Collaboration in a Secure Development Process Part 2

Gunnar Peterson

Overview

In the first part of this series we examined the software development life cycle and specific ways that security can participate in the development process as a proactive stakeholder. We explored the *Analysis* phase in detail showing how security can be a meaningful partner in the analytic activities of software development rather than a passive policy-making body. In this part we will trace the software development life cycle through the *Design* phase activities looking for what artifacts and processes already exist in common development methods today. We will also consider gaps that exist in the development process and how to address these.

In the first part of the series we identified a hypothetical development process called the *Secure Development Process* (SDP). The SDP has four main phases:

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

In this part we will harvest the work from the Analysis Phase and put it to use in the following three phases.

Bookstores are filled with weight loss guides, yet there are many overweight people. Identifying the "right"thing to do is one part of the solution, but active, ongoing commitment and execution are the keys to seeing it through to a successful implementation. The Security team should partner with the development team throughout the development life cycle to achieve the right blend of security, features, and time to market for the enterprise.

There are many security tasks with regard to locking down development, testing, and Quality Assurance servers. These are important tasks, however this series focusses on the collaboration within the design of the application, not the security context that governs development.

Tradeoff Analysis in the Design Phase

The Design Phase of SDP garners the resultant analytic artifacts produced in the Analysis Phase. These artifacts include *Use Cases*, *Requirements*, *Glossary*, and *Misuse Cases*. The artifacts are updated and refined throughout the Design Phase as proof of concept work. Prototyping, requirements change, design work, and, yes, feature creep inevitably conspire to date the accuracy of

the analysis as the development rolls along. The Analysis Phase artifacts are grist for the security tradeoff analysis in the Design Phase. The Analysis Phase artifacts lend specificity to the design discussions so that the key system stakeholders can have an informed debate as the relative merits of security versus features versus time to market rather than oblique references to XYZ technology being classified as "secure"or "insecure."

During the Design Phase, business requested items begin to emerge as features that the system will support. Each feature has been previously defined in Use Case documents in the Analysis Phase. The development team uses the Design Phase to elaborate on the Use Case specifications, and builds a more detailed picture of the requested features. The development iteratively updates the Use Case documents to reflect what is deemed feasible in the development life cycle, and works on initial designs to implement the features. As the designs materialize, they are very fungible for the security team to assess their impact to security and to begin to assess viability of tradeoffs where applicable.

Drilling Down on Security Design

The Design Phase builds upon the work done in the Analysis Phase by detailing initial conceptual solutions, which address the identified and prioritized business problems. As the Design activities progress, the Analysis artifacts are iterated on and updated, proof of concepts are built to assess and manage architectural risk factors, assumptions are tested, and designs are clarified. The design phase is generally a very collaborative exercise within the development team. There are three main types of artifacts that come out of the Design Phase found in typical enterprise development methodologies today.

Unified Modeling Language (UML)

UML "helps you specify, visualize, and document models of software systems, including their structure and design, in a way that meets all of these requirements" according to OMG (http://www.omg.org/uml).

In practice, UML provides a standard set of models that the development team can use to collaboratively design object oriented solutions for business problems.

UML defines a set of models that blueprint the software model. Like a blueprint, the models are visually rich and understandable for non-technical team members, and at the same time specific enough to be useful for developers.

UML is not tied directly to any specific programming language. UML models can be realized in any object oriented programming language. In fact, some vendors like IBM Rational and Borland have tools that generate code from UML Models into popular enterprise programming languages like Java.

Software Architecture

The Software Architecture describes the key elements in the system, and their structure and relationships. Typically, this is defined in a Software Architecture Document or SAD. The SAD is comprised of a number of views that depict key system concerns from a number of different perspectives. There are many types of Software Architecture including IBM Rational's 4+1, the Zachman framework, Reference Model for Open Distributed Processing, and OMG's Model Driven Architecture (MDA).

The purpose of architecture is to consider the system from a number of different perspectives or views. The architectural views enable the architect to show all the primary characteristics in a complex system and their relationships to each other and to external systems. Each architecture type defines their own type of views and attributes. For example, the architect in an enterprise application concerned with a system-wide property like security will also need to factor in impacts across the system's servers, network, databases, application and web servers, and other infrastructure. Software architecture's chief function is to provide a mechanism whereby the team architecture can be comprehended and reasoned about by the enterprise and its stakeholders. For more information on Software Architecture, consult the Resources section at the end of this article.

What makes software architecture valuable from a security standpoint? First, the combination of the architectural views are an important building block for the security team, because the architecture views and documentation have the necessary breadth to both illuminate security risks and to educate the security team about the system's scope. Secondly, many security solutions, for example Single Sign On, are horizontally focussed and exist across a set of applications and supporting infrastructure. So the architectural views are the basis to begin to model security solutions.

The security team can also have a good opportunity to partner with the architecture team. By collaborating with the architecture team to drive security solutions the security team can have a very valuable ally. The architecture team can help to triangulate the classic "development versus security"dynamic through education, policy, and design.

UML Object Models

During Design, developers use UML Object Models to show the key structural and behav-

ioral parts of the application. Like Use Cases, UML Models are an abstraction, which facilitate inclusive debate and discussions among the development team and its stakeholders. UML Models bridge the gap between Analysis artifacts, such as Use Case and Requirements, to the application code. UML Models depict the objects attributes, methods, and interaction points. Generally speaking UML Models show structure or behavioral characteristics. Some important UML Models are:

- *Class Diagrams*: these static models are intended to show the structure, navigability and relationships of the application's classes. Class diagrams include the Class name, its key attributes, methods, and its relationships with other classes. Class diagrams can help to identify the interaction points, i.e. methods, between classes, but not the order of execution.

  The Class diagram example (Figure 1) shows the structure of the payment types that the systems implements. The CreditCard and Cheque classes inherit from the abstract class Payment. These classes inherit the structure and behavior defined in Payment so that both CreditCard and Cheque contain *amount* and *currencyType* attributes and an *authorizePayment* method. The CreditCard and Cheque classes implement their own unique attributes and methods in addition to the inherited characteristics.

- *Sequence Diagrams*: describe the sequence of operations between classes and the input and return payload messages from each operation.

  The Sequence diagram illustrates the behavior of the program. In this example (Figure 2), a web user logs in and is authenticated by the system. The authenticator service returns a validSessionToken (such as a cookie), and the client is then able to execute the *viewProduct* call. The Sequence diagram is a good way to "see" where the authentication and authorization services should be placed. Note that the Sequence diagram will not necessarily show all of the internal operations inside a given class, getting the "right" level of detail is part of the skill of the modeler.

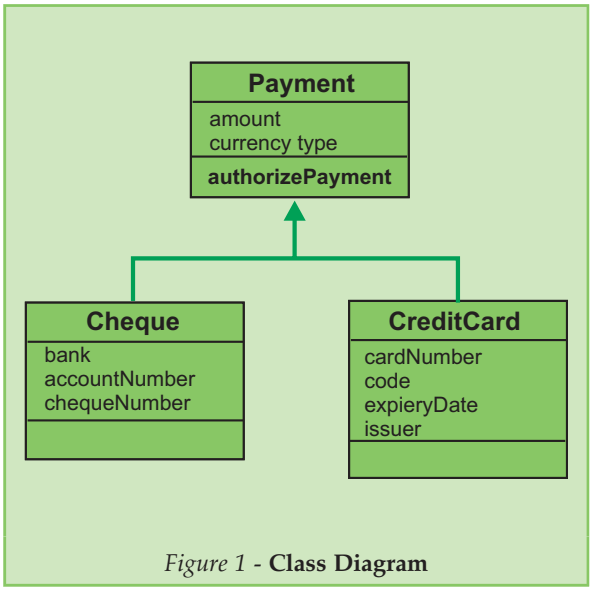

Figure 1 - **Class Diagram**

- *Deployment Diagrams*: these diagrams plot the logical models and packages onto Physical hardware and nodes. Deployment diagrams typically show interfaces, processing nodes, connectivity, and protocols.

There are many UML Models to consider. A full description of them is out of scope for this article, but consult the Resources section for more pointers.

So what can the security team learn from UML Models? The first thing that the security team can learn is the domain design knowledge, which is distilled into the UML Models. UML is an excellent way to grasp the key parts of an application without reading every single line of source code. Decomposing the application into separate slices that represent structural and interaction views is useful to derive design patterns and visualize the application for design and review purposes.

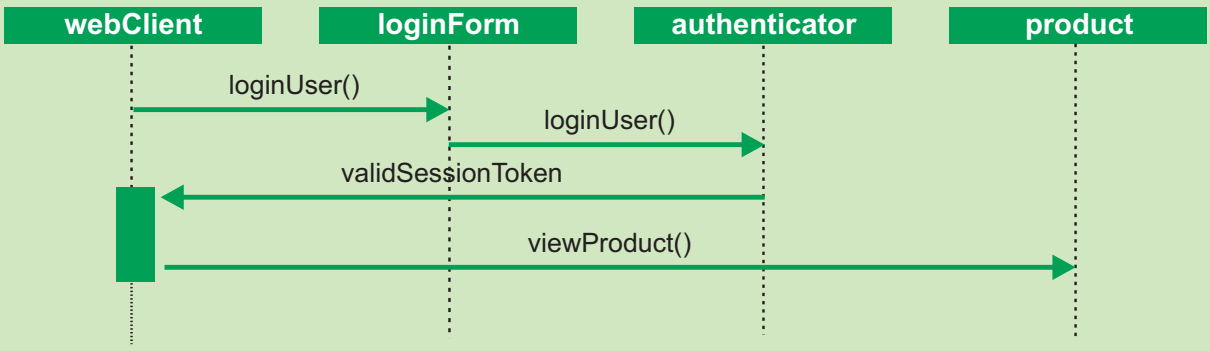

Figure 2 – **Sequence Diagram**

Reviewing Object Models are a good way for the security team to find areas of interest to focus on in the application, for example:

- *State Changes*: There are many security implications to consider when state changes. When sensitive data's state is changed it may be important to write information to the audit log to reflect this change. In a user management system a state change in an user attribute implies that user's role and attributes may need to be refreshed in that instance to avoid incorrect or escalated privileges. State change information is typically seen in Sequence, Interaction, and Activity diagrams.
- *Context Change/Trust Relationship Chain*: Changing context for example when a call to the web server is propagated to the application server and to the database server, is a critical part of the security view. The key decision points here typically center around choosing between impersonation or delegation [1]. The chain of events that generate the call sequences are typically seen in the Sequence, Interaction, and Activity diagrams. The physical implications of this chain are manifested in the Component and Deployment diagrams
- *Persistence*: Generally speaking, when data is persisted a trust handoff has also occurred. The data leaves the domain, which the developer controls and enters a domain controlled by DBA or Sys Admin. The security team's job is to identify these persistence points to determine that the trust model is consistent in the persisted environment as it is in the application environment. For example, if an application writes certain classified data to a log file on the file system, is that log file protected to the same degree as the application that generated it?
- *Access Points/Privilege Change Requirements*: Access points, authentication and authorization events are visible in design models. These provide a point of reference for the security to ensure that the security assumptions made in the Analysis phase requiring access levels for functionality usage are implemented in the design.
- *Identify Inputs*: Design Models show the points in the application where data is entered as well the expected structure of the data to be input. From a security team standpoint reviewing the Design Models facilitates the understanding of where the data input validation should be placed and the content that should be allowed or not allowed to be input.
- *Understand Exception Scenarios*: the application will not always travel the happy path, particularly when a malicious user is involved. Exception handling and management are key to understanding how resilient the application will be in the face of attack. What actions are taken in the Try..Catch block for example to fail open or fail closed? Once the exception is caught, what mechanisms are in place to both log and manage the exception and to recover in the program?
- *Roles*: Most of the list above focuses on what can be learned through the behavioral aspects of the application. Once the chain of operations and methods is understood, the security team needs to work towards a coherent set of roles from which to design the authorization system.

UML has another important advantage for the security team in that it is not technology specific, UML Modeling can be applied to .Net, COM, Java, J2EE, Python, and many other languages and programming paradigms. This is extremely useful to the security team who may support a large development organization with heterogeneous sets of languages and technologies.

This section just scratched the surface of the type of security relevant information that is visible in UML Models. In short, if the development team is doing UML, there are a lot of valuable security data points waiting to be harvested; and if the organization is UML savvy then the security team can provide useful design level guidance for a variety of technologies and languages.

Design Patterns

Once the object landscape is defined, design patterns can begin to be identified. Design Patterns were born out of "real world"architecture, specifically the work of the architect Christopher Alexander [2]. Alexander defined a pattern as:

As an element in the world, each pattern is a relationship between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain spatial configuration which allows these forces to resolve themselves.

As an element of language, a pattern is an instruction, which shows how this spatial configuration can be used, over and over again, to resolve the given system of forces, wherever the context makes it relevant.

Alexander's architectural concepts were brought into focus in the software development world through the seminal book *Design Patterns* [3]. The authors of Design Patterns defined three types of patterns, which occur in applications:

- *Creational Patterns*: are responsible for constructing objects, creating their initial instance, and managing the lifespan of the object
- *Structural Patterns*: define objects composition in terms of its interfaces, structure and relationships
- *Behavioral Patterns*: describe the ways that the objects communicate, process logic, handle state, and interact with other objects

Additionally, there are now numerous pattern books and web sites that apply the Design Pat-

terns concept to specific technology implementation like .Net and J2EE. As with UML Models understanding pattern types and how they are used leads the security team to a richer understanding of the application space and better leverage to design security mechanisms. The security team can utilize the pattern format to create its own Security Design Patterns. For example, the security team can define pattern sets like Access Patterns to manage common types of authentication and authorization, and Confidentiality Patterns to manage privacy and cryptographic concerns and implementations. The pattern approach is another tool, which already exists in many enterprise development environments and can be a boon to the security team. The challenge to the security team is to find the balance of specificity, which helps developers deliver quality on time and reusability, which reaches across the enterprise.

Initial integration designs are a part of most design phases, common types of integration include applications, which access databases, ERP, CRM, and legacy systems.

Participating in the Design Phase

The recurring theme throughout the development life cycle should be for the security team to identify what exists in the current development process that can be leveraged for security purposes. Then work to identify gaps where a parallel set of documents and activities need to be introduced and integrated into the development process.

Security-Centric Artifacts

The above set of artifacts commonly found in enterprise development processes provides leverage for the security team to both understand and influence the security in the development effort. Building upon the security-centric Analysis artifacts we examined in Part 1, there are a number of security-centric Design level Artifacts that enable the security team to add value through fine and coarse grained design techniques.

Threat Models

Threat Models elaborate on Misuse Cases defined in the Analysis Phase. Threat Models distill security risks into a coherent set of models, which the development team can use to help design the proper level of security controls and to achieve the right risk profile for the application. Threat Models assist the development team in seeing both where the threats come from, how they may be executed, what conditions are necessary for them to occur, and what the impacts could be. In the SDP lifecycle, Threat Models bridge the gap between Misuse Cases that define threats at an analysis level, Countermeasure development, and Testing through systems testing and Unit Hacking, which validates what attacks the system is resilient to.

The Threat Modeling process consists of:

- *Threat Identification and Classification*: the first step in Threat Modelling is to identify what the threats are that face the application. In the SDP, we have a head start due to the work done in the Analysis phase on Misuse Cases. Each Misuse Case should be correlated to one or more Threat Models. Since the application is evolving throughout the life cycle, there may be new security risks that were not apparent in the Analysis phase and new security risks appear on a daily basis as a fact of life, so Threat Models are not generated 100% out of the Analysis phase.

  STRIDE is a classification system that categorizes threats as *Spoofing, Tampering, Repudiation, Denial of Service*, or *Elevation*. The main difference of Threat Modeling as opposed to traditional software development artifacts is to see the application from an attacker point of view. STRIDE enables a structured way to categorize the relevant threats by attacker goals.

- *Threat Impact*: Once the threats are identified the security team should work to categorize the impacts of the threats if successful. The impact assessment is a valuable part of the trade-off analysis exercise in that it gives the security team a bargaining chip with a validated tangible business impact. The business value of the Threat Impact should be contrasted with the business value of the feature, or feature set, which generates it.

  DREAD is a system that classifies risks by *Damage, Reproducibility, Exploitability, Affected Users*, and *Discoverability*. These are all relevant factors in assessing threat impact. The security team should ensure that the threat impact assessment uses similar economic and business factors to show risk as the business teams uses to show business value so that the information is congruent across teams and viewpoints.

- *Threat Prioritization and Countermeasure*: Once the threats are refined, the security, business and development teams need to define a prioritization and scoping of what is realistic in the development time-line and budget. As the threats are prioritized in scope, each Threat Model requires one or more countermeasures which provide security mechanisms to defend or detect against it.

  Countermeasures can be implemented at an application level through code, design, or administrative methods. Or the threat can not be explicitly defended against at all from a protection standpoint, in this case the security and development teams develop signatures that can be plugged into detection systems. While the more detailed countermeasure develop-

ment exists in the following coding phases initial decisions for how to handle threats such as the "Defend or Detect?"question are handled in this phase.

- *Threat Test Cases*: Threat Test Cases distill the threat down into a set of conditions, which constitute a threat, for example a SQL injection attack through a web form. These will typically be implemented in the life cycle as scripts or test cases. The Test Case also describes the expected response for the system. The Test Case gives the development team an important psychological target to aim at and to achieve the elusive "how to know when you are done."

In a large enterprise a general Threat Model framework should be developed, for example a B2B application Threat Model. This generic framework can be more easily managed and updated to include the relevant threats. Then the teams can focus more resources on understanding the application-specific impacts and risks. For more information on STRIDE and DREAD please see Microsoft's new Threat Modelling Centre on MSDN [4].

Data Classification

Data classification is the process of categorizing data by relevant business and security factors to determine what levels of controls must be applied to protect the data. Just as the business analysts decompose features and functionality into Use Cases and Actors, application data should be analyzed and categorized. Classification assists in designing and applying Role Based Access Control and integrating the application and database security models. Data Classification is particularly important for enterprise applications that integrate multiple disparate data sources since the application must support a coherent role structure that spans multiple discrete security models. Note that data classification is not limited to databases and ERP systems, but that any persistence store like flat files and configuration files should be classified as to its security and business sensitivity.

Initially, the data's business and security classification categories can be set, based on the following factors:

- *Business Value*: What is the value of the data to the enterprise? To what extent and business units and operations impacted if the data is mishandled or unavailable?

- *Threats*: Threat Models are in many cases tied in the end to some data element. Use Threat Models to determine what is the threat likelihood and what is their impact on customers or company?

- *Confidentiality*: Is the data public, proprietary, personal, or company private? Is the data subject to business or industry specific confidentiality controls?

- *Business Unit Ownership*: Sometimes organization structures can provide clues as to ownership of data elements. The business unit, which controls the data source, may have business roles and functions that are required to support which may further imply classification structures to be analyzed.

- *Third Party Control*: While data may be "owned"by business units it may in some cases be subject to controls by third parties, examples include regulatory bodies and trading partners. The nature of the third party control will generally drive a separate classification level and in the case of more complicated relationships (as in a trading partner network) and regulations a number of levels of classification specific to the relationship or regulation.

At an implementation level, the end result of the Data Classification activity is to understand what types and levels of data classification are required to drive security controls in both the application and the controls supporting the persistence mechanisms. At a logical design level the Data Classification provides a way to correlate the application's roles with the data classification levels.

Security Integration Design

No enterprise application is an island. Each application's functionality and security model exist in relation to some set of other business processes, applications, or systems. Security Integration Design is the act of building the detailed view of security integration points, and its mechanisms and assumptions. The fundamental questions to be answered in Security Integration Design are all about trust. The main factors to consider in the Security Integration Design are to understand what other systems are you trusting to vouch for what security controls. For example, does your application perform its own authentication or does it rely on Active Directory or an external LDP server? How are those credentials passed and protected?

To begin with, the security team should have a view, which shows all of the integration points of the application. This includes data flows, users, and supporting systems. While it may not be practically feasible to audit the internals of all the systems that the applications interoperates with, a detailed understanding of the nature of the integration, and the security mechanisms which protect is an important part of the overall security view. There are many different types of integration and each has it own set of security concerns, some examples include:

- *Data Flow Integration*: How is the data transferred? How is the data payload protected?

How is the connection and entry point protected? What is the exception management process in the event of failure? What system has control over the authentication, authorization, and confidentiality of the integration point? A related point is the control flow integration. Beyond simply data flow integration, what is the exact nature of the handshake between the two systems and which system has control when?

- *User Integration*: What assumptions does your application make about the identity and security context, i.e. authentication and authorization, of it users?
- *Supporting Systems*: What trust model are the servers and other supporting systems operating under? How does this affect the security model with respect to the application?
- *Trusted Systems*: The security team should inventory the systems it trusts for security mechanisms such as key servers, user repositories, and identity management systems.

Conclusion

The Design phase Is typically the most creative part of the development process. Security and development teams should seek sets of artifacts, which clearly articulate and put the design questions and tradeoffs on the table to have a useful and high bandwidth conversation with stakeholders. This article examined some ways to generate this using traditional and non-traditional artifacts. Security teams have to deal with a variety of projects horizontally spread across the enterprise, whereas development tams and business units are typically vertically focussed. Wherever possible the security team should leverage solutions which have generic qualities and reach as in the case of Threat Models and Design Patterns. In the next article we will consider the Coding and Deployment phases in the SDP.

References

[1] Keith Brown, *The .NET Developer's Guide to Windows Security*, Pearson Education, 2004

[2] Christopher Alexander, Sara Ishikawa, Murray Silverstein, *A Pattern Language: Towns, Buildings, Construction*, Oxford University Press, 1977

[3] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides *Design Patterns: elements of reuable object-oriented software*, Addison-Wesley 1995

[4] http://msdn.microsoft.com/security/securecode/threatmodeling/default.aspx

Resources

Architecture
Dana Bredemeyer's Resources for Software Architects
http://www.bredemeyer.com/

UML
Cetus Links
http://www.cetus-links.org/oo_uml.html

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.
www.arctecgroup.net