Comparison of SDLC and OWASP

İlgün İlgün

Development processes for software construction are common knowledge and mainstream practice in most development organizations. Unfortunately, these processes offer little support in order to meet security requirements. In this essay, two high-profile processes for the development of secure software, namely OWASP’s CLASP and Microsoft’s SDL, are evaluated and compared in detail. The essay identifies the commonalities, discusses the specificity of each approach, and proposes suggestions for improvement.

1 Introduction

The construction of secure software is still largely a matter of guidelines, best practices and undocumented expert knowledge. Current practices provide guidance for particular areas such as threat modeling, risk management, or secure coding. It is crucial for these to be combined into an integrated and more comprehensive construction method. Several advances have recently been made in the definition of processes for secure software development. However, there has been no objective comparison of these methodologies so far. Therefore, it is difficult for managers and developers to appreciate their strengths and counter their weaknesses. Consequently, it is hard for the various stakeholders to make an ‘informed’ decision about which one is more appropriate for the job.

The focus of this essay is restricted to two forefront representatives, namely Microsoft’s Security Development Lifecycle (SDL) [8] and OWASP’s Comprehensive, Lightweight Application Security Process (CLASP) [12], as they are recognized as the major players in the field. Their leading role is, among others, due to a number of characteristics that are of particular interest in the context of this study. As far as completeness is concerned, they both provide an extensive set of activities covering a broad spectrum of the development lifecycle. The processes also have undergone validation. For instance, Microsoft is using SDL internally (e.g., for the Vista project) and CLASP was contributed to and reviewed by several leading security companies of the OWASP consortium. Moreover, SDL and CLASP represent a healthy mix for comparison: the former is considered to be more heavyweight and rigorous, making it more suitable for large organizations. CLASP, on the other hand, is lightweight and more affordable for small organizations with less strict security demands. Finally, the selection was also influenced by the availability of thorough documentation.
This essay contributes by theoretically evaluating and comparing CLASP and SDL in different ways: (i) generic characteristics of the two processes are discussed in order to outline the philosophy underlying a particular method; (ii) activity wise, the intersection and delta of each method are identified and discussed in order to articulate the strong and weak points of the methods and (iii) a number of improvements are outlined that both processes could benefit from, in terms of both coverage and quality.

2 Methodology

This section briefly sketches the methodology that was used for the comparison. For CLASP I decided to use the documentation of version 1.2 available at the OWASP web site. For SDL web site was used.

SDL activities are already organized into stages, which have been mapped into the above-mentioned phases using extra information provided by MSDN. As far as CLASP is concerned, activities have been assigned to phases by looking at the role that is responsible for the activity itself.

This information was inserted in a matrix, which contains a section for each of the phases listed above, with each row representing a certain activity and each column representing one of the approaches. Figure 1 shows a fragment of the matrix, focusing on the ‘Project Inception’ phase. The matrix was subsequently used to drive the comparison and analysis of the two candidate approaches, e.g., to identify the intersection and the differences between the two.

3 Background and General Characteristics

In this section, the two processes are further introduced and a number of characteristics are discussed in order to describe their overall philosophy.  

CLASP. Originally defined by Secure Software [1] and later donated to OWASP, CLASP is a lightweight process for building secure software [12]. It includes a set of 24 top-level activities and additional resources, which can be tailored to the development process in use. Key characteristics include:

Security at the center stage: The primary goal of CLASP is to support the construction of software in which security takes a central role. The activities of CLASP are defined and conceived primarily from a security-theoretical perspective and, hence, the coverage of the set of activities is fairly broad.

Limited structure: CLASP is defined as a set of independent activities that have to be integrated in the development process and its operating environment. The choice of the activities to be executed and the order of execution are left open for the sake of flexibility. Moreover, the execution frequency of activities is specified
per individual activity, resulting in complex coordination. Two roadmaps (‘legacy’ and ‘green-field’) have been defined to give some guidance on how to combine the activities into a coherent and ordered set. However, this is not an integrated process.

<table>
<thead>
<tr>
<th>Project Inception Phase</th>
<th>Activity</th>
<th>SDL</th>
<th>CLASP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2.1. Build security</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.1.1. Build security team</td>
<td>✓</td>
<td>□</td>
</tr>
<tr>
<td></td>
<td>2.1.2. Assign security advisor</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>2.1.3. Institute accountability for security issues</td>
<td>□</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>2.2. Determine whether the application is covered by methodology</td>
<td>✓</td>
<td>□</td>
</tr>
<tr>
<td></td>
<td>2.3. Initial security</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.3.1. Provide tools to track security issues</td>
<td>✓</td>
<td>□</td>
</tr>
<tr>
<td></td>
<td>2.3.2. Determine the bug bar</td>
<td>✓</td>
<td>□</td>
</tr>
<tr>
<td></td>
<td>2.4. Monitor security metrics</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.4.1. Identify metrics to collect &amp; identify how they will be used</td>
<td>□</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>2.4.2. Institute data collection and reporting strategy</td>
<td>□</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>2.4.3. Periodically collect and evaluate metrics (ongoing during entire lifecycle)</td>
<td>□</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>2.5. Institute rewards</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>2.6. Build a global security policy, if necessary</td>
<td>□</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Figure 1.** Excerpt of the matrix.

Role-based: CLASP defines the roles that can have an impact on the security posture of the software product and assigns activities to these roles. As such, roles are used as an additional perspective to structure the set of activities. Roles are responsible for the finalization and the quality of the results of an activity.

Rich in resources: CLASP provides an extensive set of security resources that facilitate and support the implementation of the activities. One of these resources is a list of 104 known security vulnerabilities in application source code (e.g., to be used as a checklist during code reviews).

**Microsoft SDL.** As a result of its commitment to trustworthy computing proclaimed in 2002, Microsoft defined the SDL to address the security issues they
frequently faced in many of their products. SDL comprises a set of activities, which complement Microsoft’s development process and which are particularly aimed at addressing security issues. SDL can be characterized as follows:

Security as a supporting quality: The primary goal of SDL is to increase the quality of functionality-driven software by improving its security posture. Security activities are most often related to functionality-based construction activities. For instance, threat modeling starts from architectural dependencies with external systems, while architecture could in fact reduce such threats in the first place. SDL is designed as an add-on to the software construction process.

Well-defined process: The SDL process is well organized and related activities are grouped in stages. Although these stages are security specific, it is straightforward to map them to standard software development phases. Furthermore, several activities have a continuous character in the SDL process, including threat modeling and education. As such, the SDL process incorporates support for revising and improving intermediate results.

Good guidance: SDL does a good job at specifying the method that must be used to execute activities, which, on average, are concrete and often somewhat pragmatic. For instance, attack surface reduction is guided by a flow chart and threat modeling is described as a more detailed sub-process. As a result, the execution of an activity is quite achievable, even for less experienced people.

4 Phase-by-phase comparison

In this section, a phase-by-phase comparison is presented. For each of the phases both the intersection and the delta between the studied approaches are discussed. The intersection contains everything that both approaches cover, while the delta highlights the differences between them. As an example, Assign security advisor (2.1.2) in Figure 1 is a candidate to be part of the intersection, while Provide tools to track security issues (2.3.1) and periodically collect and evaluate metrics (2.4.3) are candidates for the delta of SDL and CLASP respectively.

4.1 Education and Awareness

Intersection: Both approaches acknowledge that education is very important. Everyone on the team should, at least, receive initial education, which consists of two parts. First, everyone needs to be made aware of the importance of security. Second, everyone needs education about the basics of security engineering which includes teaching the basic security concepts, types of security breaches, possible solutions, and so on.
As security is a rapidly evolving field, with new threats emerging frequently, periodical education is crucial for everyone involved in the project. Both approaches include this activity, e.g., in the form of an annual training.

Delta: There are some differences between the two approaches when it comes to education. First, SDL focuses on training the developers. CLASP acknowledges that training should include all project roles, e.g., including the project manager. The courses should cover fuzz testing, threat modeling, code reviews and so on.

4.2 Project Inception

Intersection: Both methodologies acknowledge that building security into your project requires the assignment of a security advisor to the project itself. The advisor helps the developers with security related issues and possibly serves as a gateway between developers and a dedicated security team (if available).

Delta: There are several differences between the two approaches. First, CLASP has an extra focus on security metrics, while SDL does not. Specific metrics are a valuable tool to enforce accountability of security issues or to detect the security weaknesses in the project. Within CLASP, this activity covers the identification of the metrics to collect, the institution of the data collection and reporting strategy and the periodical collection and evaluation of the identified metrics. Note that this last activity actually crosscuts the different phases of the software development process.

Second, CLASP emphasizes the role of the global organizational policy. It is suggested to develop such a document, which should be used as a baseline for security requirements of software projects. CLASP provides a list of global security requirements to be used as a template for the global policy. During inception, the appropriateness of each global requirement must be evaluated to determine the coverage of the global policy for the project at hand.

SDL, in its turn, emphasizes the importance of organizational support for secure software development by defining a number of roles (or teams) and, more importantly, laying out how they should interact. Examples of such roles include the development team security contact, the security advisor, the company-wide security team and the security leadership team. The responsibilities of the latter are (i) regular communication to the development team about security and privacy bug counts and (ii) communication of security and privacy policy updates.

Moreover, SDL includes two initial security activities: one to make sure that the tools are able to track security issues, e.g., they include special security-related fields, and the other to define the bug bar, i.e. what type of bugs will be fixed.

Finally, SDL contains an explicit activity to decide whether or not the project is covered by the methodology. This decision is based on several questions such as whether the application is designed to be used on-line. This is how ever a minor step in our opinion.
4.3 Analysis

Intersection: This might be a striking observation, but the intersection for this phase is empty. SDL seems to have no activities that are specific to the analysis phase.

Delta: CLASP focuses on identifying trust boundaries and specifying security requirements. Trust boundaries denote where trustworthy and untrustworthy entities interact. The security requirements specification is important both to discover security issues early on in the development lifecycle and to guide several security-specific construction activities later on.

4.4 Design

Intersection: Performing rigorous and thorough threat modeling is possible at this point, as one has a clear understanding of the architecture that will be built. Both approaches acknowledge its importance and cover it by means of a set of activities.

Delta: CLASP focuses on defining security architecture, a security-augmented version of the software architecture, which implies annotating the class design with security properties. It also considers the fact that most projects will be using third-party components. To mitigate the security risks involved, CLASP includes an activity that deals with researching and assessing the security posture of technology solutions.

SDL includes a product risk assessment activity, which aids in determining the best way to spend development resources by identifying the system’s level of vulnerability to attack. The results are based on questions such as:

‘On which operating system will the software be installed’,
‘Does it include ActiveX controls’, ‘is it a new project’ and so on?

4.5 Implementation, Testing, and Verification

Intersection: An interesting observation here is that SDL lacks real implementation activities. Again, this is probably due to the fact that SDL focuses more on security as a supporting quality.

Both methodologies put a lot of focus on secure testing but they have a different emphasis. SDL focuses more on black box testing, while CLASP emphasizes more on white box testing. A typical black box test is to feed the application with random input in order to observe the system re-action for unexpected failures (which are hints of possible security bugs), while an example
of white-box testing is inspecting the code for potentially insecure input processing and then feeding targeted input to the application accordingly.

Concerning verification, both methodologies include the scrubbing of the attack surface as an important activity, which is aimed at preventing an attacker from taking advantage of potentially insecure code.

Delta: SDL has two additional testing activities: (i) the security push and (ii) the final security review. Both perform extra testing but have a different focus. The security push implies testing the entire system by the project team with specific focus on legacy code, while the final security review implies testing the entire system by the central security team. Both focus on the entire system, while secure testing (see 'intersection') focuses on individual components. It is worth noting that, for the security push, SDL does cover white box testing in the form of code review.

CLASP, on the other hand, has two additional activities: (i) integrating security analysis into source management and (ii) implementing and elaborating resource policies and security technologies. The first deals with automating implementation-level security analysis and metrics collection through the use of dynamic and/or static analysis tools. The second deals with the actual implementation of the security requirements, among others by making sure that all coding guidelines are met.

4.6 Deployment and support

Intersection: Even though both approaches organize this phase quite differently, they largely end up with the same result. First, there is a need for operational planning and readiness, which includes the writing of user manuals, documenting the security architecture, and so on. Second, there needs to be a response planning that defines what to do when a new vulnerability is discovered. Both approaches also acknowledge that communication with customers is very important. One has to make sure that, amongst other things, security advisories can be released and customers have access to software updates. Finally, one needs to actually execute the plan once a new vulnerability is discovered.

Delta: CLASP puts extra emphasis on signing the code, in order to provide stakeholders with a way to validate the origin and integrity of the software.

5. Conclusions

This essay compares two high-profile development processes for secure software, in a theoretical way. The general characteristics of the processes have been described as well as the specific differences over the various development
phases, from the perspective of process activities. In summary, it is fair to say that SDL offers a well guided process that is targeted at security as a supporting software quality, while CLASP addresses security from a broader perspective and it can be flexibly tailored to the specific development environment. Apart from this theoretical evaluation, experimental assessment in concrete products will clearly provide additional validation of both approaches.

As ongoing research, the authors are working on combining the strong points of both approaches in order to distill an improved, consolidated process. This requires addressing, as well as validating, most of the areas of improvement that were discussed in the second part of the essay.

References

[17] D. Wheeler. Secure programming for linux and unix how to. citeseer.ist.psu.edu/wheeler00secure.html.