Asian Journal of Information Technology 6 (8): 902-906, 2007
ISSN: 1682-3915

Software Process Improvement Through Secured Development Lifecycle

[1]S. Chitra and [2]M. Rajaram
[1]M. Kumarasamy College of Engineering, Karur-639 113, India
[2]Department of EEE, Thanthai Periyar Government Institute of Technology, Vellore, India

Abstract: This study discusses the software process improvement through Secured Development Lifecycle (or SDL), a process adopted for the development of software that needs to withstand malicious attack. The process encompasses the addition of a series of security-focused activities and deliverables to each of the phases of software development process. These activities and deliverables include the development of threat models during software design, the use of static analysis code-scanning tools during implementation and the conduct of code reviews and security testing during a focused "security push".

Key words: SDL, TSP, deployment, verification, KSL

INTRODUCTION

This study assumes that there is a central group within the company (or software development organization) that drives the development and evolution of security best practices and process improvements, serves as a source of expertise for the organization as a whole and performs a review (the Final Security Review or FSR) before software is released. The existence of such an organization is critical to successful implementation of the SDL as well as to improving software security. While, some organizations might consider having the "central security team" role performed by a contractor or consultant. This study describes the integration of a set of steps intended to improve software security into the software development process that is typically used by large software development organizations. These steps have been designed and implemented by Microsoft as part of its Computing Initiative. The goal of these process improvements is to reduce the quantity and severity of security vulnerabilities in software used by customers. In this document, the modified software development process, which is currently being implemented at Microsoft, is referred to as the Computing Software Development Lifecycle (or simply the SDL).

TEAM SOFTWARE PROCESS AND SECURITY

In this study, the design principles of Security Team Software Process can be categorized into 4 phases (Humphery, 2000):

- Secure design process.
- Secure implementation process.
- Secure review and inspection process.
- Secure verification process.

The goal of this effort is to develop a process that:

- Supports secure systems development practices.
- Predicts the likelihood of latent security defects.
- Can be dynamically tailored to respond to new threats.

Design principles for secure applications: The principles for producing secure application are also well known and easy to understand:

- Authorize and authenticate all users.
- Mistrust all user input.
- Encrypt sensitive data from login to logout.
- Protect persistent data.

PROBLEMS

Stakeholder's issues: A number of ways users can inhibit requirements gathering (Humphery, 1994):

- Users don't understand what they want.
- Users wont commit to a set of written requirements
- Users insist on new requirements after the cost and schedule have been fixed.
- Communication with users is slow.
- Users often do not participate in reviews or are incapable of doing so.
- Users are technically unsophisticated.
- Users don't understand the development process.

Corresponding Author: S. Chitra, M. Kumarasamy College of Engineering, Karur-639 113, India

This may lead to the situation where user requirements keep changing even when system or product development has been started.

Engineer/developer issues: Possible problems caused by engineers and developers during requirements analysis are (Bazier, 1990).

Technical personnel and end users may have different vocabularies. Consequently, they can believe they are in perfect agreements until the finished product is supplied.

Engineers and developers may try to make the requirements fit an existing system or model, rather than develop a system specific to the needs of the client.

Engineers or programmers, rather than personnel may often carry out analysis with the people and the domain knowledge to understand a client's needs properly.

ATTEMPTED SOLUTIONS

One attempted solution to communications problems has been to employ specialists in business or system analysis.
Techniques such as (Humphery, 2002):

- Prototyping.
- Unified Modeling Language (UML).
- Use cases.

MAIN TECHNIQUES

Conceptual requirements analysis includes 3 types of activity:

Eliciting requirements: The task of communicating with customers and users to determine what their requirements are.

Analyzing requirements: Determining whether the stated requirements are unclear, incomplete, ambiguous, or contradictory and resolving these issues.

Recording requirements: Requirements may be documented in various forms, such as natural language, use cases, or process specifications.

Requirements analysis can be a long and arduous process during which many delicate psychological skills are involved. New system change the environment and relationships between people, so it is important to identify all the stakeholders, take into account all their needs and ensure they understand the implication of the new systems. Analysts can employ several techniques to elicit the requirements from the customer. Historically,

This has included such things as holding interviews, or holding focus groups and creating requirement list. More modern techniques include prototyping and use cases. Where necessary, the analyst will employ a combination of these methods to establish the exact requirements of the stakeholders, so that a system that meets the business needs is produced.

Stakeholder interviews: Stakeholder interviews are a common method used in requirement analysis. Some selection is usually necessary, cost being one factor in deciding whom to interview. These interviews may reveal requirements not previously envisaged as being with in the scope of the project and requirements may be contradictory. However, stakeholder shall have an idea of his expectation or shall have visualized his requirements.

Requirement workshops: In some cases it may be useful to gather stakeholders together in "requirement workshop". These workshops are more properly termed Joint Requirement Development (JRD) sessions, where requirements are jointly identified and defined by stakeholders. It may be useful to carry out such workshop in a controlled environment, so that the stakeholders are not distracted. A facilitator can be used to keep the process focused and these sessions will often benefit from a dedicated scribe to document the discussion. Facilitators may make use of a projector and diagramming software or may use props as simple as paper and markers. One role of the facilitator may be to ensure that the weight attached to proposed requirements is not overly dependent on the personalities of those involved in the process.

Contract-stylerequirement lists: One traditional way of documenting requirements has been contract style requirement lists. In a complex system such requirement list can run to hundreds of pages.

Measurable goals: Best practices take the composed lists of requirements merely as clues and ask why, repeatedly, until actual business purposes are discovered. Then stakeholders and developers can devise tests to measure what level of each goal has been achieved so far. These goals change more slowly than the long list of specific but unmeasured requirement. Once the small set of critical, measured goals has been established, rapid prototyping and short iterative development phases may proceed to deliver actual stakeholder value long before the project is half over.

Prototypes: Prototypes are mockups of the screens of an application, which allow users to visualize the application that isn't yet constructed. Prototypes help users get an idea of what the system will look like and make it is a for users to make design decisions without waiting for the system to be build. Major improvement in communication between the users and developers were often seen with a introduction of prototypes. Early views of the screen led to fewer changes later and hence reduced over all cost considerably.

However, over the next decade, while proving a useful technique, it did not solve the requirement problem:

- Managers once they see the prototype have a hard time understanding that the finished design will not be produced for some time.
- Designers often feel compelled to use the patched together prototype code in the real system, because they are afraid to 'waste time' starting again.
- Prototype principally helps with design decisions and user interface design. However, they can't tell you what the requirements were originally.
- Designers and end users can focus too much on user interface design and too little on producing a system that serves the business process.

Prototypes can be flat diagrams or working application using synthesized functionality. Prototype are made in a variety of graphic design document and often remove all colors from the software design in instances where the final software is expected to have graphic design applied to it. This helps to prevent confusion over the final visual look and feel of the application.

Use cases: Use cases are a technique for documenting the potential requirements of a new system or software change. Each use case provides one or more scenarios that convey how the system should interact with the end user or another system to achieve a specific business goal. Use cases typically avoid technical jargon, preferring instead the language of the end user or domain expert. Use cases are often co-authored by software developers and end users.

Use cases are deceptively simple tools for describing the behavior of the software. A use case contains a textual description of all of the ways, which the intended users could work with software through its interface. Use cases do not describe any internal workings of the software, nor do they explain how that software will be implemented. They simply show the steps that the user follows to use the software to do his work. All of the ways that the user interacts with the software can be described in this manner.

Each use case focuses on describing how to achieve a single business goal or task. From a traditional software engineering perspective, a use case describes just one feature of the system. For most software projects, this means that perhaps tens or sometimes hundreds of use cases are needed to fully specify the new system. The degree of formality of a particular software project and the stage of the project will influence the level of detailed required in each use case. A use case defines the interactions between external actors and the system under consideration to accomplish a business goal. Actors are parties outside the system that interact with the systems; an actor can be a class of users, a role users can play, or another system.

Use cases treat the system as a "black box" and the interactions with the system, including system responses, are as perceived from outside the system. This is deliberate policy, because it simplifies the description of requirements and avoids the trap of making assumption about how this functionality will be accomplished.
A use case should (Humphery, 2002):

- Describe a business task to serve a business goal.
- Be at an appropriate level of detail.
- Be short enough to implement by one software developer in a single release.

Use cases can be very good for establishing functional requirements, but they are not suited to capturing Non-functional requirements.However,4 performance engineering specifies that each critical use case should have associated performance oriented non-functional requirements. Software requirements specification.

A Software Requirement Specification (SRS) is a complete description of the behavior of the system to be developed. It includes a set of use cases that describes all of the interaction that the users will have with the software. Use cases are also known as functional requirement. In addition to use cases, the SRS also contains non functional requirements. Non-functional requirements are requirements which imposed constrains on the design or implementation.

Stakeholderidentification: It is increasingly recognized that stakeholders are not limited to the organization employing the analyst (Jones, 2004). Other stakeholders will include:

- Those organizations that integrate horizontally with organization the analyst is designing the system for
- Any back office system or organizations.
- Senior management.

SECURE BY DEFAULT

In the real world, software will not achieve perfect security, so designers should assume that security flaws would be present. To minimize the harm that occurs when attackers target these remaining flaws, software's default state should promote security. For example, software should run with the least necessary privilege and services and features that are not widely needed should be disabled by default or accessible only to a small population of users.

SECURE IN DEPLOYMENT

Tools and guidance should accompany software to help end users and/or administrators use it securely. Additionally, updates should be easy to deploy.

COMMUNICATIONS

Software developers should be prepared for the discovery of product vulnerabilities and should communicate openly and responsibly with end users and/or administrators to help them take protective action (such as patching or deploying workarounds).

The first two elements secure by design and secure by default provide the most security benefit. Secure by design mandates processes intended to prevent the introduction of vulnerabilities in the first place, while secure by default requires that the default exposure of the software its "attack surface" be minimized.

RESULT ANALYSIS

Windows Server (2003) was the first operating system release at Microsoft that implemented large portions of the SDL. Figure 1 shows the number of security bulletins issued within the year after release for the two most recent Microsoft server operating systems: Windows (2000) and Windows Server (2003). When Windows (2000) was released, Microsoft did not have a formal security bulletin severity rating system. Microsoft has evaluated each security bulletin that applies to Windows (2000) against its current severity rating system.) As has been discussed earlier in this study, Windows Server (2003) was developed with most (but not all) the SDL processes.

The security of a software-intensive system is directly related to the quality of its software.

- Over 90% of software security attackers exploiting known software defects cause incidents.

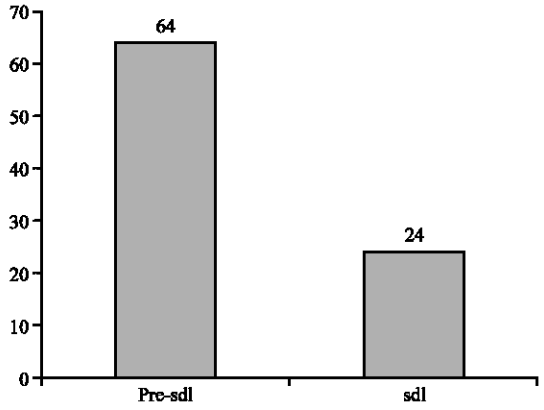

Fig. 1: Windows pre- and post-SDL critical and important security bulletins

- Experienced and capable software engineers
- Inject, on average, one defect every nine lines of code.
- A one million line of code systems typically contains 1,000-5,000 defects when shipped.

TSP fosters good practices based on engineering principles.

With TSP, software teams

- Build detailed, accurate plans
- Manage and track their commitments
- Produce nearly defect-free software (<0.1 defects/KSLOC)

Software produced with TSP has one or two orders of magnitude fewer defects than current practice.

- 0.02 defects/KSLOC vs 2 defects/KSLOC
- 20 defects per MSLOC vs. 2000 defects per MSLOC

If 5% of the defects are potential security holes, with TSP there would be 1 vulnerability per million SLOC.

CONCLUSION

The secured development lifecycle is effective at reducing the incidence of security vulnerabilities. Initial implementation of the SDL (Windows Server 2003; SQL Server, 2000; Service Pack 3 and Exchange, 2000; Server Service Pack 3) resulted in significant improvements in software security and subsequent software versions, reflecting enhancements to SDL, appear to be showing further improvements in software security. The

development and implementation of the Secured development lifecycle represent a major investment for Microsoft and a major change in the way that software is designed, developed and tested. The increasing importance of software to society emphasizes the need for Microsoft and the industry as whole to continue to improve software security.

REFERENCES

Boris, B., 1990. Software Testing Techniques. (2nd Edn.), pp: 78-143.

Humphery, W.S., 2000. Introduction to Team Software Process. Pearson Edu. Singapore, pp: 27-196.

Humphrey, W.S., 1994. Process Feedback and Learning. The 9th Int. Software Process Workshop, Arlington, pp: 232-250.

Humphrey, W.S., 2002. Managing the Software Process, Pearson Education, Singapore, pp: 300-323.

Jones, 2000. Capers. Software Assessments. Benchmarks and Best Practices, Addison-Wesley, pp: 165-189.

Viega, Jones and McGraw, Gary, 2001. Building Secure Software Building Secure Software. Addison Wesley, pp: 124-168.