permitted access to the feature.

This permission matrix will serve as an artifact to document the correct access control rights for the business logic of the overall system. As such, it should be created by the project teams with input from business stakeholders. After initial creation, it should be updated by business stakeholders before every release, but usually toward the beginning of the design phase.

Identify security design patterns from architecture

Across software projects at an organization, each should be categorized in terms of the generic architecture type. Common categories include client-server applications, embedded systems, desktop applications, web-facing applications, web services platforms, transactional middleware systems, mainframe applications, etc. Depending on your organization's specialty, more detailed categories may need to be developed based upon language, or processor architecture, or even era of deployment.

For the generic software architecture type, a set of general design patterns representing sound methods of implementing security functionality can be derived and applied to the individual designs of an organization's software projects. These security design patterns represent general definitions of generic design elements they can be researched or purchased, and it is often even more effective if these patterns are customized to be made more specific to your organization. Example patterns include a single-sign-on subsystem, a cross-tier delegation model, a hardened interface design, separation-of-duties authorization model, a centralized logging pattern, etc.

The process of identification of applicable and appropriate patterns should be carried out by architects, senior developers, and other technical stakeholders during the design phase.

Defensive Design

DD 3

Measure proactive effort of project teams to make defensive design more efficient.

ACTIVITIES

Establish formal reference platforms

After working with security patterns specific to each type of architecture, a collection of code implementing these pieces of functionality should be selected from project teams and used as the basis for a shared code-base. This shared code base can initially start as a collection of commonly recommended libraries that each project needs to use and it can grow over time into one or more software frameworks representing reference platforms upon which project teams build their software. Examples of reference platforms include frameworks for model-view-controller web applications, libraries supporting transactional back-end systems, frameworks for web services platforms, scaffolding for client-server applications, frameworks for middleware with pluggable business logic, etc. Another method of building initial reference platforms is to select a particular project early in the life-cycle and have security-savvy staff work with them to build the security functionality in a generic way so that it could be extracted from the project and utilized elsewhere in the organization.

Regardless of approach to creation, reference platforms have advantages in terms of speeding audit and security-related reviews, increasing efficiency in development, and lowering maintenance overhead.

Architects, senior developers and other technical stakeholders should participate in design and creation of reference platforms. After creation, a team must maintain ongoing support and updates.

Validate usage of frameworks, patterns, and platforms

During routine audits of projects conduct additional analysis of project artifacts to measure usage of recommended frameworks, design patterns, and reference platforms. Though conducted during routine audits, the goal of this activity is to collect feedback from project teams as much as to measure their individual proactive security effort.

Overall, it is important to verify several factors with project teams. Identify use of non-recommended frameworks to determine if there may be a gap in recommendations versus the organization's functionality needs. Examine unused or incorrectly used design patterns and reference platform modules to determine if updates are needed. Additionally, there may be more or different functionality that project teams would like to see implemented in the reference platforms as the organization evolves.

This analysis can be conducted by any security-savvy technical staff. Metrics collected from each project should be collated for analysis by managers and stakeholders.

RESULTS

- ◆ Customized application development platforms that provide built-in security protections
- ◆ Organization-wide expectations for proactive security effort in development
- ◆ Stakeholders better able to make tradeoff decisions based on business need for secure design

ADD'L SUCCESS METRICS

- ◆ >50% of active projects using reference platforms
- ◆ >80% of projects reporting framework, pattern, and platform usage feedback in past 6 months
- ◆ >3.0 Likert on usefulness of guidance/platforms reported by project teams

ADD'L COSTS

- ◆ Buildout or license of reference platform(s)
- ◆ Ongoing maintenance and support of reference platforms
- ◆ Ongoing project overhead from usage validation during audit

ADD'L PERSONNEL

- ◆ Managers (1 day/yr)
- ◆ Business Owners (1 day/yr)
- ◆ Architects (3-4 days/yr)
- ◆ Developers (2-3 days/yr)
- ◆ Security Auditors (2 days/yr)

RELATED OBJECTIVES

- ◆ Standards & Compliance - 2
- ◆ Architecture Review - 3
- ◆ Code Review - 3
- ◆ Security Testing - 3

RELATED COMPLIANCE

- ◆

Architecture Review

The Architecture Review function is focused on inspection of software designs and architecture models for security problems. This allows an organization to detect architecture-level problems early in software development and thereby avoid potentially large costs from refactoring down the road.

Beginning with lightweight activities to build understanding of the security-relevant details about an architecture, an organization evolves toward more formal inspection methods that verify completeness in provision of security mechanisms. At the organization level, architecture review services are built and offered to stakeholders.

In a sophisticated form, the architecture review function involves detailed, data-level inspection of designs and enforcement of a standard expectation for assessment findings before releases are accepted.

OBJECTIVES	Support ad hoc reviews of software architecture to ensure baseline mitigations for known risks.	Offer assessment services to review software design against comprehensive best practices for security.	Require assessments and validate artifacts to develop detailed understanding of protection mechanisms.
ACTIVITIES	<ul style="list-style-type: none"> ◆ Identify software attack surface ◆ Analyze design against known security requirements 	<ul style="list-style-type: none"> ◆ Inspect for complete provision of security mechanisms ◆ Deploy design review service for project teams 	<ul style="list-style-type: none"> ◆ Develop data-flow diagrams for sensitive resources ◆ Establish release gates for architecture review
RESULTS	<ul style="list-style-type: none"> ◆ High-level understanding of security implications from perimeter architecture ◆ Enable development teams to self-check designs for security best-practices ◆ Lightweight process for conducting project-level architecture reviews 	<ul style="list-style-type: none"> ◆ Formally offered assessment service to consistently review architecture for security ◆ Pinpoint security flaws in maintenance-mode and legacy systems ◆ Deeper understanding amongst project stakeholders on how the software provides assurance protections 	<ul style="list-style-type: none"> ◆ Granular view of weak points in a system design to encourage better compartmentalization ◆ Organization-level awareness of project standing against baseline security expectations for architecture ◆ Comparisons between projects for efficiency and progress toward mitigating known flaws



Architecture Review

AR 1

Support ad hoc reviews of software architecture to ensure baseline mitigations for known risks.

ACTIVITIES

Identify software attack surface

For each software project, create a simplified view of the overall architecture. Typically, this should be created based on project artifacts such as high-level requirements and design documents, interviews with technical staff, or module-level review of the code base. It is important to capture the high-level modules in the system, but a good rule of thumb for granularity is to ensure that the diagram of the whole system under review fits onto one page.

From the single page architecture view, analyze each component in terms of accessibility of the interfaces from authorized users, anonymous users, operators, application-specific roles, etc. The components providing the interfaces should also be considered in the context of the one-page view to find points of functional delegation or data passthrough to other components on the diagram. Group interfaces and components with similar accessibility profiles and capture this as the software attack surface.

For each interface, further elaborate the one-page diagram to note any security-related functionality. Based on the identified interface groups comprising the attack surface, check the model for design-level consistency for how interfaces with similar access are secured. Any breaks in consistency can be noted as assessment findings

This analysis should be conducted by security-savvy technical staff, either within the project team or external. Typically, after initial creation, the diagram and attack surface analysis only needs to be updated during the design phase when additions or changes are made to the edge system interfaces.

Analyze design against known security requirements

Security requirements, either formally identified or informally known, should be identified and collected. Additionally, identify and include any security assumptions upon which safe operation of the system relies.

Review each item on the list of known security requirements against the one-page diagram of the system architecture. Elaborate the diagram to show the design-level features that address each security requirement. Separate, granular diagrams can be created to simplify capturing this information if the system is large and/or complex. The overall goal is to verify that each known security requirement has been addressed by the system design. Any security requirements that are not clearly provided at the design level should be noted as assessment findings.

This analysis should be conducted by security-savvy technical staff with input from architects, developers, managers, and business owners as needed. It should be updated during the design phase when there are changes in security requirements or high-level system design.

RESULTS

- ◆ High-level understanding of security implications from perimeter architecture
- ◆ Enable development teams to self-check designs for security best-practices
- ◆ Lightweight process for conducting project-level architecture reviews

SUCCESS METRICS

- ◆ >50% of projects with updated attack surface analysis in past 12 months
- ◆ >50% of projects with updated security requirements design-level analysis in past 12 months

COSTS

- ◆ Buildout and maintenance of architecture diagrams for each project
- ◆ Ongoing project overhead from attack surface and security requirement design inspection

PERSONNEL

- ◆ Architects (2-3 days/yr)
- ◆ Developers (1-2 days/yr)
- ◆ Managers (1 day/yr)
- ◆ Security Auditor (1 day/yr)

RELATED OBJECTIVES

- ◆ Security Requirements - 1

RELATED COMPLIANCE

- ◆

Architecture Review

Offer assessment services to review software design against comprehensive best practices for security.

RESULTS

- ◆ Formally offered assessment service to consistently review architecture for security
- ◆ Pinpoint security flaws in maintenance-mode and legacy systems
- ◆ Deeper understanding amongst project stakeholders on how the software provides assurance protections

ADD'L SUCCESS METRICS

- ◆ >80% of stakeholders briefed on status of review requests in past 6 months
- ◆ >75% of projects undergoing architecture review in past 12 months

ADD'L COSTS

- ◆ Buildout, training, and maintenance of design review team
- ◆ Ongoing project overhead from review activities

ADD'L PERSONNEL

- ◆ Architects (1-2 days/yr)
- ◆ Developers (1 day/yr)
- ◆ Managers (1 day/yr)
- ◆ Security Auditors (2-3 days/yr)

RELATED OBJECTIVES

- ◆ Education & Guidance - 2
- ◆ Strategic Planning - 2

RELATED COMPLIANCE

- ◆

ACTIVITIES

Inspect for complete provision of security mechanisms

For each interface on a module in the high-level architecture diagram, formally iterate through the list of security mechanisms and analyze the system for their provision. This type of analysis should be performed on both internal interfaces, e.g. between tiers, as well as external ones, e.g. those comprising the attack surface.

The six main security mechanisms to consider are authentication, authorization, input validation, output encoding, error handling and logging. Where relevant, also consider the mechanisms of cryptography and session management. For each interface, determine where in the system design each mechanism is provided and note any missing or unclear features as findings.

This analysis should be conducted by security-savvy staff with assistance from the project team for application-specific knowledge. This analysis should be performed once per release, usually toward the end of the design phase. After initial analysis, subsequent releases are required to update the findings based on changes being made during the development cycle.

Deploy design review service for project teams

Design a process whereby project stakeholders can request an architecture review. This service may be provided centrally within the organization or distributed across existing staff, but all reviewers must be trained on performing the reviews completely and consistently.

The review service should be centrally managed in that the review request queue should be triaged by senior managers, architects, and stakeholders that are familiar with the overall business risk profile for the organization. This allows prioritization of project reviews in alignment with overall business risk.

During a design review, the review team should work with project teams to collect information sufficient to formulate an understanding of the attack surface, match project-specific security requirements to design elements, and verify security mechanisms at module interfaces.

Architecture Review

AR 3

Require assessments and validate artifacts to develop detailed understanding of protection mechanisms.

ACTIVITIES

Develop data-flow diagrams for sensitive resources

Based on the business function of the software project, conduct analysis to identify details on system behavior around high-risk functionality. Typically, high-risk functionality will correlate to features implementing creation, access, update, and deletion of sensitive data. Beyond data, high-risk functionality also includes project-specific business logic that is critical in nature, either from a denial-of-service or compromise perspective.

For each identified data source or business function, select and use a standardized notation to capture relevant software modules, data sources, actors, and messages that flow amongst them. It is often helpful to start with a high-level design diagram and iteratively flesh out relevant detail while removing elements that do not correspond to the sensitive resource.

With data-flow diagrams created for a project, conduct analysis over them to determine internal choke-points in the design. Generally, these will be individual software modules that handle data with differing sensitivity levels or those that gate access to several business functions of various levels of business criticality.

Establish release gates for architecture review

Having established a consistent architecture review program, the next step of enforcement is to set a particular point in the software development life-cycle where a project cannot pass until an architecture review is conducted and findings are reviewed and accepted. In order to accomplish this, a baseline level of expectations should be set, e.g. no projects with any high-severity findings will be allowed to pass and all other findings must be accepted by the business owner.

Generally, architecture reviews should occur toward the end of the design phase to aide early detection of security issues, but it must occur before releases can be made from the project team.

For legacy systems or inactive projects, an exception process should be created to allow those projects to continue operations, but with an explicitly assigned timeframe for each to be reviewed to illuminate any hidden vulnerabilities in the existing systems. Exceptions for should be limited to no more than 20% of all projects.

RESULTS

- ◆ Granular view of weak points in a system design to encourage better compartmentalization
- ◆ Organization-level awareness of project standing against baseline security expectations for architecture
- ◆ Comparisons between projects for efficiency and progress toward mitigating known flaws

ADD'L SUCCESS METRICS

- ◆ >80% of projects with updated data-flow diagrams in past 6 months
- ◆ >75% of projects passing architecture review audit in past 6 months

ADD'L COSTS

- ◆ Ongoing project overhead from maintenance of data-flow diagrams
- ◆ Organization overhead from project delays caused by failed architecture review audits

ADD'L PERSONNEL

- ◆ Developers (2 days/yr)
- ◆ Architects (1 day/yr)
- ◆ Managers (1-2 days/yr)
- ◆ Business Owners (1-2 days/yr)
- ◆ Security Auditors (2-3 days/yr)

RELATED OBJECTIVES

- ◆ Defensive Design - 3
- ◆ Code Review - 3

RELATED COMPLIANCE

- ◆

Code Review

The Code Review function is focused on inspection of software at the source code level in order to find security vulnerabilities. Code-level vulnerabilities are generally simple to understand conceptually, but even informed developers can easily make mistakes that leave software open to potential compromise.

To begin, an organization uses lightweight checklists and for efficiency, only inspects the most critical software modules. However, as an organization evolves it uses automation technology to dramatically improve coverage and efficacy of code review activities.

Sophisticated provision of this function involves deeper integration of code review into the development process to enable project teams to find problems earlier. This also enables organizations to better audit and set expectations for code review findings before releases can be made.

OBJECTIVES	Opportunistically find basic code-level vulnerabilities and other high-risk security issues.	Make code review during development more accurate and efficient through automation.	Mandate comprehensive code review process to discover both language-level and application-specific risks.
ACTIVITIES	<ul style="list-style-type: none"> ◆ Create review checklists from known security requirements ◆ Perform point-review of high-risk code 	<ul style="list-style-type: none"> ◆ Utilize automated code analysis tools ◆ Integrate code analysis into development process 	<ul style="list-style-type: none"> ◆ Customize code analysis for application-specific concerns ◆ Establish release gates for code review
RESULTS	<ul style="list-style-type: none"> ◆ Inspection for common code vulnerabilities that lead to likely discovery or attack ◆ Lightweight review for coding errors that lead to severe security impact ◆ Basic code-level due diligence for security assurance 	<ul style="list-style-type: none"> ◆ Development enabled to consistently self-check for code-level security vulnerabilities ◆ Routine analysis results to compile historic data on per-team secure coding habits ◆ Stakeholders aware of unmitigated vulnerabilities to support better tradeoff analysis 	<ul style="list-style-type: none"> ◆ Increased confidence in accuracy and applicability of code analysis results ◆ Organization-wide baseline for secure coding expectations ◆ Project teams with an objective goal for judging code-level security

CR **1**

CR **2**

CR **3**

Code Review

CR 1

Opportunistically find basic code-level vulnerabilities and other high-risk security issues.

ACTIVITIES

Create review checklists from known security requirements

From the known security requirements for a project, derive a lightweight code review checklist for security. These can be checks specific to the security concerns surrounding the functional requirements or checks for secure coding best practices based on the implementation language, platform, typical technology stack, etc. Due to these variations, often a set of checklist are needed to cover the different types of software development within an organization.

Regardless, of whether created from publicly available resources or purchased, technical stakeholders such as development managers, architects, developers, and security auditors should review the checklists for efficacy and feasibility. It is important to keep the lists short and simple, aiming to catch high-priority issues that are straightforward to find in code either manually or with simple search tools. Code analysis automation tools may also be used to achieve this same end, but should also be customized to reduce the overall set of security checks to a small, valuable set in order to make the scan and review process efficient.

Developers should be briefed on the goals of checklists appropriate to their job function.

Perform point-review of high-risk code

Since code-level vulnerabilities can have dramatically increased impacts if they occur in security-critical parts of software, project teams should review high-risk modules for common vulnerabilities. Common examples of high-risk functionality include authentication modules, access control enforcement points, session management schemes, external interfaces, input validators and data parsers, etc.

Utilizing the code review checklists, the analysis can be performed as a normal part of the development process where members of the project team are assigned modules to review when changes are made. Security auditors and automated review tools can also be utilized for the review.

During development cycles where high-risk code is being changed and reviewed, development managers should triage the findings and prioritize remediation appropriately with input from other project stakeholders.

RESULTS

- ◆ Inspection for common code vulnerabilities that lead to likely discovery or attack
- ◆ Lightweight review for coding errors that lead to severe security impact
- ◆ Basic code-level due diligence for security assurance

SUCCESS METRICS

- ◆ >80% of project teams briefed on relevant code review checklists in past 6 months
- ◆ >50% of project teams performing code review on high-risk code in past 6 months
- ◆ >3.0 Likert on usefulness of code review checklists reported by developers

COSTS

- ◆ Buildout or license of code review checklists
- ◆ Ongoing project overhead from code review activities of high-risk code

PERSONNEL

- ◆ Developers (2-4 days/yr)
- ◆ Architects (1-2 days/yr)
- ◆ Managers (1-2 days/yr)
- ◆ Business Owners (1 day/yr)

RELATED OBJECTIVES

- ◆ Security Requirements - I

RELATED COMPLIANCE

- ◆

Code Review

Make code review during development more accurate and efficient through automation.

RESULTS

- ◆ Development enabled to consistently self-check for code-level security vulnerabilities
- ◆ Routine analysis results to compile historic data on per-team secure coding habits
- ◆ Stakeholders aware of unmitigated vulnerabilities to support better tradeoff analysis

ADD'L SUCCESS METRICS

- ◆ >50% of projects with code review and stakeholder sign-off in past 6 months
- ◆ >80% of projects with access to automated code review results in past 1 month

ADD'L COSTS

- ◆ Research and selection of code analysis solution
- ◆ Initial cost and maintenance of automation integration
- ◆ Ongoing project overhead from automated code review and mitigation

ADD'L PERSONNEL

- ◆ Developers (1-2 days/yr)
- ◆ Architects (1 day/yr)
- ◆ Managers (1-2 days/yr)
- ◆ Security Auditors (3-4 days/yr)

RELATED OBJECTIVES

◆

RELATED COMPLIANCE

◆

ACTIVITIES

Utilize automated code analysis tools

Many security vulnerabilities at the code level are complex to understand and require careful inspection for discovery. However, there are many useful automation solutions available to automatically analyze code for bugs and vulnerabilities.

There are both commercial and open-source products available to cover popular programming languages and frameworks. Selection of an appropriate code analysis solution is based on several factors including depth and accuracy of inspection, product usability and usage model, expandability and customization features, applicability to the organization's architecture and technology stack(s), etc.

Utilize input from security-savvy technical staff as well as developers and development managers in the selection process, and review overall results with stakeholders.

Integrate code analysis into development process

Once a code analysis solution is selected, it must be integrated into the development process to encourage project teams to utilize its capabilities. An effective way to accomplish this is to setup the infrastructure for the scans to run automatically at build time or from code in the project's code repository. In this fashion, results are available earlier thus enabling development teams to self-check along the way before release.

A potential problem with legacy systems or large ongoing projects is that code scanners will typically report findings in modules that were not being updated in the release. If automatic scanning is setup to run periodically, an effective strategy to avoid review overhead is to limit consideration of findings to those that have been added, removed, or changed since the previous scan. If is critical to not ignore the rest of the results however, so development managers should take input from security auditors, stakeholders, and the project team to formulate a concrete plan for addressing the rest of the findings.

If unaddressed findings from code review remain at release, these must be reviewed and accepted by project stakeholders.

Code Review

CR 3

Mandate comprehensive code review process to discover both language-level and application-specific risks.

ACTIVITIES

Customize code analysis for application-specific concerns

Code scanning tools are powered by built-in a knowledge-base of rules to check code based on language APIs and commonly used libraries, but have limited ability to understand custom APIs and designs to apply analogous checks. However, through customization, a code scanner can be a powerful, generic analysis engine for finding organization and project-specific security concerns.

While details vary between tools in terms of ease and power of custom analysis, code scanner customization generally involves specifying checks to be performed at specific APIs and function call sites. Checks can include analysis for adherence to internal coding standards, unchecked tainted data being passed to custom interfaces, tracking and verification of sensitive data handling, correct usage of an internal API, etc.

Checkers for usage of shared code-bases are an effective place to begin scanner customizations since the created checkers can be utilized across multiple projects. To customize a tool for a code-base, a security auditor should inspect both code and high-level design to identify candidate checkers to discuss with development staff and stakeholders for implementation.

Establish release gates for code review

To set a code-level security baseline for all software projects, a particular point in the software development life-cycle should be established as a checkpoint where a minimum standard for code review results must be met in order to make a release.

To begin, this standard should be straightforward to meet, for example by choosing one or two vulnerability types and a setting the standard that no project may pass with any corresponding findings. Over time, this baseline standard should be improved by adding additional criteria for passing the checkpoint.

Generally, the code review checkpoint should occur toward the end of the implementation phase, but must occur before release.

For legacy systems or inactive projects, an exception process should be created to allow those projects to continue operations, but with an explicitly assigned timeframe for mitigation of findings. Exceptions should be limited to no more than 20% of all projects.

RESULTS

- ◆ Increased confidence in accuracy and applicability of code analysis results
- ◆ Organization-wide baseline for secure coding expectations
- ◆ Project teams with an objective goal for judging code-level security

ADD'L SUCCESS METRICS

- ◆ >50% of projects using code analysis customizations
- ◆ >75% of projects passing code review audit in past 6 months

ADD'L COSTS

- ◆ Buildout and maintenance of custom code review checks
- ◆ Ongoing project overhead from code review audit
- ◆ Organization overhead from project delays caused by failed code review audits

ADD'L PERSONNEL

- ◆ Architects (1 day/yr)
- ◆ Developers (1 day/yr)
- ◆ Security Auditors (1-2 days/yr)
- ◆ Business Owners (1 day/yr)
- ◆ Managers (1 day/yr)

RELATED OBJECTIVES

- ◆ Standards & Compliance - 2
- ◆ Defensive Design - 3

RELATED COMPLIANCE

- ◆

Security Testing

The Security Testing function is focused on inspection of software while operating in the runtime environment in order to find security problems. These testing activities bolster the assurance case for software by checking it in the same context in which it is expected to run, thus making visible operational misconfigurations or errors in business logic that are difficult to otherwise find.

Starting with specification of high-level test cases based on the basic functionality of software, an organization evolves toward usage of security testing automation to cover the wide variety of test cases that might demonstrate a vulnerability in the system.

In an advanced form, provision of this function involves customization of testing automation to build a battery of security tests covering application-specific concerns in detail. With additional visibility at the organization level, security testing enables organizations to set minimum expectations for security testing results before a project release is accepted.

OBJECTIVES	Enable testing process to perform basic security tests based on software requirements.	Make security testing during development more complete and efficient through automation.	Require application-specific security testing to ensure baseline security before deployment.
ACTIVITIES	<ul style="list-style-type: none"> ◆ Derive test cases from known security requirements ◆ Explicitly evaluate known security test-cases 	<ul style="list-style-type: none"> ◆ Utilize automated security testing tools ◆ Integrate security testing into development process 	<ul style="list-style-type: none"> ◆ Employ application-specific security testing automation ◆ Establish release gates for security testing
RESULTS	<ul style="list-style-type: none"> ◆ Independent verification of expected security mechanisms surrounding critical business functions ◆ High-level due diligence toward security testing ◆ Ad hoc growth of a security test suite for each software project 	<ul style="list-style-type: none"> ◆ Deeper and more consistent verification of software functionality for security ◆ Development teams enabled to self-check and correct problems before release ◆ Stakeholders better aware of open vulnerabilities when making risk acceptance decisions 	<ul style="list-style-type: none"> ◆ Organization-wide baseline for expected application performance against attacks ◆ Customized security test suites to improve accuracy of automated analysis ◆ Project teams aware of objective goals for attack resistance

ST **1**

ST **2**

ST **3**

Security Testing

ST

1

Enable testing process to perform basic security tests based on software requirements.

ACTIVITIES

Derive test cases from known security requirements

From the known security requirements for a project, identify a set of test cases to check the software for correct functionality. Typically, these test cases are derived from security concerns surrounding the functional requirements and business logic of the system, but should also include generic tests for common vulnerabilities based on the implementation language or technology stack.

Often, it is most effective to use the project team's time to build application-specific test cases and utilize publicly available resources or purchased knowledge bases to select applicable general test cases for security. Although not required, automated security testing tools can also be utilized to cover the general security test cases.

This test case planning should occur during the requirements and/or design phases, but must occur before final testing prior to release. Candidate test cases should be reviewed for applicability, efficacy, and feasibility by relevant development, security, and quality assurance staff.

Explicitly evaluate known security test-cases

Using the set of security test cases identified for each project, testing should be conducted to evaluate the system's performance against each case. It is common for this to occur during the testing phase prior to release.

Once specified, security test cases can be executed by security-savvy quality assurance or development staff, but first-time execution of security test cases for a project team should be monitored by a security auditor to assist and coach team members.

Prior to release or deployment, stakeholders must review results of security tests and accept the risks indicated by failing security tests at release time. In the latter case, a concrete timeline should be established to address the gaps over time.

RESULTS

- ◆ Independent verification of expected security mechanisms surrounding critical business functions
- ◆ High-level due diligence toward security testing
- ◆ Ad hoc growth of a security test suite for each software project

SUCCESS METRICS

- ◆ >50% of projects specifying security test cases in past 12 months
- ◆ >50% of stakeholders briefed on project status against security tests in past 6 months

COSTS

- ◆ Buildout or license of security test cases
- ◆ Ongoing project overhead from maintenance and evaluation of security test cases

PERSONNEL

- ◆ QA Testers (1-2 days/yr)
- ◆ Security Auditor (1-2 days/yr)
- ◆ Developers (1 day/yr)
- ◆ Architects (1 day/yr)
- ◆ Business Owners (1 day/yr)

RELATED OBJECTIVES

- ◆ Security Requirements - I

RELATED COMPLIANCE

- ◆

Security Testing

Make security testing during development more complete and efficient through automation.

RESULTS

- ◆ Deeper and more consistent verification of software functionality for security
- ◆ Development teams enabled to self-check and correct problems before release
- ◆ Stakeholders better aware of open vulnerabilities when making risk acceptance decisions

ADD'L SUCCESS METRICS

- ◆ >50% of projects with security testing and stakeholder sign-off in past 6 months
- ◆ >80% of projects with access to automated security testing results in past 1 month

ADD'L COSTS

- ◆ Research and selection of automated security testing solution
- ◆ Initial cost and maintenance of automation integration
- ◆ Ongoing project overhead from automated security testing and mitigation

ADD'L PERSONNEL

- ◆ Developers (1 days/yr)
- ◆ Architects (1 day/yr)
- ◆ Managers (1-2 days/yr)
- ◆ Security Auditors (2 days/yr)
- ◆ QA Testers (3-4 days/yr)

RELATED OBJECTIVES

◆

RELATED COMPLIANCE

◆

ACTIVITIES

Utilize automated security testing tools

In order to test for security issues, a potentially large number of input cases must be checked against each software interface, which can make effective security testing using manual test case implementation and execution unwieldy. Thus, automated security test tools should be used to automatically test software, resulting in more efficient security testing and higher quality results.

Both commercial and open-source products are available and should be reviewed for appropriateness for the organization. Selecting a suitable tool is based on several factors including robustness and accuracy of built-in security test cases, efficacy at testing architecture types important to organization, customization to change or add test cases, quality and usability of findings to the development organization, etc..

Utilize input from security-savvy technical staff as well as development and quality assurance staff in the selection process, and review overall results with stakeholders.

Integrate security testing into development process

With tools to run automated security tests, projects within the organization should routinely run security tests and review results during development. In order to make this scalable with low overhead, security testing tools should be configured to automatically run on a routine basis, e.g. nightly or weekly, and findings should be inspected as they occur.

Conducting security tests as early as the requirements or design phases can be beneficial. While traditionally, used for functional test cases, this type of test-driven development approach involves identifying and running relevant security test cases early in the development cycle, usually during design. With the automatic execution of security test cases, projects enter the implementation phase with a number of failing tests for the non-existent functionality. Implementation is complete when all the tests pass. This provides a clear, upfront goal for developers early in the development cycle, thus lowering risk of release delays due to security concerns or forced acceptance of risk in order to meet project deadlines.

For each project release, results from automated and manual security tests should be presented to management and business stakeholders for review. If there are unaddressed findings that remain as accepted risks for the release, stakeholders and development managers should work together to establish a concrete timeframe for addressing them.

Security Testing

ST 3

Require application-specific security testing to ensure baseline security before deployment.

ACTIVITIES

Employ application-specific security testing automation

Through either customization of security testing tools, enhancements to generic test case execution tools, or buildout of custom test harnesses, project teams should formally iterate through security requirements and build a set of automated checkers to test the security of the implemented business logic.

Additionally, many automated security testing tools can be greatly improved in accuracy and depth of coverage if they are customized to understand more detail about the specific software interfaces in the project under test. Further, organization-specific concerns from compliance or technical standards can be codified as a reusable, central test battery to make audit data collection and per-project management visibility simpler.

Project teams should focus on buildout of granular security test cases based on the business functionality of their software, and an organization-level team led by a security auditor should focus on specification of automated tests for compliance and internal standards.

Establish release gates for security testing

To prevent software from being released with easily found security bugs, a particular point in the software development life-cycle should be identified as a checkpoint where an established set of security test cases must pass in order to make a release from the project. This establishes a baseline for the kinds of security tests all projects are expected to pass.

Since adding too many test cases initially can result in an overhead cost bubble, begin by choosing one or two security issues and include a wide variety of test cases for each with the expectation that no project may pass if any test fails. Over time, this baseline should be improved by selecting additional security issues and adding a variety of corresponding test cases.

Generally, this security testing checkpoint should occur toward the end of the implementation or testing, but must occur before release.

For legacy systems or inactive projects, an exception process should be created to allow those projects to continue operations, but with an explicitly assigned timeframe for mitigation of findings. Exceptions should be limited to no more than 20% of all projects.

RESULTS

- ◆ Organization-wide baseline for expected application performance against attacks
- ◆ Customized security test suites to improve accuracy of automated analysis
- ◆ Project teams aware of objective goals for attack resistance

ADD'L SUCCESS METRICS

- ◆ >50% of projects using security testing customizations
- ◆ >75% of projects passing all security tests in past 6 months

ADD'L COSTS

- ◆ Buildout and maintenance of customizations to security testing automation
- ◆ Ongoing project overhead from security testing audit process
- ◆ Organization overhead from project delays caused by failed security testing audits

ADD'L PERSONNEL

- ◆ Architects (1 day/yr)
- ◆ Developers (1 day/yr)
- ◆ Security Auditors (1-2 days/yr)
- ◆ QA Testers (1-2 days/yr)
- ◆ Business Owners (1 day/yr)
- ◆ Managers (1 day/yr)

RELATED OBJECTIVES

- ◆ Standards & Compliance - 2
- ◆ Defensive Design - 3

RELATED COMPLIANCE




- ◆

Vulnerability Management

The Vulnerability Management function is focused on the processes within an organization with respect to handling vulnerability reports and operational incidents. By having processes in place, these events that often lead to chaotic and uninformed responses will instead allow projects within the organization to have consistent expectations and increased efficiency in response.

Starting from lightweight assignment of roles in the event of an incident, an organization grows into a more formal incident response process that ensures visibility and tracking on issues that occur. Communications are also improved to improve overall understanding of the processes.

In an advanced form, vulnerability management involves thorough dissecting of incidents and vulnerability reports to collect detailed metrics to feedback into the organization's downstream behavior.

OBJECTIVES	Understand high-level plan for responding to vulnerability reports or incidents.	Elaborate expectations for response process to improve consistency and communications.	Improve analysis and data gathering within response process for feedback into proactive planning.
ACTIVITIES	<ul style="list-style-type: none"> ✦ Identify point of contact for security issues ✦ Create informal security response team(s) 	<ul style="list-style-type: none"> ✦ Establish consistent incident response process ✦ Adopt a security issue disclosure process 	<ul style="list-style-type: none"> ✦ Conduct root cause analysis for incidents ✦ Collect per-incident metrics
RESULTS	<ul style="list-style-type: none"> ✦ Lightweight process in place to handle high-priority vulnerabilities or incidents ✦ Framework for stakeholder notification and reporting of events with security impact ✦ High-level due diligence for handling security issues 	<ul style="list-style-type: none"> ✦ Communications plan for dealing with vulnerability reports from third-parties ✦ Clear process for releasing security patches to software operators ✦ Formal process for tracking, handling, and internally communicating about incidents 	<ul style="list-style-type: none"> ✦ Detailed feedback for organizational improvement after each incident ✦ Rough cost estimation from vulnerabilities and compromises ✦ Stakeholders better able to make tradeoff decisions based on historic incident trends
			

Vulnerability Management

VM

1

Understand high-level plan for responding to vulnerability reports or incidents.

ACTIVITIES

Identify point of contact for security issues

For each division within the organization or for each project team, establish a point of contact to serve as a communications hub for security information. While generally this responsibility will not claim much time from the individuals, the purpose of having a predetermined point of contact is to add structure and governance for vulnerability management.

Examples of incidents that might cause the utilization include receipt of a vulnerability report from an external entity, compromise or other security failure of software in the field, internal discovery of high-risk vulnerabilities, etc. In case of an event, the closest contact would step in as an extra resource and advisor to the affected project team(s) to provide technical guidance and brief other stakeholders on progress of mitigation efforts.

The point of contact should be chosen from security-savvy technical or management staff with a breadth of knowledge over the software projects in the organization. A list of these assigned security points of contact should be centrally maintained and updated at least every six months. Additionally, publishing and advertising this list allows staff within the organization to request help and work directly with one another on security problems.

Create informal security response team(s)

From the list of individuals assigned responsibility as a security point of contact or from dedicated security personnel, select a small group to serve as a centralized technical security response team. The responsibilities of the team will include directly taking ownership of security incidents or vulnerability reports and being responsible for triage, mitigation, and reporting to stakeholders.

Given their responsibility when tapped, members of the security response team are also responsible for executive briefings and upward communication during an incident. It is likely that most of the time, the security response team would not be operating in this capacity, though they must be flexible enough to be able to respond quickly or a smooth process must exist for deferring and incident to another team member.

The response team should hold a meeting at least annually to brief security points of contact on the response process and high-level expectations for security-related reporting from project teams.

RESULTS

- ◆ Lightweight process in place to handle high-priority vulnerabilities or incidents
- ◆ Framework for stakeholder notification and reporting of events with security impact
- ◆ High-level due diligence for handling security issues

SUCCESS METRICS

- ◆ >50% of the organization briefed on closest security point of contact in past 6 months
- ◆ >1 meeting of security response team and points of contact in past 12 months

COSTS

- ◆ Ongoing variable project overhead from staff filling the security point of contact roles
- ◆ Identification of appropriate security response team

PERSONNEL

- ◆ Security Auditors (1 day/yr)
- ◆ Architects (1 day/yr)
- ◆ Managers (1 day/yr)
- ◆ Business Owners (1 day/yr)

RELATED OBJECTIVES

- ◆ Education & Guidance - 2
- ◆ Strategic Planning - 3

RELATED COMPLIANCE

- ◆

Vulnerability Management

Elaborate expectations for response process to improve consistency and communications.

RESULTS

- ◆ Communications plan for dealing with vulnerability reports from third-parties
- ◆ Clear process for releasing security patches to software operators
- ◆ Formal process for tracking, handling, and internally communicating about incidents

ADD'L SUCCESS METRICS

- ◆ >80% of project teams briefed on incident response process in past 6 months
- ◆ >80% of stakeholders briefed on security issue disclosures in past 6 months

ADD'L COSTS

- ◆ Ongoing organization overhead from incident response process

ADD'L PERSONNEL

- ◆ Security Auditors (3-5 days/yr)
- ◆ Managers (1-2 days/yr)
- ◆ Business Owners (1-2 days/yr)
- ◆ Support/Operators (1-2 days/yr)

RELATED OBJECTIVES

- ◆

RELATED COMPLIANCE

- ◆

ACTIVITIES

Establish consistent incident response process

Extending from the informal security response team, explicitly document the organization's incident response process as well as the procedures that team members are expected to follow. Additionally, each member of the security response team must be trained on this material at least annually.

There are several tenets to sound incident response process and they include initial triage to prevent additional damage, change management and patch application, managing project personnel and others involved in the incident, forensic evidence collection and preservation, limiting communication about the incident to stakeholders, well-defined reporting to stakeholders and/or communications trees, etc.

With development teams, the security responders should work together to conduct the technical analysis to verify facts and assumptions about each incident or vulnerability report. Likewise, when project teams detect an incident or high-risk vulnerability, they should follow an internal process that puts them in contact with a member of the security response team.

Adopt a security issue disclosure process

For most organizations, it is undesirable to let news of a security problem become public, but there are several important ways in which internal-to-external communications on security issues should be fulfilled.

The first and most common is through creation and deployment of security patches for the software produced by the organization. Generally, if all software projects are only used internally, then this becomes less critical, but for all contexts where the software is being operated by parties external to the organization, a patch release process must exist. It should provide for several factors including change management and regression testing prior to patch release, announcement to operators/users with assigned criticality category for the patch, sparse technical details so that an exploit cannot be directly derived, etc.

Another avenue for external communications is with third parties that report security vulnerabilities in an organization's software. By adopting and externally posting the expected process with timeframes for response, vulnerability reporters are encouraged to follow responsible disclosure practices.

Lastly, many states and countries legally require external communications for incidents involving data theft of personally identifiable information and other sensitive data type. Should this type of incident occur, the security response team should work with managers and business stakeholders to determine appropriate next-steps.

Vulnerability Management

VM 3

Improve analysis and data gathering within response process for feedback into proactive planning.

ACTIVITIES

Conduct root cause analysis for incidents

Though potentially time consuming, the incident response process should be augmented to include additional analysis to identify the key, underlying security failures. These root causes can be technical problems such as code-level vulnerabilities, configuration errors, etc. or they can be people/process problems such as social engineering, failure to follow procedures, etc.

Once a root cause is identified for an incident, it should be used as a tool to find other potential weaknesses in the organization where an analogous incident could have occurred. For each identified weakness additional recommendations for proactive mitigations should be communicated as part of closing out the original incident response effort.

Any recommendations based on root cause analysis should be reviewed by management and relevant business stakeholders in order to either schedule mitigation activities or note the accepted risks.

Collect per-incident metrics

By having a centralized process to handle all compromise and high-priority vulnerability reports, an organization is enabled to take measurements of trends over time to determine impact and efficiency of initiatives for security assurance.

Records of past incidents should be stored and reviewed at least every 6 months. Group similar incidents and simply tally the overall count for each type of problem. Additional measurements to take from the incidents include frequency of software projects affected by incidents, system downtime and cost from loss of use, human resources taken in handling and cleanup of the incident, estimates of long-term costs such as regulatory fines or brand damage, etc. For root causes that were technical problems in nature, it is also helpful to identify what kind of proactive, review, or operational practice might have detected it earlier or lessened the damage.

This information is concrete feedback into the program planning process since it represents the real security impact that the organization has felt over time.

RESULTS

- ◆ Detailed feedback for organizational improvement after each incident
- ◆ Rough cost estimation from vulnerabilities and compromises
- ◆ Stakeholders better able to make tradeoff decisions based on historic incident trends

ADD'L SUCCESS METRICS

- ◆ >80% of incidents documented with root causes and further recommendations in past 6 months
- ◆ >80% of incidents collated for metrics in the past 6 months

ADD'L COSTS

- ◆ Ongoing organization overhead from conducting deeper research and analysis of incidents
- ◆ Ongoing organization overhead from collection and review of incident metrics

ADD'L PERSONNEL

- ◆ Security Auditors (3 days/yr)
- ◆ Managers (2 days/yr)
- ◆ Business Owners (2 days/yr)

RELATED OBJECTIVES

◆

RELATED COMPLIANCE

◆

Infrastructure Hardening

The Infrastructure Hardening function is focused on bolstering the runtime environment the hosts an organization's software. Since secure operations of software can be deteriorated by problems in external components, hardening the underlying infrastructure directly improves the overall security posture of the organization's software.

By starting with simple tracking and distributing of information about the operating environment to keep development teams better informed, and organization evolves to scalable methods for managing deployment of security patches and instrumenting the operating environment with early-warning detectors for potential security issues before damage is done.

As an organization advances, the operating environment is further reviewed and hardened by deployment of protection tools to add layers of defenses and safety nets to limit damage in case any vulnerabilities are exploited.

OBJECTIVES	Understand baseline operational environment for applications and software components.	Improve confidence in application operations by hardening the operating environment.	Validate application health and status of operational environment against known best practices.
ACTIVITIES	<ul style="list-style-type: none"> ✦ Maintain operational environment specification ✦ Identify and install critical security patches 	<ul style="list-style-type: none"> ✦ Establish infrastructure patch management process ✦ Monitor baseline infrastructure configuration status 	<ul style="list-style-type: none"> ✦ Identify and deploy relevant operations protection tools ✦ Expand audit program for infrastructure configuration
RESULTS	<ul style="list-style-type: none"> ✦ Clear understanding of operational expectations within the development team ✦ High-priority risks from underlying infrastructure mitigated on a well-understood timeline ✦ Software operators with a high-level plan for security-critical maintenance of infrastructure 	<ul style="list-style-type: none"> ✦ Granular verification of security characteristics of systems in operations ✦ Formal expectations on timelines for infrastructure risk mitigation ✦ Stakeholders consistently aware of current operations status of software projects 	<ul style="list-style-type: none"> ✦ Reinforced operational environment with layered checks for security ✦ Established and measured goals for operational maintenance and performance ✦ Reduced likelihood of successful attack via flaws in external dependencies

IH **1**

IH **2**

IH **3**

Infrastructure Hardening

IH 1

Understand baseline operational environment for applications and software components.

ACTIVITIES

Maintain operational environment specification

For each project, a concrete definition of the expected operating platforms should be created and maintained. Depending on the organization, this specification should be jointly created with development staff, stakeholders, support and operations groups, etc.

Begin this specification should by capturing all details that must be true about the operating environment based upon the business function of the software. These can include factors such as processor architecture, operating system versions, prerequisite software, conflicting software, etc. Further, note any known user or operator configurable options about the operating environment that affect the way in which the software will behave.

Additionally, identify any relevant assumptions about the operating environment that were made in design and implementation of the project and capture those assumptions in the specification.

This specification should be reviewed and updated at least every 6 months for active projects or more often if changes are being made to the software design or the expected operating environment.

Identify and install critical security patches

Most applications are software that runs on top of another large stack of software composed of built-in programming language libraries, third-party components and development frameworks, base operating systems, etc. Because security flaws contained in any module in that large software stack affect the overall security of the organization's software, critical security updates for elements of the technology stack must be installed.

As such, regular research or ongoing monitoring of high-risk dependencies should be performed to stay abreast of the latest fixes to security flaws. Upon identification of a critical patch that would impact the security posture of the software project, plans should be made to get affected users and operators to update their installations. Depending on the type of software project, details on doing this can vary.

RESULTS

- ◆ Clear understanding of operational expectations within the development team
- ◆ High-priority risks from underlying infrastructure mitigated on a well-understood timeline
- ◆ Software operators with a high-level plan for security-critical maintenance of infrastructure

SUCCESS METRICS

- ◆ >50% project with updated operational environment specification in past 6 months
- ◆ >50% of projects with updated list of relevant critical security patches in past 6 months

COSTS

- ◆ Ongoing project overhead from buildout and maintenance of operational environment specification
- ◆ Ongoing project overhead from monitoring and installing critical security updates

PERSONNEL

- ◆ Developers (1-2 day/yr)
- ◆ Architects (1-2 day/yr)
- ◆ Managers (2-4 day/yr)
- ◆ Support/Operators (3-4 days/yr)

RELATED OBJECTIVES

- ◆ Operational Enablement - 2

RELATED COMPLIANCE

- ◆

IH 2

Infrastructure Hardening

Improve confidence in application operations by hardening the operating environment.

RESULTS

- ◆ Granular verification of security characteristics of systems in operations
- ◆ Formal expectations on timelines for infrastructure risk mitigation
- ◆ Stakeholders consistently aware of current operations status of software projects

ADD'L SUCCESS METRICS

- ◆ >80% of project teams briefed on patch management process in past 12 months
- ◆ >80% of stakeholders aware of current patch status in past 6 months

ADD'L COSTS

- ◆ Ongoing organization overhead from patch management and monitoring
- ◆ Buildout or license of infrastructure monitoring tools

ADD'L PERSONNEL

- ◆ Architects (1-2 days/yr)
- ◆ Developers (1-2 days/yr)
- ◆ Business Owners (1-2 days/yr)
- ◆ Managers (1-2 days/yr)
- ◆ Support/Operators (3-4 days/yr)

RELATED OBJECTIVES

- ◆

RELATED COMPLIANCE

- ◆

ACTIVITIES

Establish infrastructure patch management process

Moving to a more formal process than ad hoc application of critical patches, an ongoing process should be created in the organization to consistently apply security patches to infrastructure in the operating environment.

In the most basic form, the process should aim to make guarantees for time lapse between release and application of security patches. To make this process efficient, organizations typically accept high latency on lower priority patches, e.g. maximum of 2 days for critical patches spanning to a maximum of 30 days for low priority patches.

This activity should be primarily conducted by support and operations staff, but routine meetings with development should also be conducted to keep the whole project abreast of past changes and scheduled upgrades.

Additionally, development staff should share a list of third-party components upon which the software project internally depends so that support and operations staff can monitor those as well to cue development teams on when an upgrade is required.

Monitor baseline infrastructure configuration status

Given the complexity of monitoring and managing patches alone across the variety of components comprising the infrastructure for a software project, automation tools should be utilized to automatically monitor systems for soundness of configuration.

There are both commercial and open-source tools available to provide this type of functionality, so project teams should select a solution based on appropriateness to the organization's needs. Typical selection criteria includes ease of deployment and customization, applicability to the organization's platforms and technology stacks, built-in features for change management and alerting, metrics collection and trend tracking etc.

In addition to host and platform checks, monitoring automation should be customized to perform application-specific health checks and configuration verifications. Support and operations personnel should work with architects and developers to determine the optimal amount of monitoring for a given software project.

Ultimately, after a solution is deployed for monitoring the infrastructure's configuration status, unexpected alerts or configuration changes should be collected and regularly reviewed by project stakeholders as often as weekly but at least once per quarter.

Infrastructure Hardening

IH 3

Validate application health and status of operational environment against known best practices.

ACTIVITIES

Identify and deploy relevant operations protection tools

In order to build a better assurance case for software in its operating environment, additional tools can be used to enhance the security posture of the overall system. Operational environments can vary dramatically, thus the appropriateness of given protection technology should be considered in the project context.

Commonly used protections tools include web application firewalls, XML security gateways for web services, anti-tamper and obfuscation packages for client/embedded systems, network intrusion detection/prevention systems for legacy infrastructure, forensic log aggregation tools, host-based integrity verification tools, etc.

Based on the organization and project-specific knowledge, technical stakeholders should work with support and operations staff to identify and recommend selected operations protection tools to business stakeholders. If deemed a valuable investment in terms of risk-reduction versus cost of implementation, stakeholders should agree on plans for a pilot, widespread rollout, and ongoing maintenance.

Expand audit program for infrastructure configuration

When conducting routine project-level audits, expand the review to include inspection of artifacts related to hardening the operating environment. Beyond an up-to-date specification for the operational environment, audits should inspect current patch status and historic data since the previous audit. By tapping into monitoring tools, audits can also verify key factors about application configuration management and historic changes. Audits should also inspect the usage of operations protections tools against those available for the software's architecture type.

Audits for infrastructure can occur at any point after a project's initial release and deployment, but should occur at least every 6 months. For legacy systems or projects without active development, infrastructure audits should still be conducted and reviewed by business stakeholders. An exception process should be created to allow special-case projects to continue operations, but with an explicitly assigned timeframe for mitigation of findings. Exceptions should be limited to no more than 20% of all projects.

RESULTS

- ◆ Reinforced operational environment with layered checks for security
- ◆ Established and measured goals for operational maintenance and performance
- ◆ Reduced likelihood of successful attack via flaws in external dependencies

ADD'L SUCCESS METRICS

- ◆ >80% of stakeholders briefed on relevant operations protection tools in past 6 months
- ◆ >75% of projects passing infrastructure audits in past 6 months

ADD'L COSTS

- ◆ Research and selection of operations protection solutions
- ◆ Buildout or license of operations protections tools
- ◆ Ongoing operations overhead from maintenance of protection tools
- ◆ Ongoing project overhead from infrastructure-related audits

ADD'L PERSONNEL

- ◆ Business Owners (1 day/yr)
- ◆ Managers (1-2 days/yr)
- ◆ Support/Operators (3-4 days)

RELATED OBJECTIVES

- ◆ Standards & Compliance - 2

RELATED COMPLIANCE

- ◆

Operational Enablement

The Operational Enablement function is focused on gathering security critical information from the project teams building software and communicating it to the users and operators of the software. Without this information, even the most securely designed software carries undue risks since important security characteristics and choices will not be known at a deployment site.

Starting from lightweight documentation to capture the most impactful details for users and operators, an organization evolves toward building complete operational security guides that are delivered with each release.

In an advanced form, operational enablement also entails organization-level checks against individual project teams to ensure that information is being captured and shared according to expectations.

OBJECTIVES	Enable communications between development teams and operators for critical security-relevant data.	Improve expectations for continuous secure operations through provision of detailed procedures.	Mandate communication of security information and validate artifacts for completeness.
ACTIVITIES	<ul style="list-style-type: none"> ❖ Capture critical security information for deployment ❖ Document procedures for typical application alerts 	<ul style="list-style-type: none"> ❖ Create per-release change management procedures ❖ Maintain formal operational security guides 	<ul style="list-style-type: none"> ❖ Expand audit program for operational information ❖ Perform code signing for application components
RESULTS	<ul style="list-style-type: none"> ◆ Ad hoc improvements to software security posture through better understanding of correct operations ◆ Operators and users aware of their role in ensuring secure deployment ◆ Improved communications between software developers and users for security-critical information 	<ul style="list-style-type: none"> ◆ Detailed guidance for security-relevant changes delivered with software releases ◆ Updated information repository on secure operating procedures per application ◆ Alignment of operations expectations among developers, operators, and users. 	<ul style="list-style-type: none"> ◆ Organization-wide understanding of expectations for security-relevant documentation ◆ Stakeholders better able to make tradeoff decisions based on feedback from deployment and operations ◆ Operators and/or users able to independently verify integrity of software releases

OE 1

OE 2

OE 3

Operational Enablement

OE 1

Enable communications between development teams and operators for critical security-relevant data.

ACTIVITIES

Capture critical security information for deployment

With software-specific knowledge, project teams should identify and security-relevant configuration and operations information and communicate it to users and operators. This enables the actual security posture of software at deployment sites to function in the same way that designers in the project team intended.

This analysis should begin with architects and developers building a list of security features built-in to the software. From that list, information about configuration options and their security impact should be captured as well. For projects that offer several different deployment models, information about the security ramifications of each should be noted to better inform users and operators about the impact of their choices.

Overall, the list should be lightweight and aim to capture the most critical information. Once initially created, it should be reviewed by the project team and business stakeholders for agreement. Additionally, it is effective to review this list with select operators or users in order to ensure the information is understandable and actionable. Project teams should review and update this information with every release, but must do so at least every 6 months.

Document procedures for typical application alerts

With specific knowledge of ways in which software behaves, project teams should identify the most important error and alert messages which require user/operator attention. From each identified event, information related to appropriate user/operator actions in response to the event should be captured.

From the potentially large set of events that the software might generate, select the highest priority set based on relevance in terms of the business purpose of the software. This should include any security-related events, but also may include critical errors and alerts related to software health and configuration status.

For each event, actionable advice should be captured to inform users and operators of required next steps and potential root causes of the event. These procedures must be reviewed by the project team and updated every 6 months, but can be done more frequently, e.g. with each release.

RESULTS

- ◆ Ad hoc improvements to software security posture through better understanding of correct operations
- ◆ Operators and users aware of their role in ensuring secure deployment
- ◆ Improved communications between software developers and users for security-critical information

SUCCESS METRICS

- ◆ >50% of projects with updated deployment security information in past 6 months
- ◆ >50% of projects with operational procedures for events updated in past 6 months

COSTS

- ◆ Ongoing project overhead from maintenance of deployment security information
- ◆ Ongoing project overhead from maintenance of critical operating procedures

PERSONNEL

- ◆ Developers (1-2 days/yr)
- ◆ Architects (1-2 days/yr)
- ◆ Managers (1 days/yr)
- ◆ Support/Operators (1 days/yr)

RELATED OBJECTIVES

- ◆

RELATED COMPLIANCE

- ◆

Operational Enablement

Improve expectations for continuous secure operations through provision of detailed procedures.

RESULTS

- ◆ Detailed guidance for security-relevant changes delivered with software releases
- ◆ Updated information repository on secure operating procedures per application
- ◆ Alignment of operations expectations among developers, operators, and users.

ADD'L SUCCESS METRICS

- ◆ >50% of projects with updated change management procedures in past 6 months
- ◆ >80% of stakeholders briefed on status of operational security guides in past 6 months

ADD'L COSTS

- ◆ Ongoing project overhead from maintenance of change management procedures
- ◆ Ongoing project overhead from maintenance of operational security guides

ADD'L PERSONNEL

- ◆ Developers (1-2 days/yr)
- ◆ Architects (1-2 days/yr)
- ◆ Managers (1 days/yr)
- ◆ Support/Operators (1 days/yr)

RELATED OBJECTIVES

- ◆ Infrastructure Hardening - I

RELATED COMPLIANCE

- ◆

ACTIVITIES

Create per-release change management procedures

To more formally update users and operators on relevant changes in the software, each release must include change management procedures relevant to upgrade and first-time installation. Overall, the goal is to capture the expected accompanying steps that ensure the deployment will be successful and not incur excessive downtime or degradation of security posture.

To build these procedures during development, the project teams should setup a lightweight internal process for capturing relevant items that would impact deployments. It is effective to have this process in place early in the development cycle so that this information can be retained as soon as it is identified while in the requirements, design, and implementation phases.

Before each release, the project team should review the list as a whole for completeness and feasibility. For some projects, extensive change procedures accompanying a given release may warrant special handling, such as building automated upgrade scripts to prevent errors during deployment.

Maintain formal operational security guides

Starting from the information captured on critical software events and the procedures for handling each, project teams should build and maintain formal guides that capture all the security-relevant information that users and operators need to know.

Initially, this guide should be built from the known information about the system, such as security-related configuration options, event handling procedures, installation and upgrade guides, operational environment specifications, security-related assumptions about the deployment environment, etc. Extending this, the formal operational security guide should elaborate on each of these to cover more details such that the majority of the users and operators will be informed for all the questions they might have had. For large or complex systems, this can be challenging, so project teams should work with business stakeholders to determine the appropriate level of documentation. Additionally, project teams should document any recommendations for deployments that would enhance security.

The operational security guide, after initial creation, should be reviewed by project teams and updated with each release.

Operational Enablement

OE 3

Mandate communication of security information and validate artifacts for completeness.

ACTIVITIES

Expand audit program for operational information

When conducting routine project-level audits, expand the review to include inspection of artifacts related to operational enablement for security. Projects should be checked to ensure they have an updated and complete operational security guides as relevant to the specifics of the software.

These audits should begin toward the end of the development cycle close to release, but must be completed and passed before a release can be made. For legacy systems or inactive projects, this type of audit should be conducted and a one-time effort should be made to address findings and verify audit compliance, after which additional audits for operational enablement are no longer required.

Audit results must be reviewed with business stakeholders prior to release. An exception process should be created to allow projects failing an audit to continue with a release, but these projects should have a concrete timeline for mitigation of findings. Exceptions should be limited to no more than 20% of all active projects.

Perform code signing for application components

Though often used with special-purpose software, code signing allows users and operators to perform integrity checks on software such that they can cryptographically verify the authenticity of a module or release. By signing software modules, the project team enables deployments to operate with a greater degree of assurance against any corruption or modification of the the deployed software in its operating environment.

Signing code incurs overhead for management of signing credentials for the organization. An organization must follow safe key management processes to ensure the ongoing confidentiality of the signing keys. When dealing with any cryptographic keys, project stakeholders must also consider plans for dealing with common operational problems related to cryptography such as key rotation, key compromise, or key loss.

Since code signing is not appropriate for everything, architects and developers should work with security auditors and business stakeholders to determine which parts of the software should be signed. As projects evolve, this list should be reviewed with each release, especially when adding new modules or making changes to previously signed components.

RESULTS

- ◆ Organization-wide understanding of expectations for security-relevant documentation
- ◆ Stakeholders better able to make tradeoff decisions based on feedback from deployment and operations
- ◆ Operators and/or users able to independently verify integrity of software releases

ADD'L SUCCESS METRICS

- ◆ >80% of projects with updated operational security guide in last 6 months
- ◆ >80% of stakeholders briefed on code signing options and status in past 6 months

ADD'L COSTS

- ◆ Ongoing project overhead from audit of operational guides
- ◆ Ongoing organization overhead from management of code signing credentials
- ◆ Ongoing project overhead from identification and signing of code modules.

ADD'L PERSONNEL

- ◆ Developers (1 days/yr)
- ◆ Architects (1 days/yr)
- ◆ Managers (1 days/yr)
- ◆ Security Auditors (1-2 days/yr)

RELATED OBJECTIVES

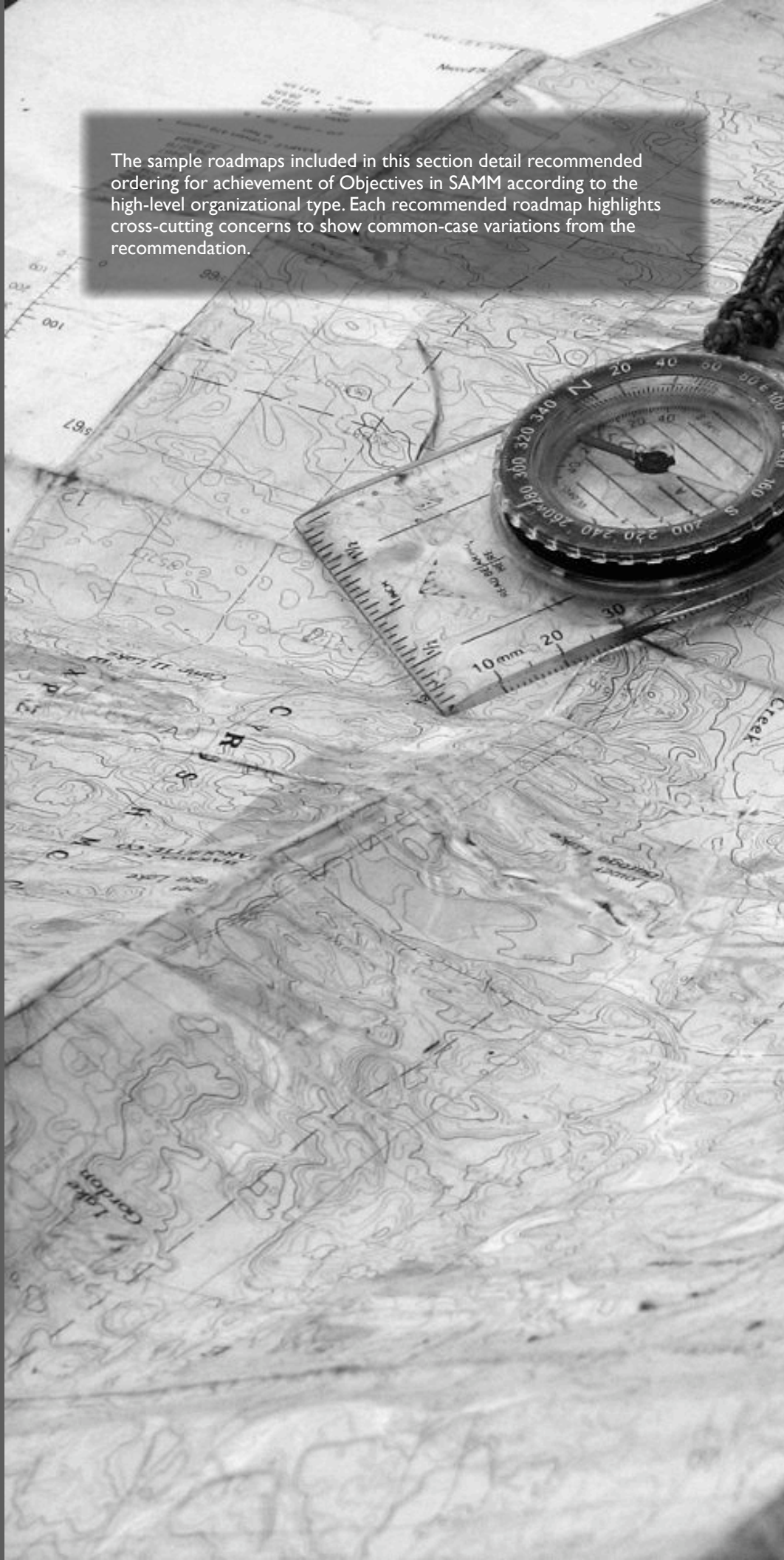
- ◆

RELATED COMPLIANCE

- ◆

Roadmaps

The sample roadmaps included in this section detail recommended ordering for achievement of Objectives in SAMM according to the high-level organizational type. Each recommended roadmap highlights cross-cutting concerns to show common-case variations from the recommendation.





Independent Software Vendors 56

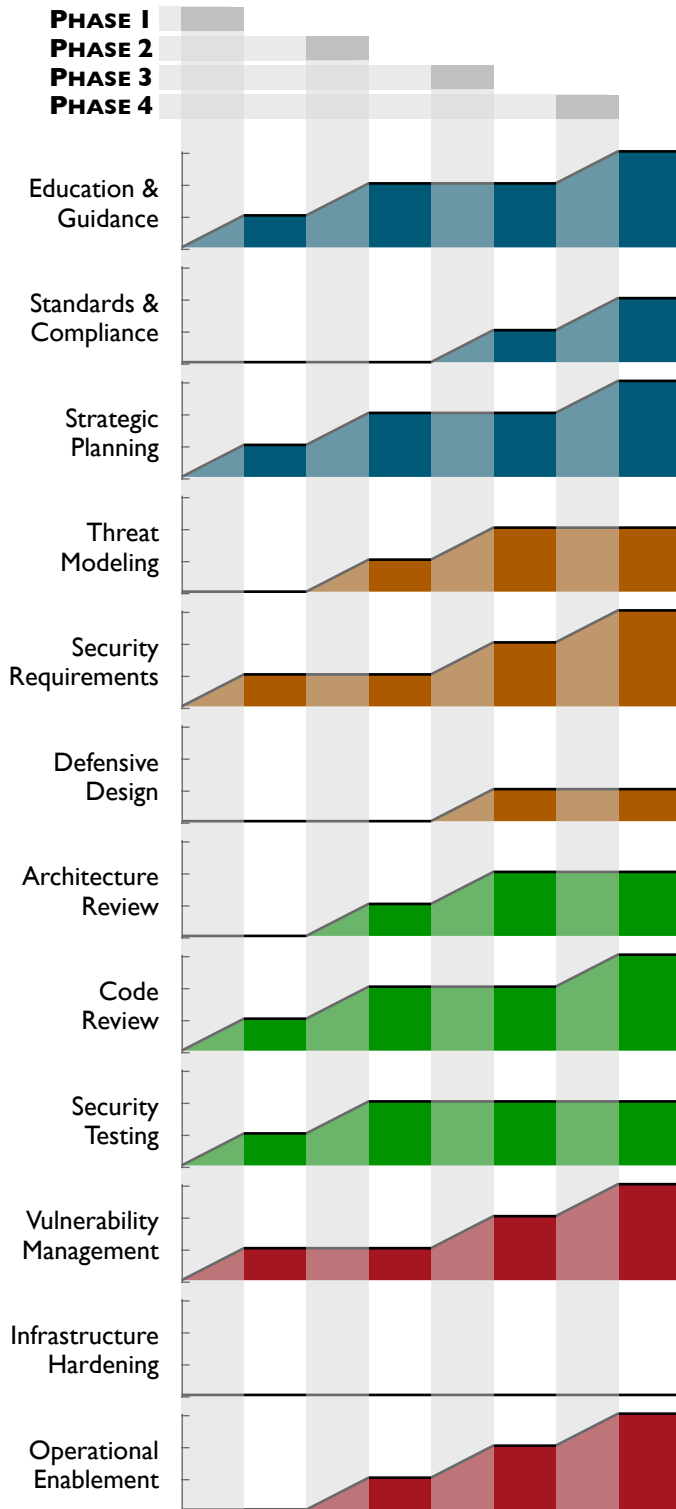
Online Services Providers 58

Financial Services Organizations *coming soon*

Government Organizations *coming soon*

Independent Software Vendors

Sample Roadmap



An Independent Software Vendor (ISV) involves the core business function of building and selling software components and applications.

Initial drivers to limit common vulnerabilities affecting customers and users leads to early concentration on Code Review and Security Testing activities.

Shifting toward more proactive prevention of security errors in product specification, an organization adds activities for Security Requirements over time.

Also, to minimize the impact from any discovered security issues, the organization ramps up Vulnerability Management activities over time.

As the organization matures, knowledge transfer activities from Operational Enablement are added to better inform customers and users about secure operation of the software.

RATIONALE

PHASE 1

- EG 1** Conduct technical security awareness training
Build and maintain technical guidance
- SP 1** Estimate overall business risk profile
Build and maintain assurance program roadmap
- SR 1** Derive security requirements from business functionality
Use guidelines, standards, and compliance resources
- CR 1** Create review checklists from known security requirements
Perform point-review of high-risk code
- ST 1** Derive test cases from known security requirements
Explicitly evaluate known security test-cases
- VM 1** Identify point of contact for security issues
Create informal security response team(s)

PHASE 2

- EG 2** Conduct role-specific application security training
Utilize security coaches to enhance project teams
- SP 2** Classify data and applications based on business risk
Establish per-classification security goals
- TM 1** Build and maintain per-application attack trees
Develop attacker profile from software architecture
- AR 1** Identify software attack surface
Analyze design against known security requirements
- CR 2** Utilize automated code analysis tools
Integrate code analysis into development process
- ST 2** Utilize automated security testing tools
Integrate security testing into development process
- OE 1** Capture critical security information for deployment
Document procedures for typical application alerts

PHASE 3

- SC 1** Identify and monitor external compliance drivers
Build and maintain compliance checklist
- TM 2** Elaborate attack trees with application-specific data
Adopt a weighting system for measurement of threats
- SR 2** Build and maintain abuse-case models per project
Specify security requirements based on known risks
- DD 1** Maintain list of recommended software frameworks
Explicitly apply security principles to perimeter interfaces
- AR 1** Inspect for complete provision of security mechanisms
Deploy design review service for project teams
- VM 2** Establish consistent incident response process
Adopt a security issue disclosure process
- OE 2** Create per-release change management procedures
Maintain formal operational security guides

PHASE 4

- EG 3** Create application security support portal
Establish role-based examination/certification
- SC 2** Build internal standards for security and compliance
Establish project audit practice
- SP 3** Conduct periodic industry-wide cost comparisons
Collect metrics for historic security spend
- SR 3** Build security requirements into vendor agreements
Expand audit program for security requirements
- CR 3** Customize code analysis for application-specific concerns
Establish release gates for code review
- VM 3** Conduct root cause analysis for incidents
Collect per-incident metrics
- OE 3** Expand audit program for operational information
Perform code signing for application components

ACTIVITIES

There are several common factors to ISVs that may affect the roadmap for an organization.

OUTSOURCED DEVELOPMENT

For organizations using external development resources, restrictions on code access typically leads to prioritization of Security Requirements activities instead of Code Review activities. Additionally, advancing Threat Modeling in earlier phases would allow the organization to better clarify security needs to the outsourced developers. Since expertise on software configuration will generally be strongest within the outsourced group, contracts should be constructed to account for the activities related to Operations Enablement.

INTERNET-CONNECTED APPLICATIONS

Organizations building applications that use online resources have additional risks from the core internet-facing infrastructure that hosts the internet-facing systems. To account for this risk, organizations should add activities from Infrastructure Hardening to their roadmaps.

DRIVERS AND EMBEDDED DEVELOPMENT

For organizations building low-level drivers or software for embedded systems, security vulnerabilities in software design can be more damaging and costly to repair. Therefore, roadmaps should be modified to emphasize Defensive Design and Architecture Review activities in earlier phases.

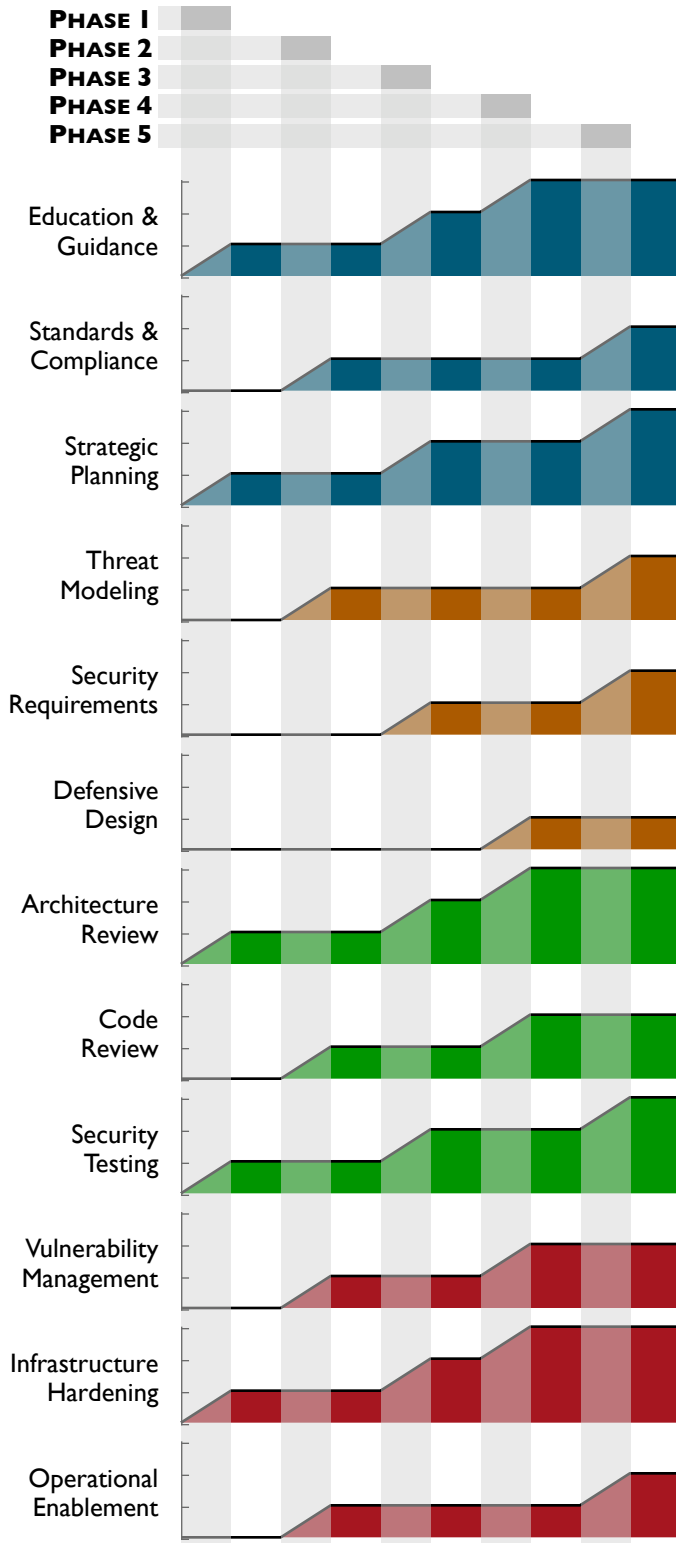
ORGANIZATIONS GROWN BY ACQUISITION

In an organization grown by acquisition, there can often be several project teams following different development models with varying degrees of security-related activities incorporated. An organization such as this may require a separate roadmap for each division or project team to account for varying starting points as well as project-specific concerns if a variety of software types are being developed.

CONSIDERATIONS

Online Services Providers

Sample Roadmap



An Online Services Provider (OSP) involves the core business function of building web applications and other network-accessible interfaces.

Initial drivers to validate the overall soundness of design without stifling innovation lead to early concentration on Architecture Review and Security Testing activities.

Since critical systems will be network-facing, Infrastructure Hardening activities are also added early and ramped over time to account for risks from the hosted environment.

Though it can vary based on the core business of the organizations, Standards & Compliance activities should be started early and then advanced according to the criticality of external compliance drivers.

As the organization matures, activities from Threat Modeling, Security Requirements, and Defensive Design are slowly added to help bolster proactive security after some baseline expectations for security have been established.

RATIONALE

PHASE 1

- EG 1** Conduct technical security awareness training
Build and maintain technical guidance
- SP 1** Estimate overall business risk profile
Build and maintain assurance program roadmap
- AR 1** Identify software attack surface
Analyze design against known security requirements
- ST 1** Derive test cases from known security requirements
Explicitly evaluate known security test-cases
- OE 1** Maintain operational environment specification
Identify and install critical security patches

PHASE 2

- SC 1** Identify and monitor external compliance drivers
Build and maintain compliance checklist
- TM 1** Build and maintain per-application attack trees
Develop attacker profile from software architecture
- CR 1** Create review checklists from known security requirements
Perform point-review of high-risk code
- VM 1** Identify point of contact for security issues
Create informal security response team(s)
- OE 1** Capture critical security information for deployment
Document procedures for typical application alerts

PHASE 3

- EG 2** Conduct role-specific application security training
Utilize security coaches to enhance project teams
- SP 2** Classify data and applications based on business risk
Establish per-classification security goals
- SR 1** Derive security requirements from business functionality
Use guidelines, standards, and compliance resources
- AR 2** Inspect for complete provision of security mechanisms
Deploy design review service for project teams
- ST 2** Utilize automated security testing tools
Integrate security testing into development process
- IH 2** Establish infrastructure patch management process
Monitor baseline infrastructure configuration status

PHASE 4

- EG 3** Create application security support portal
Establish role-based examination/certification
- DD 1** Maintain list of recommended software frameworks
Explicitly apply security principles to perimeter interfaces
- AR 3** Develop data-flow diagrams for sensitive resources
Establish release gates for architecture review
- CR 2** Utilize automated code analysis tools
Integrate code analysis into development process
- VM 2** Establish consistent incident response process
Adopt a security issue disclosure process
- IH 3** Identify and deploy relevant operations protection tools
Expand audit program for infrastructure configuration

PHASE 5

- SC 2** Build internal standards for security and compliance
Establish project audit practice
- SP 3** Conduct periodic industry-wide cost comparisons
Collect metrics for historic security spend
- TM 2** Elaborate attack trees with application-specific data
Adopt a weighting system for measurement of threats
- SR 2** Build and maintain abuse-case models per project
Specify security requirements based on known risks
- ST 3** Employ application-specific security testing automation
Establish release gates for security testing
- OE 2** Create per-release change management procedures
Maintain formal operational security guides

ACTIVITIES

There are several common factors to OSPs that may affect the roadmap for an organization.

OUTSOURCED DEVELOPMENT

For organizations using external development resources, restrictions on code access typically leads to prioritization of Security Requirements activities instead of Code Review activities. Additionally, advancing Threat Modeling in earlier phases would allow the organization to better clarify security needs to the outsourced developers. Since expertise on software configuration will generally be strongest within the outsourced group, contracts should be constructed to account for the activities related to Operations Enablement.

PCI COMPLIANT ORGANIZATIONS

Organizations required to be in compliance with the Payment Card Industry Data Security Standard (PCI-DSS) should place activities from Standards & Compliance in earlier phases of the roadmap. This allows the organization to opportunistically establish activities that ensure compliance with PCI-DSS and allows the future roadmap to be tailored accordingly.

WEB SERVICES PLATFORMS

For organizations building web services platforms, design errors can carry additional risks and be more costly to mitigate. Therefore, activities from Threat Modeling, Security Requirements, and Defensive Design should be placed in earlier phases of the roadmap.

ORGANIZATIONS GROWN BY ACQUISITION

In an organization grown by acquisition, there can often be several project teams following different development models with varying degrees of security-related activities incorporated. An organization such as this may require a separate roadmap for each division or project team to account for varying starting points as well as project-specific concerns if a variety of software types are being developed.

CONSIDERATIONS

Case Studies

This section features a number of scenarios in which the Maturity Model is adapted and used by a specific business. Using the recommended roadmaps as a guide, the case studies tell the story of how an organization might adapt best practices and take into account organization-specific risks.





VirtualWare - an Independent Software Vendor	62
❖ Phase 1 (Months 0 – 3) – Awareness & Planning	64
❖ Phase 2 (Months 3 – 6) – Education & Testing	66
❖ Phase 3 (Months 6 – 9) – Architecture & Infrastructure	68
❖ Phase 4 (Months 9 – 12) – Governance & Operational Security	70
❖ Ongoing (Months 12+)	72
Moogle - an Online Services Provider	<i>coming soon</i>
UniGroup - a Financial Services Organization	<i>coming soon</i>
SocialLink - a Government Organization	<i>coming soon</i>

VirtualWare

Case Study: A medium-sized ISV

BUSINESS PROFILE

VirtualWare is a leader within their market for providing integrated virtualized application platforms to help organizations consolidate their application interfaces into a single environment. Their technology is provided as a server application and desktop client built for multiple environments including Microsoft, Apple and Linux platforms.

The organization is of medium size (200-1000 employees) and has a global presence around the world with branch offices in most major countries.

ORGANIZATION

Virtualware has been developing their core software platform for over 8 years. During this time they have had limited risk from common web vulnerabilities due to minimal usage of web interfaces. Most of the Virtualware platforms are run through either a server based systems or thick clients running on the desktop.

Recently Virtualware started a number of new project streams, which deliver their client and server interfaces via web technology. Knowing the extent of common attacks seen over the web, this has driven the organization to review their Software Security strategy and ensure that it adequately addresses possible threats towards their organization going forward.

Previously the organization had undertaken basic reviews of the application code, and has been more focused on performance and quality rather than security. Virtualware developers have been using a number of code quality analysis tools to identify bugs and address them within the code.

With this in mind, the upper management team has set a strategic objective to review the current status of the security of their applications and determine the best method of identifying, removing, and preventing vulnerabilities in their applications.

ENVIRONMENT

VirtualWare develops their virtualization technology on a mixture of Java, C++ and Microsoft .NET technology. Their core application virtualization technology has been written in C++ and has had a number of reviews for bugs and security, but currently no formal processes exists for identifying and fixing known or unknown security bugs.

Virtualware has chosen to support their web technology on Java, although the back-end systems are built using Microsoft and C++ technologies. The development team focused on the new web interfaces is primarily composed of Java developers.

Virtualware employs over 300 developers, with staff broken up into teams based on the projects that they work on. There are 12 teams with an average of 20 – 40 developers per team. Within each team there is minimal experience with software security, and although senior developers perform basic assessments of their code, security is not considered a critical goal within the organization.

Each team within Virtualware adopts a different development model. Currently the two primary methodologies used are agile SCRUM and iterative Waterfall style approaches. There is minimal to no guidance from the IT department or project architects on software security.

KEY CHALLENGES

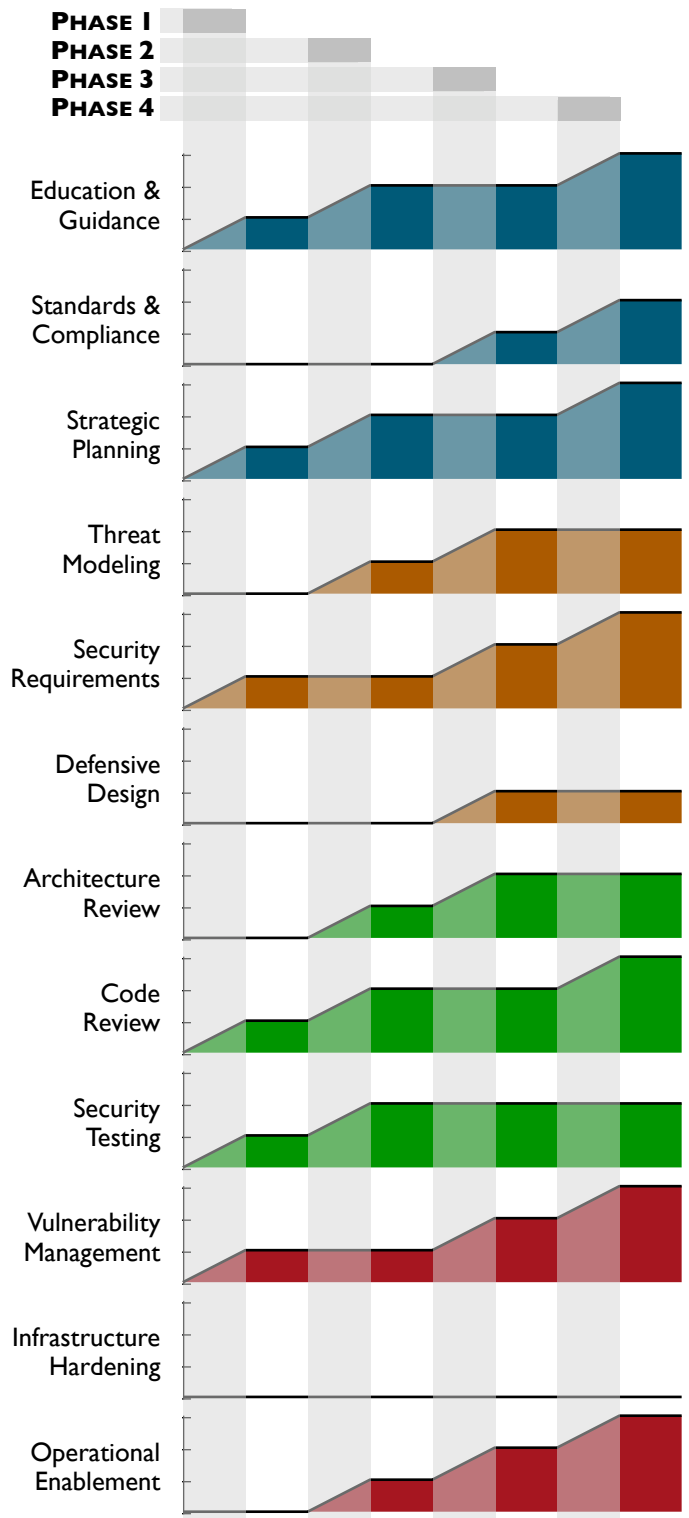
- ◆ Rapid release of application features to ensure they maintain their competitive edge over rivals
- ◆ Limited experience with software security concepts, currently minimal effort is associated with security related tasks
- ◆ Developers leave the organization and are replaced with less experienced developers
- ◆ Multiple technologies used within applications, with legacy applications that have not been updated since originally built
- ◆ No understanding of existing security posture or risks facing the organization

Virtualware wanted to focus on ensuring that their new web applications would be delivered securely to their customers. Therefore the initial focus on implementing the security assurance program was on education and awareness for their development teams, as well as providing some base technical guidance on secure coding and testing standards.

The organization previously had received bug requests and security vulnerabilities through their support@virtualware.net address. However as this was a general support address, existing requests were not always filtered down to the appropriate teams within the organization and handled correctly. The need to implement a formal security vulnerability response program was also identified by Virtualware.

IMPLEMENTATION STRATEGY

The adoption of a Security Assurance program within an organization is a long term strategy, and significantly impacts on the culture of developers and the process taken by the business to develop and deliver business applications. The adoption of this strategy is set over a 12 month period, and due to the size of the organization will be relatively easy to implement in that period.



VirtualWare - Phase 1

PHASE 1 (MONTHS 0 – 3) – AWARENESS & PLANNING

Virtualware previously identified that they had limited knowledge and awareness of application security threats to their organization and limited secure coding experience. The first phase of the deployment within Virtualware focused on training developers and implementing guidance and programs to identify current security vulnerabilities.

Development teams within Virtualware had limited experience in secure coding techniques therefore, an initial training program was developed that can be provided to the developers within the organization on defensive programming techniques.

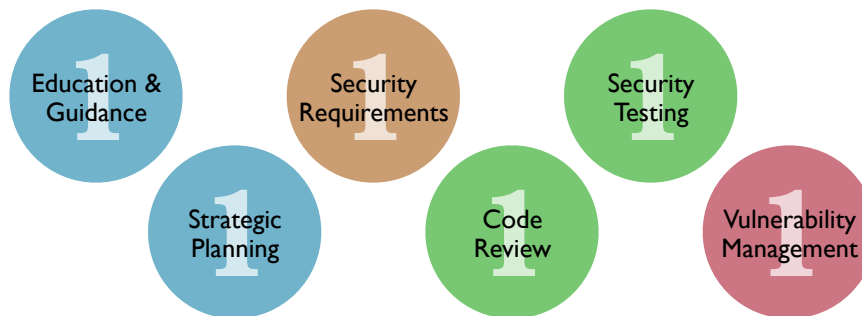
With over 300 developers and multiple languages supported within the organization one of the key challenges for Virtualware was to provide an education program that was technical enough to teach developers some of the basic's in secure coding concepts. The objective of this initial education course was primarily on coding techniques and testing tools. The course developed and delivered within the organization lasted for 1 day and covered the basics of secure coding.

Virtualware was aware that they had a number of applications with vulnerabilities and no real strategy in which to identify existing vulnerabilities and address the risks in a reasonable time-frame. A basic risk assessment methodology was adopted and the organization undertook a review of the existing application platforms.

This phase also included implementing a number of concepts for the development team to enhance their security tools. The development teams already had a number of tools available to perform quality type assessments. Additional investigation into code review and security testing tools was performed.

SAMM Objectives

During this phase of the project, Virtualware implemented the following SAMM Functions & Objectives.



ACTIVITIES

- | | | | |
|-------------|--|-------------|---|
| EG I | ✦ Conduct technical security awareness training | CR I | ✦ Create review checklists from known security requirements |
| | ✦ Build and maintain technical guidance | | ✦ Perform point-review of high-risk code |
| SP I | ✦ Estimate overall business risk profile | ST I | ✦ Derive test cases from known security requirements |
| | ✦ Build and maintain assurance program roadmap | | ✦ Explicitly evaluate known security test-cases |
| SR I | ✦ Derive security requirements from business functionality | VM I | ✦ Identify point of contact for security issues |
| | ✦ Use guidelines, standards, and compliance resources | | ✦ Create informal security response team(s) |

To achieve these maturity levels Virtualware implemented a number of programs during this phase of the roll-out. The following initiatives were adopted;

- ◆ 1 Day Secure Coding Course (High-level) for all developers;
- ◆ Build a technical guidance whitepaper for application security on technologies used within the organization;
- ◆ Create a risk process and perform high-level business risk assessments for the application platforms and review business risk;
- ◆ Prepare initial technical guidelines and standards for developers;
- ◆ Perform short code reviews on application platforms that present significant risk to the organization;
- ◆ Develop test and use cases for projects and evaluate the cases against the applications;
- ◆ Appointed a role to application security initiatives;
- ◆ Generated a Draft strategic roadmap for the next phase of the assurance program.

Due to the limited amount of expertise in-house within Virtualware, the company engaged with a third party security consulting group to assist with the creation of the training program, and assist in writing the threat modeling and strategic roadmap for the organization.

One of the key challenges faced during this phase, was to get all 300 developers through a one day training course. To achieve this Virtualware ran 20 course days, with only a small number of developers from each team attending the course at one time. This reduced the overall impact on staff resources during the training period.

During this phase of the project, Virtualware invested significant resources effort into the adoption of a risk review process and reviewing the business risk to the organization. Although considerable effort was focused on these tasks, they were critical to ensuring that the next steps implemented by Virtualware were in line with the business risks faced by the organization.

Virtualware management received positive feedback from most developers within the organization on the training program. Although not detailed, developers felt that the initial training provided some basic skills that could assist them immediately day to day in writing secure code.

IMPLEMENTATION COSTS

A significant amount of internal resources and costs were invested in this phase of the project. There were three different types of costs associated with this phase including.

Internal Resource Requirements

Internal resource effort used in the creation of content, workshops and review of application security initiatives within this phase. Effort is shown in total days per role.

Developer	14 days	Business Owner	8 days
Architect	10 days	QA Tester	3 days
Manager	8 days	Security Auditor	9 days

Training Resource Requirements (Training per person for period)

Each developer within Virtualware was required to attend a training course, and therefore every developer had a single day allocated to the application security program.

Developer (per person)	1 day
------------------------	-------

Outsourced Resources

Due to the lack of knowledge within Virtualware, external resources were used to assist with the creation of content, and create/deliver the training program to the developers.

Consultant (Security)	15 days	Consultant (Training)	22 days
-----------------------	---------	-----------------------	---------

VirtualWare - Phase 2

PHASE 2 (MONTHS 3 – 6) – EDUCATION & TESTING

Virtualware identified in phase 1 that a number of their applications contained vulnerabilities that may be exploited by external threats. Therefore one of the key objectives of this phase was to implement basic testing and review capabilities to identify the vulnerabilities and address them in the code.

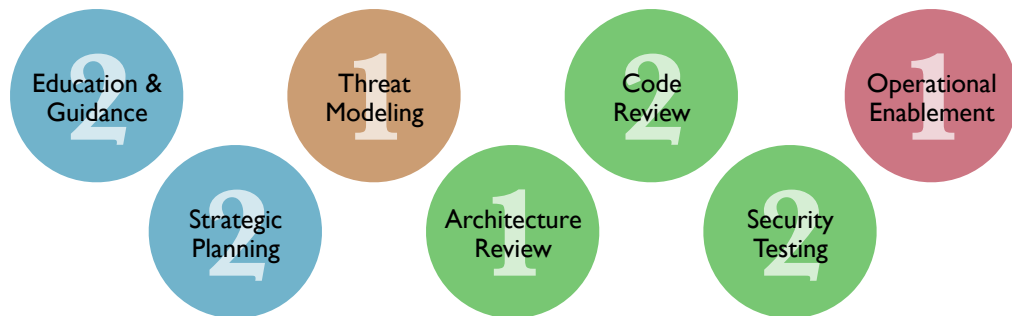
The introduction of automated tools to assist with code coverage and findings weaknesses was identified as one of the biggest challenges in this phase of the implementation. Traditionally in the past developers have used automated tools with great difficulty and therefore implementing new tools was seen as a significant challenge.

To ensure a successful rollout of the automation tools within the organization, Virtualware proceeded with a staged roll-out. The tools would be given to senior team leaders first, with other developers coming online over a period of time. Teams were encouraged to adopt the tools however no formal process was put in place for their use.

This phase of the implementation also saw the introduction of a more formal education and awareness program. Developers from the previous training requested more specific training in the areas of Web Services, and data validation. The new 6 hour specific training course was developed with these two focus areas. Virtualware will also implement additional training programs for Architects, Managers and adopt an awareness campaign within the organization.

SAMM Objectives

During this phase of the project, Virtualware implemented the following SAMM Functions & Objectives.



ACTIVITIES

- | | | | |
|-------------|---|-------------|--|
| EG 2 | ✦ Conduct role-specific application security training | CR 2 | ✦ Utilize automated code analysis tools |
| | ✦ Utilize security coaches to enhance project teams | | ✦ Integrate code analysis into development process |
| SP 2 | ✦ Classify data and applications based on business risk | ST 2 | ✦ Utilize automated security testing tools |
| | ✦ Establish per-classification security goals | | ✦ Integrate security testing into development process |
| TM 1 | ✦ Build and maintain per-application attack trees | OE 1 | ✦ Capture critical security information for deployment |
| | ✦ Develop attacker profile from software architecture | | ✦ Document procedures for typical application alerts |
| AR 1 | ✦ Identify software attack surface | | |
| | ✦ Analyze design against known security requirements | | |

To achieve these maturity levels Virtualware implemented a number of programs during this phase of the roll-out. The following initiatives were adopted;

- ◆ Additional Education & Training courses for QA Testers, Managers & Architects;
- ◆ Conduct Data asset classification and set security goals;
- ◆ Develop the risk assessment methodology into a threat modeling approach with attack tress and profiles;
- ◆ Review and identify security requirements per application platform;
- ◆ Introduction of automated tools to assist with code coverage and security analysis of existing applications and new code bases;
- ◆ Review and enhance existing penetration testing programs;
- ◆ Enhance the existing SDL to support security testing as a part of the development process.

Virtualware adapted the existing application security training program, to provider a smaller less technical version as a Business Application Security awareness program. This was a shorter 4 hour course, and was extended to Managers, Business Owners of the organization.

A high-level review of the existing code review and penetration testing programs identified that the process was inadequate and needed to be enhanced to provide better testing and results on application security vulnerabilities. The team set out to implement a new program of performing penetration testing and code reviews. As a part of this program, each senior developer in a program team was allocated approximately 4 days to perform a high-level source code review of their application.

Virtualware management understood that the infrastructure and applications are tightly integrated, and during this phase the operational side of the application platforms (infrastructure) was reviewed. This phase looked at the infrastructure requirements and application integration features between the recommended deployed hardware and the application interfaces.

During this phase the strategic roadmap and methodology for application security was reviewed by the project team. The objective of this review and update was to formally classify data assets and set the appropriate level of business risk associated with the data assets and applications. From this the project team was able to set security goals for these applications.

IMPLEMENTATION COSTS

A significant amount of internal resources and costs were invested in this phase of the project. There were three different types of costs associated with this phase including.

Internal Resource Requirements

Internal resource effort used in the creation of content, workshops and review of application security initiatives within this phase. Effort is shown in total days per role.

Developer	8 days	Business Owner	5 days
Architect	10 days	QA Tester	3 days
Manager	8 days	Security Auditor	15 days
Support Operations	2 days		

Training Resource Requirements (Training per person for period)

Each developer within Virtualware was required to attend a training course, and therefore every developer had a single day allocated to the application security program.

Architect (per person)	1 day	Manager (per person)	1/2 day
Bus. Owner (per person)	1/2 day		

Outsourced Resources

Due to the lack of knowledge within Virtualware, external resources were used to assist with the creation of content, and create/deliver the training program to the developers.

Consultant (Security)	22 days	Consultant (Training)	5 days
-----------------------	---------	-----------------------	--------

VirtualWare - Phase 3

PHASE 3 (MONTHS 6 – 9) – ARCHITECTURE & INFRASTRUCTURE

The third phase of the assurance program implementation within Virtualware builds on from the previous implementation phases and focuses on risk modeling, architecture, infrastructure and operational enablement capabilities.

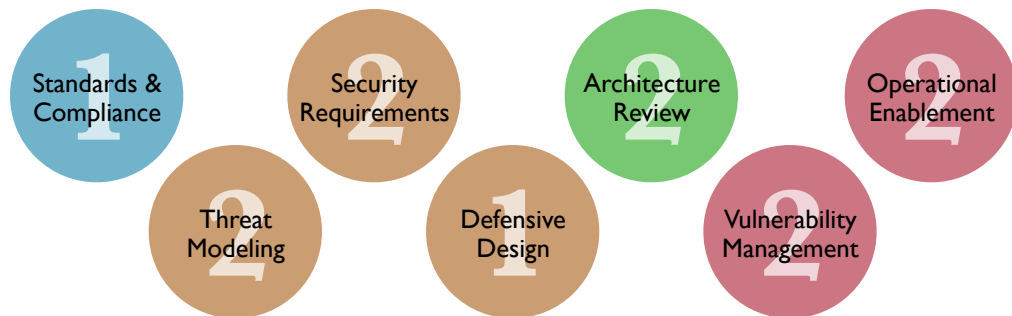
The key challenge in this phase was on establishing a tighter integration between the application platforms and operational side of the organization. In the previous phase Virtualware teams were introduced to vulnerability management and the operational side of application security. During this phase Virtualware has adopted the next phase of these areas and introduced clear incident response processes and detailed change control procedures.

Virtualware has chosen to start two new areas for this implementation. Although Virtualware is not impacted by regulatory compliance, a number of their customers have started to ask about whether the platforms can assist in passing regulatory compliance. A small team has been setup within Virtualware to identify the relevant compliance drivers and create a checklist of drivers.

In the previous phase Virtualware introduced a number of new automated tools to assist with the review and identification of vulnerabilities. Although not focused on in this phase, the development teams have adopted the new tools and have reported that they are starting to gain a benefit from using these tools within their groups.

SAMM Objectives

During this phase of the project, Virtualware implemented the following SAMM Functions & Objectives.



ACTIVITIES

- | | | | |
|-------------|--|-------------|---|
| SC 1 | <ul style="list-style-type: none"> ✦ Identify and monitor external compliance drivers ✦ Build and maintain compliance checklist | AR 2 | <ul style="list-style-type: none"> ✦ Inspect for complete provision of security mechanisms ✦ Deploy design review service for project teams |
| TM 1 | <ul style="list-style-type: none"> ✦ Elaborate attack trees with application-specific data ✦ Adopt a weighting system for measurement of threats | VM 2 | <ul style="list-style-type: none"> ✦ Establish consistent incident response process ✦ Adopt a security issue disclosure process |
| SR 2 | <ul style="list-style-type: none"> ✦ Build and maintain abuse-case models per project ✦ Specify security requirements based on known risks | OE 2 | <ul style="list-style-type: none"> ✦ Create per-release change management procedures ✦ Maintain formal operational security guides |
| DD 1 | <ul style="list-style-type: none"> ✦ Maintain list of recommended software frameworks ✦ Explicitly apply security principles to perimeter interfaces | | |

To achieve these maturity levels Virtualware implemented a number of programs during this phase of the roll-out. The following initiatives were adopted;

- ◆ Define and publish technical guidance on security requirements and defensive design for projects within the organization;
- ◆ Identify and document compliance and regulatory requirements;
- ◆ Identify and create guidelines for security of application infrastructure;
- ◆ Create a defined list of approved development frameworks;
- ◆ Enhance the existing threat modeling process used within Virtualware;
- ◆ Adopt a incident response plan and prepare a security disclosure process;
- ◆ Introduce Change Management procedures and formal guidelines for all projects.

To coincide with the introduction of automated tools for developers (from the previous phase), formal technical guidance on secure coding techniques was introduced into the organization. These were specific technical documents relating to languages and technology and provided guidance on secure coding techniques in each relevant language/application.

With a combined approach from the education and awareness programs, technical guidance and then the introduction of automation tools to help the developers, Virtualware started to see a visible difference in the code being delivered into production versions of their applications. Developers provided positive feedback on the tools and education made available to them under the program.

For the first time in Virtualware project teams became responsible for their security and design of their application platforms. During this phase a formal review process and validation against best practices were performed by each team. Some teams identified gaps relating to both security and business design that needed to be reviewed. A formal plan was put in place to ensure these gaps were addressed.

A formal incident response plan and change management procedures were introduced during this phase of the project. This was a difficult process to implement, and Virtualware teams initially struggled with the process as the impact on culture and the operational side of the business was significant. However over time each team member identified the value in the new process and the changes were accepted by the team over the implementation period.

IMPLEMENTATION COSTS

A significant amount of internal resources and costs were invested in this phase of the project. There were three different types of costs associated with this phase including.

Internal Resource Requirements

Internal resource effort used in the creation of content, workshops and review of application security initiatives within this phase. Effort is shown in total days per role.

Developer	5 days	Business Owner	6 days
Architect	7 days	Security Auditor	10 days
Manager	9 days	Support Operations	3 days

Outsourced Resources

Due to the lack of knowledge within Virtualware, external resources were used to assist with the creation of content, and create/deliver the processes, guidelines and assist teams.

Consultant (Security)	20 days
-----------------------	---------

VirtualWare - Phase 4

PHASE 4 (MONTHS 9 – 12) – GOVERNANCE & OPERATIONAL SECURITY

The fourth phase of the assurance program implementation within Virtualware continues on from the previous phases, by enhancing existing security functions within the organization. By now Virtualware has implemented a number of critical application security processes and mechanisms to ensure that applications are developed and maintained securely.

A core focus in this phase is bolstering the Alignment & Governance Discipline. These three functions play a critical role in the foundation of an effective long term application security strategy. A completed education program is implemented, whilst at the same time a long term strategic roadmap is put in place for Virtualware.

The other key focus within this phase is on the operational side of the implementation. Virtualware management identified previously that the need for incident response plans and dedicated change management processes are critical to the long term strategy.

Virtualware saw this phase as the stepping stones to their long term future. This phase saw the organization implement a number of final measures to cement the existing building blocks that have been laid down in the previous phases. In the long term this will ensure that the processes, concepts and controls put in place will continue to work within the organization to ensure the most secure outcome for their application platforms.

Virtualware chose this phase to introduce their customers to their new application security initiatives, provide details of a series of programs to Virtualware customers about application security, deploying applications securely and reporting of vulnerabilities in Virtualware applications. The key goal from these programs is to instill confidence in their customer base that Virtualware applications are built with security in-mind, and Virtualware can assist customers in ensuring their application environments using their technology are secure.

SAMM Objectives

During this phase of the project, Virtualware implemented the following SAMM Functions & Objectives.



ACTIVITIES

- EG 3** ✦ Create application security support portal
- EG 3** ✦ Establish role-based examination/certification
- SC 2** ✦ Build internal standards for security and compliance
- SC 2** ✦ Establish project audit practice
- SP 3** ✦ Conduct periodic industry-wide cost comparisons
- SP 3** ✦ Collect metrics for historic security spend
- SR 3** ✦ Build security requirements into vendor agreements
- SR 3** ✦ Expand audit program for security requirements
- CR 3** ✦ Customize code analysis for application-specific concerns
- CR 3** ✦ Establish release gates for code review
- VM 3** ✦ Conduct root cause analysis for incidents
- VM 3** ✦ Collect per-incident metrics
- OE 3** ✦ Expand audit program for operational information
- OE 3** ✦ Perform code signing for application components

To achieve these maturity levels Virtualware implemented a number of programs during this phase of the roll-out. The following initiatives were adopted;

- ◆ Create well defined security requirements and testing program for all projects;
- ◆ Create and implement a incident response plan;
- ◆ Reviewed existing alerts procedure for applications and document a process for capturing events;
- ◆ Create a customer security white-paper on deploying applications security;
- ◆ Review existing security spend within projects and determine if appropriate budget has been allocated to each project for security;
- ◆ Implement the final education and awareness programs for application roles;
- ◆ Complete a long term application security strategy roadmap for the organization.

In previous phases Virtualware had released a formal incident response plan for customers to submit vulnerabilities found with their code. During this phase, Virtualware took the results of the submitted vulnerabilities and conducted assessments of why the problem occurred, how and attempted a series of reporting to determine any common theme identified amongst the reported vulnerabilities.

As a part of the ongoing effort to ensure applications are deployed internally securely as well as on customer networks, Virtualware created a series of white-papers, provided to customers based on industry standards for recommended infrastructure hardening. The purpose of these guidelines is to provide assistance to customers on the best approach to deploying their applications.

Virtualware implemented during this phase a short CBT based training so that existing and new developers could maintain their skills in application security. It was also mandated that all “application” associated roles undertake a mandatory 1 day course per year. This was completed to ensure that the skills given to developers were not lost and new developers would be up skilled during their time with the company.

One of the final functions implemented within Virtualware was to complete a “AS IS” gap assessment and review, and determine how effective the past 12 months has been. During this short program questionnaires were sent to all team members involved as well as a baseline review against SAMM. The weaknesses and strengths identified during this review were documented into the final strategic roadmap for the organization and the next twelve months strategy was set for Virtualware.

IMPLEMENTATION COSTS

A significant amount of internal resources and costs were invested in this phase of the project. There were three different types of costs associated with this phase including.

Internal Resource Requirements

Internal resource effort used in the creation of content, workshops and review of application security initiatives within this phase. Effort is shown in total days per role.

Developer	4 days	Business Owner	6 days
Architect	7 days	QA Tester	1 day
Manager	9 days	Security Auditor	11 days

Outsourced Resources

Due to the lack of knowledge within Virtualware, external resources were used to assist with the implementation of this phase, including documentation, processes and workshops.

Consultant (Security)	22 days
-----------------------	---------

VirtualWare - Ongoing

ONGOING (MONTHS 12+)

Over the past twelve months Virtualware has started by implementing a number of training and education programs, to developing internal guidelines and policies. In the final phase of the assurance program implementation, Virtualware began to publish externally and work with their customers to enhance the security of their customer application platforms.

Virtualware Management set an original mandate to ensure that software developed within the company was secure, and to ensure that the market was aware of the security initiatives taken and to assist customers in securing their application platforms.

To achieve these management goals the first twelve months set the path for an effective strategy within Virtualware, and finally by starting to assist customers in securing their application environments. Moving forward Virtualware has set a number of initiatives within the organization to ensure that the company doesn't fall into their old habits. Some of these programs include:

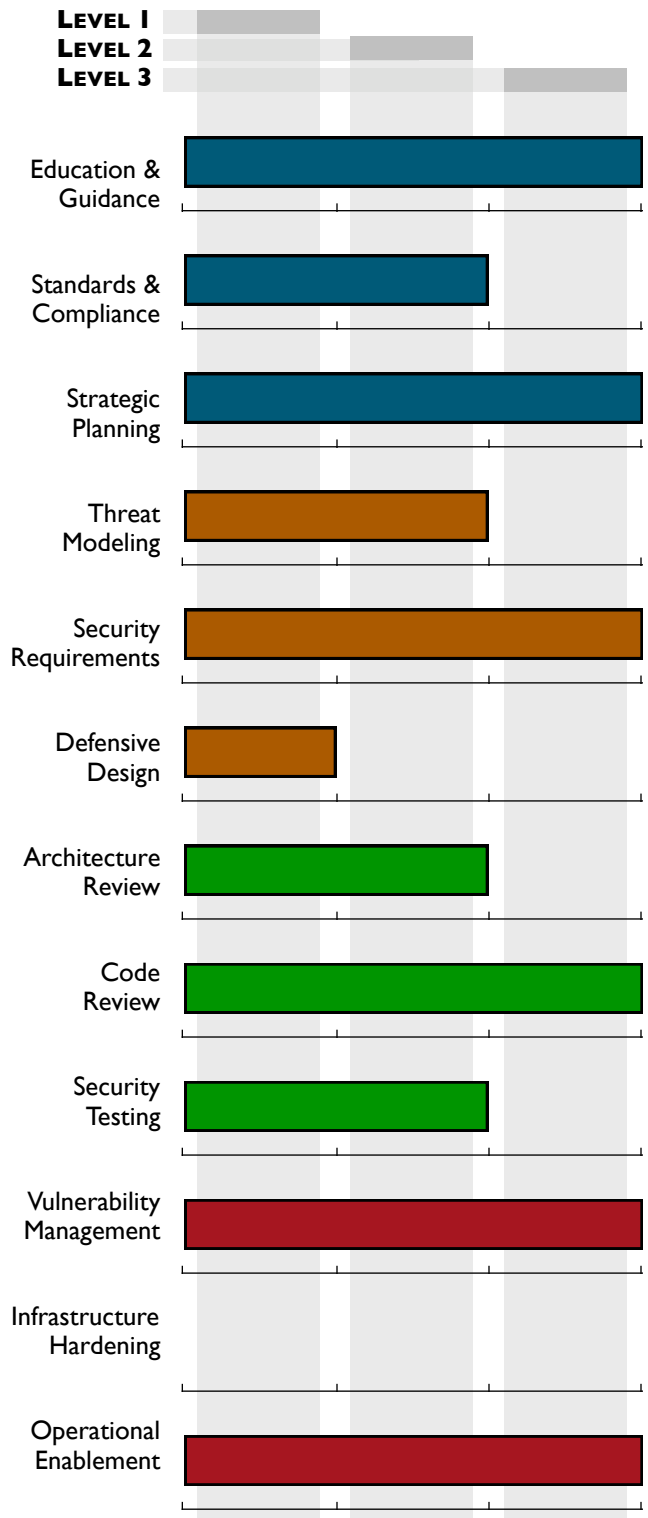
- ◆ Business Owners and Team Leaders are aware of the risk associated with their applications and are required to sign-off on applications before release;
- ◆ Team Leaders now require all applications to formally go through the security process, and code reviews are performed weekly by developers;
- ◆ Ongoing yearly training and education programs (including CBT) are provided to all project staff and developers are required to attend a course at least once a year;
- ◆ A dedicated Team Leader for Application Security has been created, and is now responsible for customer communications, and customer technical papers and guidelines.

Going forward Virtualware now has a culture of security being a part of their SDL, thus ensuring that applications developed and provided to customers are secure and robust. An effective process has been put in place where vulnerabilities can be reported on and handled by the organization when required.

During the final implementation phase a project gap assessment was performed to identify any weaknesses that appeared during the implementation. In particular due to the high-turnover of staff, Virtualware needed to constantly train new developers as they started with the organization. A key objective set to address this problem was an induction program to be introduced specifically for developers so that they receive formal security training when they start with the organization. This will also help to set the mindset that security is important within the organization and its development team.

MATURITY SCORECARD

The maturity scorecard was completed as a self assessment during the implementation of the Software Assurance Program by Virtualware. The final scorecard (shown to the right) represents the status of Virtualware at the time of the end of phase four of this implementation.



[This page intentionally left blank.]

[This page intentionally left blank.]

[This page intentionally left blank.]