

On the Secure Software Development Process: CLASP and SDL Compared

Johan Grégoire, Koen Buyens, Bart De Win, Riccardo Scandariato, Wouter Joosen
DistriNet, Department of Computer Science, K.U.Leuven
Celestijnenlaan 200A, B-3001 Leuven, Belgium
{firstname.lastname}@cs.kuleuven.be

Abstract

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. Over the years, research efforts have been invested in specific methodologies and techniques for secure software engineering, yet complete, dedicated processes have been proposed only recently.

In this paper, 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 paper 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 paper 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 charac-

teristics 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 paper 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.

The structure of this paper is as follows. Section 2 describes the methodological approach that was adopted for the comparison. Section 3 contains a bird-eye presentation of the two processes, with a particular focus on their characterizing philosophy. Section 4 zooms into both the commonalities and the differences of the two processes, on a phase-by-phase basis. Section 5 discusses a number of challenges for the presented processes and highlights the room for improvement. Section 6 discusses related work. Finally, Section 7 presents the conclusions as well as directions for future work.

2. Methodology

This section briefly sketches the methodology that was used for the comparison. To begin, the team agreed on the

sources to be used, since possible differences and inconsistencies may exist in different versions. For CLASP it was decided to use the documentation of version 1.2 available at the OWASP web site. For SDL the book by Howard and Lipner [8] was used.

Based on the available sources, two by-the-book initial lists of activities were laid down. For some activities this required interpretation of text to elicit implicit activities. Furthermore, some clean-up was necessary on these lists. This meant optimizing the hierarchical structure of activities and similar transformations. Once the lists of activities were agreed upon, they were merged and subsequently organized into the different phases of a typical software development process ('Education and Awareness', 'Project Inception', 'Analysis', 'Design', 'Implementation, Testing and Verification' and 'Deployment and Support'). 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 is left open for the sake of flexibility. Moreover, the execution

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

Figure 1. Excerpt of the matrix.

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.

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).

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 an 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 however 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 a *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 reaction 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. Discussion and improvements

In this section we discuss a number of fundamental improvements, which both processes could benefit from.

5.1. Quality and coverage

During the analysis of activities in both SDL and CLASP, it was observed that some activities do not correspond to the state-of-the-practice definition for process activities. From a general perspective, a process involves a

temporally ordered series of activities that, starting from an input state, leads to an outcome by using a set of resources like time, and expertise.

Activities must produce added value in the form of an output, which must be clearly identified in an artifact. As a counter-example, only two activities in CLASP clearly specify the artifacts that should be produced. It is also important that the delta between input and output, the '*visible impact*', is large enough to make an activity worthwhile. If this criterion is applied to both SDL and CLASP, it can be observed that some of the construction steps are guidelines rather than real activities. For instance, the SDL activity *secure coding policy* suggests to use the latest compilers and to avoid using deprecated functions from libraries. According to the above discussion, this is more of a coding guideline than a process activity. As a positive example, CLASP addresses the impact of each of the 24 top-level activities (in the Activity-Assessment View).

According to Davenport [5], the characteristic of a business process is to focus on the business logic of the process itself (how work is done), instead of taking a product perspective (what type of work is done). That is, activities should provide a '*systematic method*' to put in practice what they suggest. It is observed that both SDL and CLASP fall short in supporting the aforementioned property. For instance, the CLASP activity *apply security principles to design* has no clear methodological indication on how to realize the activity.

Besides these quality issues, a few coverage gaps were observed, i.e., phases that have no or little support. First, activities at the design stage are mostly oriented towards detailed design. No activity at all is defined for the architectural design of secure software. For instance, there is room for specific activities in the area of trade-off analysis, e.g., security vs. usability. Second, in SDL there is very limited support during the deployment phase, as coverage is restricted to user documentation only. Further support for deployment could include the management of the security solution by enforcing operational procedures, as well as the monitoring of the security solution in order to proactively discover weaknesses. Third, in both SDL and CLASP, threat modeling focuses on architectural and technical weaknesses rather than business-level threats.

5.2. Security in context

Two main constraints must be considered when it comes to security-specific activities: the integration with an existing process and the cost of augmenting a process with security.

Concerning *integration*, many medium-to-large organizations already have a heavyweight and rigorous development process in place, e.g., according to the Unified Pro-

cess [9]. Furthermore, many of them have achieved a significant level of process maturity and have received a certification for their accomplishment. For these organizations, it is key to have a set of clear guidelines to ease up the difficult task of integrating new security-centric activities in a well-established, preexisting instance of a standard process.

For smaller organizations with a more lightweight and less rigorous process, merging the security process with the existing informal software development process is less of a problem. However, these organizations may have a non-traditional process in place, e.g., they may be using agile techniques such as Extreme Programming [4]. There is a knowledge gap with respect to the problem of integrating CLASP in agile methods. Conversely, SDL support in this direction is far more extensive.

It is impossible to build a 100% secure system, especially in day-to-day system construction where a trade-off is made between security and other constraints such as the *cost*. This will have an impact on the accuracy with which certain activities are being executed and on the set of activities that will be executed in the first place. Therefore, it would be useful to provide guidelines for selecting the set of relevant activities. At a minimum, a distinction could be made between a core set of mandatory activities and its accessory extensions. Also, an indication of the risk associated with the omission of accessory activities is useful. For instance, CLASP provides such information to some extent (in the Activity-Assessment View). However, it seems to make more sense to institute a maturity-like approach (e.g., based on SSE-CMM [2]) to indicate the desired rigor to be put in the execution of certain activities. Based on this, a number of profiles could be defined for different types of environments and applications domains. Additionally, these indications could be used to drive the assurance process of the software afterwards.

5.3. Verification

Application functionality is a positively spaced problem: functional requirements are specified in an affirmative manner (what behavior should occur) and verifying the correctness of a functional requirement focuses on the (limited) set of executions of the particular function. In clear contrast, security is negatively spaced. Security requirements typically state which behavior should not occur. Since the set of executions of a software artifact is infinite, it is much harder to cover this set in order to guarantee the correct realization of a security requirement in software. Due to the difficulty of covering the negatively spaced security requirements and due to the risk of introducing major security vulnerabilities as a result of small errors made during the software construction process, it is crucial to institute verification as an important fill rouge throughout the process, complementary

to construction activities.

Verification is in practice often closely linked to a particular construction activity, i.e. to ensure that the activity has been done right. For instance, when building a threat tree one could verify that all types of threats are covered systematically for every asset in the system. While the importance of this type of verification is clear, the authors claim that it is as important to include dedicated verification efforts both focusing on relations between activities and spanning significant sets of activities. For instance, one could cross-check the security requirements set with the operational policy (e.g., to verify that the separation of duty principle holds – see Section 5.5). In general, a security-focused process should exhibit a good balance between both types of verification.

5.4. Metrics

The software engineering practice assigns a significant role to metrics. They are a quantitative means to measure the progress of quality for the process itself and the produced artifacts. Both CLASP and (particularly) SDL have only limited focus on metrics. Two directions of improvement can be foreseen.

First, a program to evaluate the effectiveness of the security process itself must be instituted. The program should be responsible for continuously assessing both the real impact of security activities on product quality and the optimal usage of resources (e.g., time and personnel). To this aim, reference values can be useful. For instance, CLASP provides information about the time that is required to carry out activities.

Second, specific measuring activities should be included in all development phases to assess the quality of artifacts from a security perspective and at different levels of abstraction. Such measures could play a leading role in defining acceptance criteria for software artifacts during all stages of the development process. More importantly, metrics should constitute an analysis tool to identify criticalities early on, with remarkable impact on cost. To some extent, CLASP tries to pursue this direction, as a whole top-level activity is dedicated to metrics. However, the implementation of this type of measuring program is difficult. Indeed, there is not a clear understanding of which properties should be observed in order to assess security qualities. Some work exists concerning the implementation and the deployment phase [10], but significant gaps have to be filled by the research community.

5.5. Principled process

Several security principles have been enumerated in the literature [14, 16, 15]. Examples include ‘minimize attack

surface’ and ‘run with least privilege’. The value of such principles is so widely recognized that their enforcement should be explicitly assisted by the process.

Activities could be annotated in order to highlight their contribution to the realization of a specific principle. For instance, CLASP activity *map roles and resources to entry point* relates to the ‘minimize attack surface’ principle. In CLASP, principles are seldom addressed a few times, but the relationship between activities and principles is not worked out systematically. In SDL, principles are explicitly mentioned only twice. Furthermore, new constructive activities could be introduced to better support principles. Finally, some verification activities can be put in place to assess the compliance of process artifacts with principles. For instance, the SDL activity *re-evaluate the attack surface of software* contributes to the verification of the above-mentioned ‘minimize attack surface’ principle.

5.6. Process support for moving targets

It is common knowledge that security is a moving target: applications change, execution environments change, attackers change, and new (types of) bugs are found. This all influences the security posture of a software application. This evolving character affects the construction process of secure software in two ways. First, under the hypothesis that the previously articulated security assumptions remain valid, the process must include continuous support to address new security vulnerabilities after the software has been released. This is supported in both processes by including activities that focus on software updates and security advisories. Second, and more challenging, when intermediate results turn out to be incorrect (such as an incomplete threat model), or when security assumptions change after deployment, the process must be backtracked in order to correct the no-longer valid decisions and assumptions. In a process, backtracking can be supported by introducing iterative cycles, or by inserting dedicated checkpoints and feedback loops. In our opinion, this kind of support is limited in both SDL and CLASP.

In the same way, it is difficult to enforce security requirements on a moving target. For instance, if the functionality of a system changes frequently, threat modeling must be repeated every time and since many activities are based on the outcome of threat modeling, this will have several ripple effects. These effects can be minimized by choosing a limited set of locations in the process to which security activities are added.

Traceability of decisions and relations between intermediate results constitute a way to improve the support for change management. As such, one can more clearly analyze and estimate the impact of modification on the developed system.

6. Related work

To the authors' knowledge, no evaluative studies have been performed on secure software development processes. Therefore, the discussion on related work focuses on (i) related processes covering the entire secure development lifecycle, and (ii) methodologies focusing specifically on particular construction or assurance activities.

There are a number of resources that cover the entire secure development lifecycle. BSI [3] is a website maintained by the Software Engineering Institute (SEI) and sponsored by the DHS National Cyber Security Division. The website provides an overview of existing processes and methods for each development phase. These methods are not secure software methodologies, but rather tools that can be used and that only cover part of a secure software engineering methodology. In his articles, Peterson [13] discusses how a development team can integrate security into any iterative development process by finding what (artifacts) already exists and how this can be leveraged for security's goals and by whom. Praxis [6] proposes a methodology to build reliable and secure software by ensuring correctness for every step of a software lifecycle process, often by using formal verification techniques. McGraw [11] suggests a methodology based on three main pillars: risk-management, seven high-level areas of improvements (called 'touch-points') and ancillary resources, i.e. the knowledge base.

Other sources either address particular phases of the development lifecycle, or focus on specific platforms. Howard and Leblanc [7] elaborate on designing secure applications, on writing robust code and on testing applications. Wheeler [17] provides a set of guidelines for designing and implementing secure programs on the Linux and Unix platforms. Concerning methodologies for assurance, the Common Criteria provides assurance that the process of specification, implementation and evaluation of a product has been conducted in a rigorous and standard manner. Similarly, the Systems Security Engineering Capability Maturity Model (SSE-CMM) [2] covers all phases of the development process and can be used to evaluate and improve an existing process.

7. Conclusions

This paper 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 develop-

ment 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 paper.

References

- [1] Secure software. <http://www.securesoftware.com/>, 2006.
- [2] Systems security engineering capability maturity model (SSE-CMM), 2006. Standard ISO/IEC 21827.
- [3] Build security in, 2007. <https://buildsecurityin.us-cert.gov/>.
- [4] K. Beck. *Extreme Programming Explained*. Addison-Wesley, Oct. 1999.
- [5] T. Davenport. *Process Innovation: Reengineering Work Through Information Technology*. Harvard Business School Press, Boston, 1993.
- [6] A. Hall and R. Chapman. Correctness by construction: Developing a commercial secure system. *IEEE Software*, 19(1):18–25, 2002.
- [7] M. Howard and D. E. Leblanc. *Writing Secure Code*. Microsoft Press, Redmond, WA, USA, 2002.
- [8] M. Howard and S. Lipner. *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software*. Microsoft Press, 2006.
- [9] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, Feb. 1999.
- [10] A. Jaquith. *Security Metrics: Replacing Fear, Uncertainty, and Doubt*. Addison-Wesley Professional, 2007.
- [11] G. McGraw. *Software Security: Building Security In*. Addison Wesley, 2006.
- [12] OWASP. Comprehensive, lightweight application security process. <http://www.owasp.org>, 2006.
- [13] G. Peterson. Collaboration in a secure development process. <http://www.arctecgroup.net/articles.htm>, Jun 2004.
- [14] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, Sept. 1975.
- [15] G. Stoneburner, C. Hayden, and A. Feringa. Engineering principles for information technology security. NIST Special Publication 800-27, Revision A, June 2004.
- [16] J. Viega and G. McGraw. *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison-Wesley, 2002.
- [17] D. Wheeler. Secure programming for linux and unix howto. citeseer.ist.psu.edu/wheeler00secure.html.