Software security, Lecture 3

Ismo Aulaskari

Helsinki 9.4.2014
Lecture notes
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos
Lecture notes from the third lecture "Security in a software project" by Henri Lindberg and Antti Vähä-Sipilä. The software security practise is described in relation to software development process models and the reality in the business-driven field. Tools that help secure the process are also introduced.
Sisältö

1 Security in the software process 1

2 Formalized software security 2
   2.1 Security process models .......................... 2
   2.2 Certifications .................................... 3

3 Security versus the software process 5

4 Tools for testing and testing automation 6
   4.1 Code analysis tools ................................ 6
   4.2 Security testing tools .............................. 6
   4.3 Operational tools .................................. 6

Lähteet 7
1 Security in the software process

There are many different kinds of software process models. The waterfall model is heavy on specification and produces a result later at the end. Agile models can instead produce many iterated results and focus on rapid delivery. Security testing happens late in the process, trying to find problems in an existing product. However, security should be part of the process in every phase from the beginning to the end as early mistakes are the most difficult to fix [AVS14]. Design time, implementation and operation of a product have different needs [GW03].

The process categories can be further refined and described as follows [AVS14]:

**Product management and requirements engineering** create functional requirements as in (security) features and non-functional requirements as the required quality of features. This part is also about avoiding bad designs as well as providing enough time for security engineering.

**Design** involves doing architectural risk analysis (threat model) and the creating of an attack model.

**Development** is securitywise about avoiding design and implementation bugs and using security features correctly.

**Building** of the software contains adding security analysis and testing to automated builds.

**Testing** includes different architectural levels of verifying that security functionality works as well as reacting to bugs that should stop deployment.

**Deployment** can have deployment automation to speed up security updates. Also, may include training of users and operators.

**Operations** encompasses keeping the environment of the product secure and monitoring it for signs of attack.

**Support** for the software should not forget operating a security contact point, tracking known vulnerabilities concerning the product and feeding the information into development.

**Decommissioning** is the final category, concerning keeping the data that should be kept and securely erasing the rest.
2 Formalized software security

There is no single way to attempt to secure a software process. Organisations have had different sizes and ways of working and so have implementations of security processes been adapted, localized as well as extended to fit the needs of local software production. There are lightweight and heavy processes and mandatory and self-service audits.

2.1 Security process models

There are many formal models for the fulfillment of the needs described above. Some that are known are BSIMM, OpenSAMM, ISO 27034 (based on Microsoft SDL, it contains a standard the vocabulary and concepts of the software process), PA-DSS payment application standard and the Finnish govermental VAHTI specification.

In picture 1 the BSIMM Software Security Framework’s twelve main categories for software security practises grouped in four distinct domains. 112 different security-related operations have been categorized under the twelve labels.

<table>
<thead>
<tr>
<th>The Software Security Framework (SSF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Governance</td>
</tr>
<tr>
<td>Strategy and Metrics</td>
</tr>
<tr>
<td>Compliance and Policy</td>
</tr>
<tr>
<td>Training</td>
</tr>
</tbody>
</table>

Kuva 1: the BSIMM Software Security Framework [Bs14].
In picture 2 the Microsoft Security Development Lifecycle used as process guidance for all their software development [AVS14].

What is the Security Development Lifecycle?

The Security Development Lifecycle (SDL) is a software development process that helps developers build more secure software and address security compliance requirements while reducing development cost.

Design Phase

SDL Practice #5: Establish Design Requirements

Considering security and privacy concerns early helps minimize the risk of schedule disruptions and reduce a project's expense.

SDL Practice #6: Attack Surface Analysis/Reduction

Reducing the opportunities for attackers to exploit a potential weak spot or vulnerability requires thoroughly analyzing overall attack surface and includes disabling or restricting access to system services, applying the principle of least privilege, and employing layered defenses wherever possible.

SDL Practice #7: Use Threat Modeling

Applying a structured approach to threat scenarios during design helps a team more effectively and less expensively identify security vulnerabilities, determine risks from these threats, and establish appropriate mitigations.

Kuva 2: Microsoft SDL [Ms14].

2.2 Certifications

Security thinking could be known in-house, or it might be bought from outside in form of consultation. There are also certifications sold that certify vendors, products and/or people to have some kind of understanding of security processes. Certification is about meeting the minimum criteria for something, a thing hard to communicate otherwise. Human-supervised practical certification models are more expensive than ones that can be automated like multiple choice-based exams. One should be aware of the depth and method of certification behind a certificate to determine its
value, however its still common for players in the security business to have certain certifications to present [AVS14].
3 Security versus the software process

Securing a software development process is difficult in real life. Businesses are in the software business to make money and the monetary needs usually come first. A product has a budget from which the security work tries to get its funding and a timetable which the security process threatens and which leaves little time for finding and fixing problems. The worst software is often bought so cheaply it makes security assessment seem too expensive in comparison.

As the number of agile projects increases, it poses new problems as security testing should be continuous when doing continuous integration. The traditional waterfall process is still most common and its problem is the unpredictability of the end result. This leaves very little time for security testing if the product is late for release and even less time to fix problems and verify the fixes.

Outsourcing worsens the waterfall problems as the development is usually even less controlled. Certifications tell little about the morale of the people involved, even if security problems are identified it might be too much trouble, too expensive or too late in the project to do something about them [HI14].
4 Tools for testing and testing automation

The trend in security tools is to add automation and integration into deployment builds to create a secure continuous deployment. Following are the major types of security tools for software development and some explanation each category of tools [AVS14]:

4.1 Code analysis tools

Code analysis can be done by static analysis to non-executing code or by dynamic analysis to running programs. Of static analysis tools linters do mainly syntactic analysis and some also code complexity analysis. Static data flow analysis can often find security bugs better by tainting input and tracking variables and execution paths. These tools are language-dependent.

Dynamic analysis tools monitor running programs, detecting memory leaks and usage, also being useful for performance monitoring. Dynamic analysis is most useful in low-level programming where memory management is up to the user, but profiling has its uses in any application.

4.2 Security testing tools

Security testing tools include many kinds of attack and scanning tools and sniffers. A robustness testing tool generates broken data that is fed to the tested application, in an attempt to break it. Sniffers and protocol analysers intercept network traffic for viewing. Attack proxies place themselves between a client and the server making analysing and manipulating the intercepted data easy. Security scanners inspect the software configuration and environment.

4.3 Operational tools

Operational tools think about protecting the application from the world and also protecting the underlying system and perhaps the network from the application. Some tools aid security by isolating applications and untrusted code by virtualisation and related technologies. Another approach is firewalling attack data from an application that for some reason has vulnerabilities that are not immediately fixed.
Lähteet


