Collaboration in a Secure Development Process Part 1

Gunnar Peterson

Overview

You have to ask yourself: why is there so much bad code? How many times have you heard the statement to the effect of: "if only the developers had built these applications with security in mind, we wouldn't have these security problems?" At the same time, the developers may say: "We built to spec, give us security specifications and we will make it happen!"

Do developers want to write insecure code? No. Do security teams want to deploy insecure programs or be engaged at the zero hour before deployment? No. Who suffers? The business suffers due to poor and late information on how to manage its risk profile. What is missing from the current development and security relationship is a way to engage security in the development lifecycle at the proper time making security a participating stakeholder in the software analysis and design process.

The purpose of this article is to examine specific ways that the security and development teams can collaborate while software is being designed and developed as opposed to only patching software once it has been deployed. Software is extremely malleable in the design and development phase, once the architectural layers, tiers, and distribution models are set and the application is deployed, then the cost and complexity of making changes rises dramatically.

In order to participate in an enterprise software development process effectively the security team requires a new tools and techniques. This article examines analysis and design activities, which the security team can use to improve the security in the code built by their enterprise.

Enterprise Development Process Context

Large enterprises are heterogeneous entities. They are host to a wide variety of technology, processes, and skill sets. In this article, we will seek to focus on elements, which are commonly used in or adapt well to use in enterprise development. Since the security team does not generally arbitrate the choice of a development methodology, we will not focus on a specific methodology; rather we will look at ways to integrate into development process. We will examine four phases in a sample hypothetical development process, which we will call a *Secure Development Process* or SDP. SDP is an iterative development methodology whose constituents bear some resemblance to activities and artifacts found in *Rational Unified Process* (RUP) and *Agile* processes. The four phases in our SDP example process consist of:

- *Analysis Phase*: this phase is geared towards problem definition, requirements gathering and analysis
- *Design Phase*: this phase consists of iterating through designs, proof of concept work, and refining requirements
- *Development Phase*: this phase is characterized by programming and unit testing code
- *Deployment Phase*: the deployment phases promotes the code into production operation

Note that these phases are iterative in nature as opposed to a waterfall approach. Analysis does precede other phases, but typically Analysis artifacts and activities will be revisited throughout the development life cycle, i.e after Design prototyping. Design phase activities and artifacts are likewise updated and refined throughout the development process, especially in conjunction with the Development phase. The phase breakdown in our sample methodology is not to impose a hierarchy or chain of events, but rather to demonstrate relative areas of focus and engagement over the course of the development lifecycle. While most applications spend the vast majority of their lifespan in an operational/maintenance state, our focus here will be on securing the building phases so that the application is in a more defensible position when it is promoted to the operational/maintenance status.

Thinking Iteratively

Modern development processes like XP and RUP utilize an iterative approach to development. That is, the development phases are designed to proceed without waiting for "perfect" requirements. The common process is for the development team to gather an initial set of requirements, then proceed to design and build an initial application. The business analysts and other stakeholders then refine requirements and validate the development work on an ongoing basis. This approach is in contrast to a waterfall process where all requirements and features are frozen before development begins.

Since in reality requirements generally evolve over time even in a waterfall process, iterative development reflects a practical way to deal with this. It is worth noting that the mind set required for iterative development is different from the "classic" IT security mind set which tends be driven more by policy definition and compliance than ongoing tradeoff analysis in a development life cycle.

Security Architecture: Building a Shared Understanding of Risk

The Security Architect's goal is to first build a shared understanding among disparate stakeholders so that the team can define a design, which meets the functional as well as the security-focussed requirements. The shared understanding is fostered through education, communication and mentoring.

Shared understanding of security's definition and role in the application is a prerequisite to garnering effective and thorough security requirements. To build the shared understanding of security in the context of an enterprise development project, the Security Architect's process flow should include these activities:

- Ensure development staff are well versed in security mechanisms. Developers need to be able to participate in the analysis and design of security mechanisms, so it is necessary for the developers to be up to speed with security elements and concepts, for example authentication, authorization, and secure exception management. A kick-off seminar at the beginning of the software development project can be an effective way to educate development personnel.
- Mentor security team on the development process, its organization, and technologies. To be on point with communications, the security team needs to be proficient in the technologies, and to understand through the lens of the development process what analysis and solution issues are dealt with when and by whom. Proactive involvement in the development process is a key to avoiding the last minute security code review scenario.
- Find allies! A common problem with security involvement in the development process is that it frequently becomes an "us against them" situation. The security team is not for or against development, it is for protecting the business' assets and ensuring secure development of features. Where possible the security team should seek out enterprise architecture teams, business, and legal teams and harness their support to help gain adoption of security mechanisms in the development process. This paradigm triangulates the communication so that security is seen more in an organizational context and less in a group A versus group B way.
- Get business buy in. The business decides the resources and time-lines a project is given. Educating the business side on the exact nature of what the security mechanisms are protecting, how they impact time-lines and budgets, and where they fit into the overall process is an important part of the early stakeholder engagement process.

Phase Activity	Standard Software Development Process Artifact	Security-specific artifact
Analysis	Use Case	Misuse Case
	Functional and non-functional requirements	
	Glossary	
Design	Object modelling	Threat Model
	Design Patterns	Data Classification
		Security Integration Design
Coding	Unit Tests	Unit Hacks
	Code Development	Countermeasure and detection development
Deployment	Build and configuration	Security Baseline
	Operational processes	Response processes
		Integration to Overall Security Architecture

Table 1 – **Software Development Activities and Artifacts**

Integrating Security into the Enterprise Development Process

No widely adopted enterprise development process has security as a first principle. Enterprise development processes are typically designed to navigate the space between business-requested features, programming, timing, and budgetary realities. These two factors are then rounded out by a variety of "ility"-type requirements like scalability, performance, and security. The result for the Security Architect is that they must work to bend existing process artifacts to a security viewpoint and create new artifacts, which advocate more clearly for security goals.

A pragmatic way for the security architect to approach the software development life cycle is to first look at what already exists in the process, which can be leveraged for security purposes. Once the sum total of these existing artifacts and activities are gapped against the security program's goals additional activities and artifacts are then defined to supplement the process. Wherever possible the security team should leverage existing development artifacts and activities, this aids in gaining adoption. However, some security concepts, for example threat modelling, will not be supported in the default development process, so additional activities will need to be synthesized into the development process.

In summary, the integration of security into the development process begins by finding what already exists in the development process and how it can be leveraged for security's goals and by whom. Then a top down architectural view is used to identify and address gaps between the existing process inventory and the security team's policies and goals.

While enterprise software development processes do not contain everything that the security team requires, they are an excellent point of leverage to drive security mechanisms and goals. Utilized well, the development process provides a solid foundation from which the security team can base further security initiatives. At the same time, Table 1 shows that for the security team to participate as a partner in the development process, there is a sharp up-tick in their workload.

Analysis Phase

"A problem, properly stated, is a problem on its way to being solved," Buckminster Fuller

The Analysis Phase consists of pulling together and analysing requirements. The requirements define what the system is supposed to do, and on whose behalf. During analysis activities the development team concentrates on the "what" rather than the "how". The Analysis Phase is comprised of identifying the business goals of the system and understanding the problem space at a detailed level so that design may begin, and that code may later be validated against the inception criteria. Typically, code is not developed in this phase, but prototype and proof of concept work may start at this early stage. The artifacts that emerge from the analysis activities drive much of the future priorities and resources. This point underscores the need for the

security team to get involved early on in the development project.

Several artifacts in the Analysis Phase, properly utilized, can be beneficial to clarifying security goals and role. The main standard artifacts in the Analysis Phase and their security implications include:

Use Cases
Use Cases provide a well-defined format to document application requirements within a certain context. Use Case documents define the major pieces of functionality, the system's behaviours, the overall flow (and alternate flows), the system boundaries, the Pre and Post Conditions of the system, and its Actors.

Use Cases do not explicitly deal with Non-functional requirements. Security is typically classed as a Non-functional requirement, but there are several reasons why it is valuable for the security team to participate in the Use Case process. Reviewing Use Cases is one way for the Security team to identify security risks. Understanding the various states within the application flows, and the Pre and Post Conditions of the Use Case can illuminate ways for the Security team to refine their architecture.

Perhaps the most valuable aspect of the Use Case format is that they are designed to be inclusive. Many security problems emerge from the stovepipe view of development where communication gaps occur between business, development and security groups. Use Cases provide a way to understand and manage the relative tradeoffs each group must consider to work towards a solution, which can balance the needs of all the stakeholders.

Each Use Case is a text document, Use Case diagrams can be drawn to show contextual relationships between Use Cases, boundaries and actors. Since Use Cases are designed to be simple they are an inclusive way to build a common understanding of what the application is supposed to do (and from a security point of view: not do). While a full description of the Use Case format is out of scope for this article, there are several points in Use Cases, which are worth paying extra attention to for the Security Architect.

- *Pre-Conditions*: describes the state that the system is required to be in before action starts in the Use Case. In particular, assumptions concerning authentication, authorization, and key access should be defined in this area
- *Post-Conditions*: details the state(s) which the system can be in at the end of execution. The security implications for Post-Conditions include ensuring that the application properly cleans up after itself, for example closing connections and releasing resources. Additionally, the fail open versus fail closed question can be answered in the Post-Condition section
- *Alternate Flows*: show the possible alternate flows the application can take on account of abnormal behaviour, i.e. an exception. Exception handling and management are of great interest from a security standpoint. Each exception case and its post-conditions should be examined from a security perspective to ensure that the system reverts to a known good state and that the user session is reset as appropriate
- *Actors*: Actors are analogous to users. Use Cases are more user-centric than standard requirements. They define not only what the system does, but what it does based on which Actors are using the system. User-centric does not mean that all users are humans however; actors can be individual users or other systems. Actors are an important part of the Use Case because they show how the system performs its functions for each user type. Actors are the best starting point to begin to understand role definition in a role-based or group-based security model
- *System Boundaries*: show the boundaries of trust for the application from a security perspective. Care should be taken to ensure that the related trust models like the network security model map from a logical to physical model for the target system.

Use Cases also define the stakeholder community. Stakeholders can include not just the users of the system, but many other groups can be thought of as stakeholders, including: development team, security team, customers, and business sponsors. By defining the stakeholder landscape each representative voice can advocate for their specific goal and a more balanced view of the business value and commensurate risks can be modelled. This balanced Use Case Model is an essential component in understanding the relative merits of features, usability, time to market, and other quality factors in a security context.

A last important point to remember about Use Cases is that as recursive as it sounds, Admins are users too! Administrator Use Cases are an important source of information for how the system will be maintained over time and the administrator use case definition process can uncover some hidden assumptions and security flaws. To learn more about Use Cases, consult [1]. Alistair Cockburn's work is also a very valuable resource in this area [2].

Requirements
Requirements come in two flavours: functional and non-functional. Since not all requirements fit into the Use Case format, the Use Cases are supplemented by additional requirements documents. The non-functional requirements documents in particular are geared towards defining security requirements. Defining these require-

ments early on and being specific about precisely what the security requirements are at a system level is essential to getting a consistent security design into the application. At a minimum the non-functional requirements should defined the requirements around Authentication, Authorization, and Confidentiality before the design phase begins in earnest.

The fundamental keys to writing security requirements are to be as specific as possible, and to aim to make the requirements testable and measurable. By writing specific and measurable requirements, the development process will clearly show where the application is providing sufficient or deficient security mechanisms.

Since security is at least partially measured in negative terms, i.e. "the system must not allow unauthorized usage of its functionality", it is a more subtle art to capture the security requirements. The value of identifying the requirements is that this allows the security requirements to be managed and prioritized alongside the other requirements the development team deals with, such as features, usability, and performance. A big part of the early stages in the development process is to find the balance between business valuable factors which may be incongruent, for example security may wish to require an extremely long password which expires every month, while the usability team will find this unacceptable.

The challenge to the security team is to find a way to attach business value to its requirements, i.e how does it impact risk? What is the price of the risk to the business? Feature-driven requirements are seen in terms of the business value they provide to the enterprise. Security-driven requirements are in large part best seen by how they mitigate risk. The key here is that the security team must be as prepared to show the detailed business case for security as the team, which drives the features is. Once the relative merits of the solutions are defined an effective tradeoff analysis can occur.

Glossary
Domain glossaries were born out of application development in complicated business domains. Security keywords and definitions should be added to the glossary. The glossary functions as an important educational resource for the business and development teams to learn about security concepts and mechanisms.

Security-Specific Artifacts

Misuse Cases
Developed by Guttorm Sindre and Andreas Opdahl, Misuse Cases look at the system from the malicious attacker's point of view. From a security viewpoint, Misuse Cases are an excellent balance to Use Cases since they correlate threats, targets, and damage to functionality and legitimate use.

As in the instance of Use Cases, Misuse Cases are primarily text documents with a supplementary graphical model. Like an attacker, Misuse Cases invert the expected course of events and leverage trust to gain unauthorized access to resources.

- *"A Misuse Case is the inverse of a use case, i.e. A function that the system should not allow"* -Sindre & Opdahl.

  Misuse Cases give tangible instances of the system's main threats, and map those threats onto the functionality (Use Cases), the system boundaries, and the other affected system relationships with external systems, Actors, and Use Cases

- *"A Mis-actor is the inverse of an actor, i.e., an actor that one does not want the system to support, an actor who initiates misuse cases."* -Sindre & Opdahl.

  Mis-actors can be broken down into specific types, for example to show attacker skill level or to show internal versus external types of attackers.

Misuse Cases use much of the Use Case terminology, and define their own additional constituents, including:

- *Worst Case Threat*: the end system state if Misuse Case "succeeds"
- *Prevention and Detection Guarantees*: analogous to a Use Case Post-condition, but encapsulate security-specific mechanisms.
- *Stakeholders and Risks*: addresses business risk that is generated by the application's misuse.

Misuse cases define the starting point for more advanced threat modelling. They can be used to glean more precise security requirements, and lead into testing scenarios and Unit Hacks (more on these in the next article). Misuse Cases exist in relationship to the functionality described in Use Cases.

To learn more about Misuse Cases review [3]

Use Cases and Misuse Cases Example
The Use Case and Misuse Case formats complement each other, and provide a balanced perspective of functionality and risk. What follows is a brief illustrative example of a simple Use Case and Misuse Case for the fictitious company Foo Co.

Use Case:	**Update Product Price**
Actors	Standard Foo Co User Foo Co Product Specialist

Pre-condition	User must be authenticated and authorized to perform operation Session timeout is set
Post-condition	Product price is updated Audit log is sent message to record update action User session is expired or user explicitly logs out
Basic Flow	Foo Co Product Specialist views Product selection, and selects a product to update price Product Specialist selects an available price from market pricing schema Product Specialist updates price, and confirms update Standard Foo Co Users can now see the updated product price for quotation

Note that in a real Use Case multiple alternative flows will augment the Basic Flow to show what course of action the Use Case takes in an exceptional event.

Misuse Case	**Mis-price Product**
Mis-Actor	Malicious Insider
Pre-condition	Mis-Actor must be authenticated and authorized to perform logon operation Mis-Actor must gain unauthorized access to pricing levels Session timeout is set
Post-condition	Product price is set at below recommended level Product price is updated Audit log is sent message to record update action User session is expired or user explicitly logs out
Basic Flow	Malicious Insider views Product selection, and selects a product to update price Malicious Insider gains unauthorized access to pricing minimum level Malicious Insider selects a price, which is below market pricing schema standard Malicious Insider updates price, and confirms update Standard Foo Co Users now see the updated "incorrect" product price for quotation
Worst Case Threat	Products sold at below recommended pricing bottom line negatively impact company profit margins

Value of Use Cases and Misuse Cases Working Together

This simple example shows several things, which are valuable to the security and development teams. First the Misuse Case shows which controls are essentially ineffective in this instance. If the user can log on, and the authentication and authorization mechanisms for the application are not consistent with the mechanisms for the pricing the system is at risk. All of the other security mechanisms can perform their functions, but the system will fail and Foo Co will lose money.

Taking the example one step further we add the *Prevention and Detection Guarantees* to the Misuse Case. The Prevention and Detection Guarantees can take the form other Use Cases which are included or extend the behaviour of existing Use Cases.

Prevention Guarantee: Each access to the minimum pricing system are authenticated and authorized against the same user repository as the main Update Product Price Use Case.

Detection Guarantee: After updating Product Price off line processes read the recorded price set in the audit log and correlate it against the day's minimum pricing levels. Alerts are generated for prices, which fall below the minimum.

Identifying Misuse Cases

Generally, Use Cases are initially identified by holding a meeting where the primary stakeholders who know the domains brainstorm together to build a first cut list of Use Cases, which are further refined and edited over time. An initial list of Misuse Cases can emerge from a similar setting. Engage the stakeholders who understand the domain, the application serves, business risks, security assumptions, user community, and technology stack to define the initial set of Misuse Cases.

Since many Misuse Cases are related to technologies (i.e. IIS Web Server can have its own set of genericized Misuse Cases) or deployment patterns (i.e. 3-Tier Web-based application), over time the Security Team can reuse the Misuse Case Models. This is important since it reduces the impact of additional involvement in new projects for the security team once the initial models are done. Note that the involvement number will not approach zero since work is required to map the Misuse Cases to the domain and the threat landscape is constantly changing over time.

Conclusion

Early involvement in the development life cycle ensures less security impact as the application is deployed. The keys to success for the security team's participation in the development life cycle is to understand what elements exist that can be leveraged for security purposes and where there are gaps to be filled with additional security-centric artifacts and activities. The end result of the collaboration between the security, development and business teams is a richer shared understanding of the business risks, security mechanisms, which mitigate them, and where opportunities for improvement exist.

In the next article, we will build upon the Analysis Phase and describe collaborating in the Design, Coding and Deployment phases in our development process from a security standpoint.

References

[1] Kurt Bittner and Ian Spence, *Use Case Modelling*, Addison-Wesley, 2003

[2] http://alistair.cockburn.us/

[3] http://www.ifi.uib.no/conf/refsq2001/papers/p25.pdf

Additional Resources

Development Process
IBM Rational Software
http://www.ibm.com/software/rational/

XP Methodology
http://www.extremeprogramming.org/

Craig Larman's web site is an excellent starting point for process and modelling books, articles and links
http://www.craiglarman.com/

Use Cases
Ivar Jacobson, *Use Cases- Yesterday, Today and Tomorrow*
http://www.jaczone.com/papers/use_cases-2002-11-26.pdf

Misuse Cases
Guttorm Sindre and Andreas Opdahl, *Templates for Misuse Case Description*
http://www.ifi.uib.no/conf/refsq2001/papers/p25.pdf

About the Author

Gunnar Peterson is CTO of Arctec Group, a focussed, best-in-class IT consulting provider whose primary service is delivering pragmatic, objective, vendor-independent management and architectural consulting services for business-critical systems.

http://www.artechgroup.net