Cybersecurity Through Secure Software Development

Audun Jøsang[1(✉)], Marte Ødegaard[2], and Erlend Oftedal[3]

[1] University of Oslo, Oslo, Norway
josang@ifi.uio.no
[2] BEKK Consulting, Oslo, Norway
marte.odegaard@bekk.no
[3] NSense, Oslo, Norway
erlend@oftedal.no

Abstract. Reports about serious vulnerabilities in critical IT components have triggered increased focus on cybersecurity worldwide. Among the many initiatives to strengthen cybersecurity it is common to see the establishment and strengthening of CERTs and other centers for cybersecurity. On the other hand, strengthening education in IT security and applying methods for secure systems development are methods that receive much less attention. In this paper we explain how the lack of focus on security in IT education programs worldwide is a significant contributor to security vulnerabilities, and we propose an agile method for secure software design that requires team members to have received adequate security education and training.

Keywords: Cybersecurity · Security education · Waterfall model · Agile model · Secure agile · Secure software development

1 Introduction

Digitization of business processes and services entails huge savings and increased efficiency. To be sustainable, this development must not at the same time introduce serious security vulnerabilities, but unfortunately it often does. The exposure surface to criminals and other malicious players increases by several orders of magnitude when business processes migrate completely or partially to online platforms. This global exposure to security threats makes it natural to use the term cybersecurity in the sense of protecting assets (information, systems and business processes) that directly or indirectly are connected to the Internet. Robust cybersecurity at all levels is necessary for maintaining a balanced risk profile in a digital economy.

Achieving a secure ICT infrastructure obviously requires that it is designed, built and operated by people who understand the threats, know the security requirements and have the skills to build and operate secure systems in general.

Security vulnerabilities in software are typically caused by programmers or teams with inadequate skills in secure software development. Unfortunately,

M. Bishop et al. (Eds.): WISE9, IFIP AICT 453, pp. 53–63, 2015.
DOI: 10.1007/978-3-319-18500-2_5

thousands of IT designers and experts around the world are lacking security skills precisely because cybersecurity was not part of the study program they followed at the university. The expanding ICT infrastructure worldwide is being built by IT experts with IT degrees from universities and colleges, but unfortunately many IT experts still have insufficient security understanding and expertise. This is an unacceptable situation.

As an analogy, it would of course be irresponsible and even unthinkable to educate building architects and civilly engineers without giving them adequate knowledge about fire safety, otherwise the buildings in which we work and live would be full of firetraps. Likewise it is irresponsible to offer IT programs at universities without compulsory modules in information security. Unfortunately, still today many IT graduates leave university and go into industry without any competence in information security. Despite their great skills in programming and IT design, without skills in security these IT graduates will necessarily build vulnerable IT solutions.

This paper discusses how cybersecurity can be strengthened, not by investing in more sophisticated attack detection and malware filtering tools, but by ensuring that the very foundation of the ICT infrastructure is designed and built for strong security and robustness. This can only be achieved by ensuring that secure system design becomes a natural element in all development projects, and by encouraging, stimulating and maybe forcing technical colleges and universities to integrate mandatory security modules in the curriculum of their IT education programs.

2 Patterns of Cybersecurity

There are currently available on the market a large number of tools and services for strengthening cybersecurity. Vendors range from large and highly profiled companies to small and relatively unknown companies. Some security products are promoted to stop APTs (Advanced Persistent Threats). Other products are promoted to detect and stop singular zero-day attacks or to filter out malware. Intrusion detection and prevention can be bought as a system or as a service. Security governance frameworks such as COBIT[1], ITIL[2], ISO 27001 [6] and NIST SP-800-53 [8] can be adopted and applied by in-house security management staff or by external consultants to ensure that the organisation is managed according to best practice with regard to security. However, none of these tools, services and frameworks – neither in isolation, nor in combination – are sufficient to avoid serious cyber-vulnerabilities of the kind that e.g. Heartbleed, Shellshock and BadUSB represent.

There seems to be no approach that can provide the level of security assurance that governments and corporate managers aim for. In a recent article it is argued that the cybersecurity product market is different from other markets where a few players typically dominate the market [13]. However, for cybersecurity

[1] COBIT: Control Objectives for Information and Related Technology.
[2] ITIL: Information Technology Infrastructure Library.

there is no single vendor – not even a set of major vendors – who dominate the cybersecurity market, and from which general cybersecurity solutions can be bought. This simple analysis indicates that cybersecurity does not lend itself to a packaged solution [13].

It is up to the CISO (Chief Information Security Officer) or a similar executive in organisations to set up and run the cybersecurity program as they see appropriate, but since there is no commonly approved product, service or program that can provide waterproof cybersecurity, this task is particularly challenging. Even when following industry best practice there will be vulnerabilities that attackers can exploit to mount successful attacks. In this situation, when a serious security incident occurs, although there often is little the CISO could have done to prevent the incident, the CISO is the obvious target of blame anyway [12]. In a study by the Ponemon Institute focusing on the role of the security manager within companies, many of the CISOs who took part in the study rated their position as the most difficult in the organization. Most of CISOs questioned said their job was a bad one, or the worst job they had ever had [5].

Vulnerability management is an important branch of security management in organisations. Commonly known vulnerabilities are assigned unique identities under the CVE (Common Vulnerabilities and Exposures) scheme started by MITRE Corporation in 1999, with funding from the National Cyber Security Division of the US Department of Homeland Security. CVE IDs are used in SCAP (Security Content Automation Protocol) which is a protocol used by vulnerability management tools. A complete list of CVE IDs can be found on MITRE's CVE system as well as in the US National Vulnerability Database. Prominent software vendors can become a CNA (CVE Numbering Authority) for their products, and each CVE IDs is actually assigned by a CNA. There are three primary types of CNAs:

- The MITRE Corporation functions as Editor and Primary CNA.
- Various CNAs assign CVE IDs for own products (e.g. Microsoft, Oracle, Red Hat).
- Red Hat also provides CVE numbers for open source projects that are not a CNA.

The original CVE-ID format had just four digits for numbering vulnerabilities per year, such as CVE-2014-0160 which identifies the Heartbleed vulnerability. Only allowing 9,999 vulnerabilities per year was seen as a limitation, so that from 2014 the CVE-ID format can have five, six or more end digits to identify an arbitrarily large number of vulnerabilities each year. Figure 1 shows the number of identified vulnerabilities in the CVE scheme during the last 15 years.

The trend seen on Fig. 1 is that the number of vulnerabilities is increasing. There is also a wide range of different vulnerability types, and efforts have been made to provide classifications, taxonomies and ontologies for security vulnerabilities [11]. One purpose of vulnerability classification is to identify weaknesses in the SDLC (Software Development Lifecycle) in order to avoid vulnerabilities in the first place.

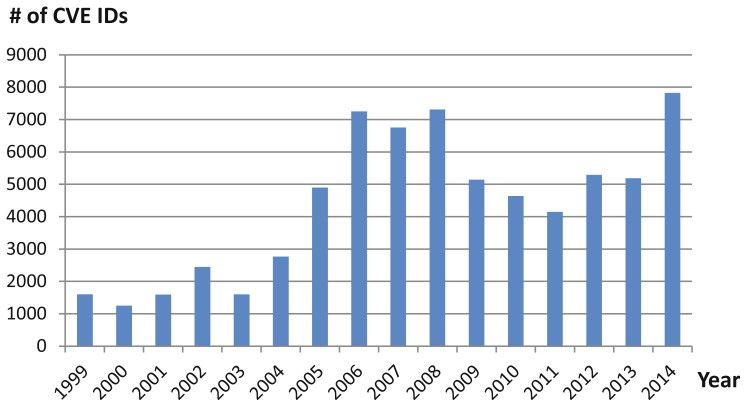

Fig. 1. Number of CVE IDs registered in the period 1999–2014

It is of course impossible to completely avoid generating security vulnerabilities during system and software design. However, the state of cybersecurity can be significantly improved by reducing both the number and the severity of security vulnerabilities generated. The question then is how best to work towards this goal. Several approaches are already used to this end, such as automatic methods that can do code analysis and fuzzing to discover and eliminate vulnerabilities before setting code into production. The most important approach is to follow principles for secure software development, and to ensure that software designers have sufficient security expertise. We discuss the latter approaches below.

3 Software Development Lifecycle

Several software development models or approaches have been proposed and applied during the last 30 years. Each model has its characteristics, advantages and disadvantages, but common to them all is that they are typically not focusing on security [4] (p. 1111). A selection of five prominent development models are briefly analysed and compared in [9]. These are:

- Waterfall model
- Iteration model
- V-shaped model
- Spiral model
- Agile model, aka. XP (Extreme Programming)

The waterfall model is the classical and most heavy-weight approach to software development, whereas the agile model is the most light-weight and flexible approach. We will briefly describe the waterfall and agile models, as they represent very different approaches to secure programming. Figure 2 shows the waterfall model.

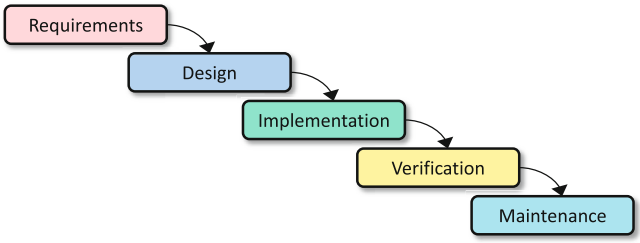

Fig. 2. The waterfall model for software development

The basic idea behind the waterfall model is that the tasks of each phase must be fully completed before the next phase, which is symbolized by the waterfall metaphor where water only flows downwards. This also implies that the complete set of requirements must be defined and fixed at the beginning of the project. In case it is necessary to revisit a previous stage, then a costly overhead is to be expected (metaphorically make water flow upwards), so this should be avoided. However, it is typically the case that requirements have to be changed in the middle of a software development project, so that many software development projects based on the waterfall model have suffered large blow-outs in cost and time.

As a reaction to the rigid structure of the waterfall model several other models have been proposed, where the most recent and radical is the agile model (also known as XP: eXtreme Programming) illustrated in Fig. 3 below.

The basic idea behind the agile model is that new or evolving requirements can be specified in parallel with, or after already implemented requirements [1]. This is possible by splitting the development into separate *stories* where each story covers a set of requirements implemented as functions that can be developed and tested more or less independently of other stories. Each cyclic iteration

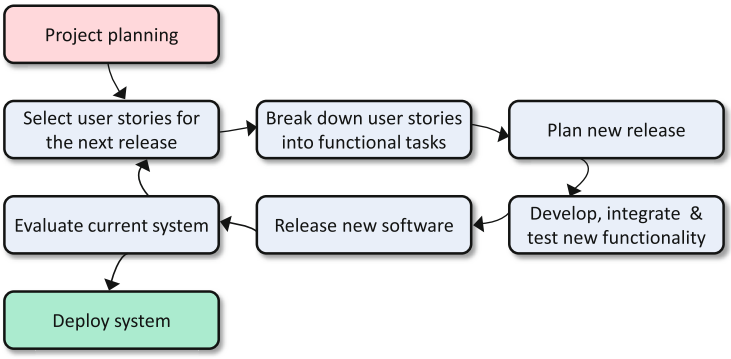

Fig. 3. The agile model for software development

in the agile model is a *sprint* which can be completed in only a few weeks. The major drawback of the agile model is that it often does not scale well to large and complex development projects.

Specific security related tasks should be included in the various phases of the SDLC, whether the development follows the waterfall model, the agile model, or any other model. Due to the radical difference between the waterfall and agile models, the development team needs to adapt the specific approach to secure development depending on the model followed, as described in the next section.

4 Secure Software Development

4.1 Secure Software Development in the Waterfall Model

There are several recommended and 'best practice' models to ensure secure development, including the NIST framework for Security Considerations in the System Development Life Cycle [7], as well as Microsoft's Security Development Lifecycle (SDL) [2] illustrated in Fig. 4 below. Both the NIST model and the Microsoft SDL model are based on the waterfall model for SDLC.

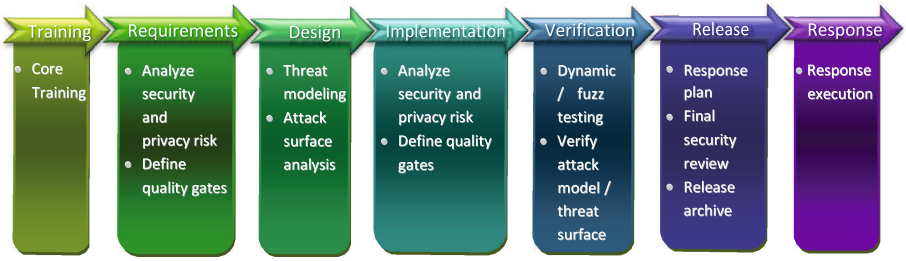

Fig. 4. SDL: Microsoft Security Development Lifecycle [2]

The 1st phase of Microsoft SDL is security training which emphasizes how important it is that programmers and other team members have acquired adequate security skills for their job. Security training empowers developers to have awareness for threats and vulnerabilities and to create secure applications.

In Microsoft SDL, every phase of the waterfall model includes security related tasks, such as in the planning phase, requirements phase and design phase. Risk analysis is for example included in the design phase.

According to [4] (p. 1102) most vulnerabilities emerge during coding. It is therefore crucial that the programmer follows strict and secure methods of secure programming. By looking at the list of 25 most common software errors maintained by SANS[3], we see that many of these are directly related to irresponsible or sloppy programming practice.

[3] http://www.sans.org/top25-software-errors/.

4.2 Security Software Development in the Agile Model

There are relatively few studies in the literature on secure agile software development models. In Wichers' proposal [14] it is argued that secure software development in the agile model needs a quite different approach to that of the waterfall model.

In [14] it is recommended to identify all stakeholders and clarify what their main security concerns are. From this analysis a set of threat models can be extracted which in turn form the basis for stakeholder security stories. Then during the development phase, one has periodic security sprints in between the regular development sprints. It is also proposed to include a final security review before deploying the final system.

Microsoft has presented a version of SDL for agile software development [3]. The Agile SDL model contains the same security steps as in the waterfall SDL model, where these steps are grouped in 3 categories:

- One-Time practices: Foundational security practices that must be established once at the start of every new Agile project.
- Every-Sprint practices: Essential security practices to be performed in every sprint.
- Bucket practices: Important security practices that must be completed on a regular basis but can be spread across multiple sprints during the project lifetime.

We find that Microsoft's agiler SDL has merit. However, it has limitations by not separating between functional and non-functional security requirements.

We therefore propose an agile model for secure software development which is partially inspired by the model described in [14] and by Microsoft's Agile SDL model, but which is also an improvement over these because it does not share their disadvantages mentioned above. Our model is also inspired by previous work in [10].

Our approach to handling security in the agile model is based on the distinction between what we call functional security controls and non-functional security controls.

- Functional security controls reflect and implement user stories that are directly related to security, such as when password management and verification is used as a control to implement a user story for logon, or when ACLs (Access Control Lists) are used as a control for specifying and enforcing policies for using various resources within a domain.
- Non-functional security controls are applied in order to eliminate or mitigate vulnerabilities in the implementation of other user stories, such as when applying secure programming techniques in order to avoid buffer overflow bugs, or when applying input filtering when designing a front-end to an SQL database in order to avoid SQL-injection. Software designer must understand that any type of user stories, both ordinary user stories as well as specific security related user stories, must be implemented in a secure way. The way to do that is precisely through non-functional security controls. The idea is that security threats that are intrinsic to a specific user story should be handled during the sprint for the same user story.

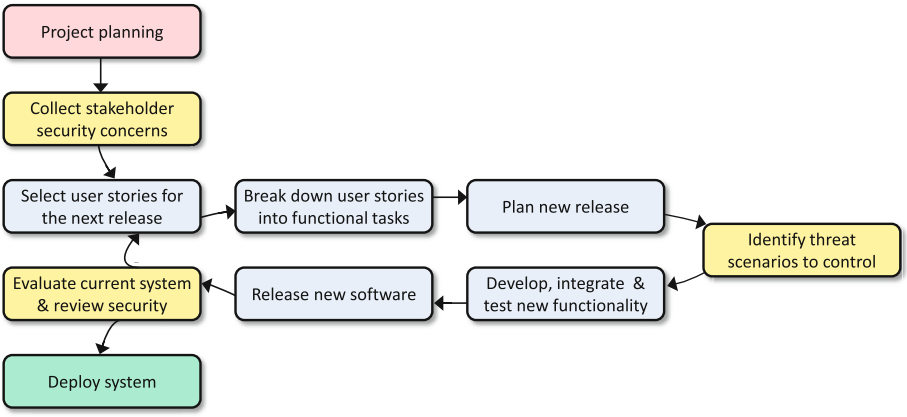

Fig. 5. Proposed model for secure agile software development (Color figure online)

A further example of non-functional security controls is when implementing a user story about the logic for handling the check-out of a shopping basket on an e-commerce website, where a threat could be that the customer is able to trick the system into changing the number of items after the price has been computed, so that he could receive many items but only pay for one. This security concern must be handled during the sprint that implements the check-out of shopping baskets. Based on these considerations we propose to introduce a new security phase into the sprint iteration. This security phase focuses specifically on identifying threats against the current user stories. The new phase should also specify how the threats can be controlled or mitigated, and should specify tests for those mitigation controls. The implementation of non-functional security controls is then handled in the ordinary phase that develops, integrates and tests the new functionality for the current sprint.

Finally, we propose to include a security review in the phase of the sprint iteration cycle where the current version of the system is evaluated. This modified phase, as well as the two new phases that are specific to security, are indicated as yellow boxes in Fig. 5 which illustrates our proposed model for secure agile software development.

It may sometimes be unclear whether a security requirement is functional or non-functional, in which case there is a danger that security requirements are not handled according to the model. The question is what happens when trying to handle functional security as part of other user stories, or when trying to handle non-functional security as a separate user story, which is the opposite of what the model recommends. We analyse these two irregular cases separately below.

– Handling an obvious non-functional security requirement as a functional security requirement would generate unnecessary overhead. This is because non-functional security is an integral part of other user stories, so if non-functional

security is handled separately it would lead to re-iteration of those user stories. Nevertheless, the non-functional security requirements could be adequately taken care of. The conclusion is that development efficiency would suffer, not security.
- In case as an obvious functional security requirements is not handled as a separate user story then the design team has no clear strategy for handling such requirements. At the limit, such functional security requirements could be awkwardly included as a sub-story of other user stories. Alternatively the requirements would not be handled at all, which would be a serious design failure.

If in doubt, it is always safe to consider a security requirement as a separate user story. However, to optimize agility, clearly non-functional security requirements should be integrated as part of other relevant user stories whenever possible.

5 Cybersecurity Training

Security design is challenging, and requires strong skills to get it right. It can therefore not be expected that IT graduates without security training have the necessary skills for developing secure systems. A typical approach to security design in the industry is to let security specialists do a security review of systems that have been developed by software designers without security skills. However, this approach wastes time and manpower. The right approach is to have system designers with security skills who are able to identify vulnerabilities and threat scenarios during the design and development.

It is interesting to notice that in the Microsoft SDL model for secure software design, the 1st phase focuses on security training. In other words, SDL assumed that the design team has acquired security skills before the proper development project starts.

In agile models for secure software design there is no separate phase for security training, neither in the model presented [14] nor in our model presented in Sect. 4.2. For both models it is assumed that team members already have the required knowledge and skills to identify threat scenarios and to craft the corresponding security controls for mitigating those threats. In other words, without security skills among the development team members, no agile model for secure software development would be practical. Given that a large proportion of students following IT programs today still get no or only limited exposure to cybersecurity it is obvious that practicing agile models for secure development is problematic. For this reason there are maturity models for secure software development, where the most prominent are *Building Security In Maturity Model 2* (BSIMM2) and OWASP's *Open Software Assurance Maturity Model* (OpenSAMM).

Governments or interest organisations in many countries are aware of this deficiency and have therefore launched programs to strengthen security education.

In the USA for example, NICE (National Initiative for Cybersecurity Education) established in 2014 builds on the previous Comprehensive National Cybersecurity Initiative started in 2008 as an initiative to strengthen cybersecurity

skills in the US federal government sector. NICE has been extended to include the commercial sector, where the goal is to strengthen cybersecurity skills from kindergarten through post-graduate school. The goal of NICE is to establish an operational, sustainable and continually improving cybersecurity education program for the nation to use sound cyber practices that will enhance USA's security.

However, NICE is a US-based imitative and it is still too early to tell how it will influence security education in university IT-programs. The global higher education sector has not yet reached a common consensus that cybersecurity should be an integral part of IT education. In contrast to IT education, in the domain of architecture and civilly engineering education it is obvious that students must acquire knowledge about fire safety. Admittedly there is a difference between these two domains. In architecture and civilly engineering there are strict regulations regarding fire safety, whereas there are no specific regulation for information security in IT systems.

We think initiatives like NICE in the USA should be copied in other countries as well, and could be supported by national computer societies and their umbrella organisation IFIP. National governments could also encourage the higher education sector to strengthen cybersecurity education as part of their IT education programs.

6 Conclusion

There is a consensus in the industry that security must be part of the software development lifecycle. It is not a question of which development model is used, but how well the organisation is able to integrate security in the process.

A weak spot in all the models is that they all depend on the team members having adequate security skills. It can not be expected that every organisation must provide security training to their own staff. Security training must therefore be part of IT education programs in higher education. There is an urgent need to strengthen IT education programs worldwide with regard to cybersecurity. For this purpose it would be interesting to define a security education maturity model for the university sector. If a university offers an IT education program with insufficient security, then that university is part of the problem of causing cybersecurity vulnerabilities. It is time for all IT education institutes to become part of the solution.

References

1. Beck, K. et al.: Manifesto for Agile Software Development, February 2001. www.agilemanifesto.org,
2. Microsoft Corporation. SDL: Microsoft Security Development Lifecycle. Version 4.1 (2009)
3. Microsoft Corporation. Security Development Lifecycle for Agile Development. Version 1.0, 30 June 2009. http://www.microsoft.com/security/sdl/discover/sdlagile.aspx

4. Harris, S.: CISSP All-in-One Exam Guide, 6th edn. McGraw-Hill, New York (2013)
5. Ponemon institute: understaffed and at risk: today's IT security department. Technical report, Ponemon Institute, January 2014
6. ISO. ISO/IEC 27001:2013 - Information technology - Security Techniques - Information security management systems - Requirements. ISO/IEC, 2013
7. Kissel, R. et al.: Security considerations in the system development life cycle - NIST special publication 800-64, Rev. 2. Technical report, National Institute of Standards and Technology, October 2008
8. Joint task force transformation initiative; computer security division; information technology laboratory. security and privacy controls for federal information systems and organizations - NIST special publication 800-53 Rev. 4. Technical report, National Institute of Standards and Technology, April 2013
9. Munassar, N.M.A., Govardhan, A.A.: A comparison between five models of software engineering. Int. J. Comput. Sci. Issues (IJCSI) **7**(5), 94–101 (2010)
10. Oftedal, E.: Leveraging agile to gain better security: an agile developer's perspective. In: OWASP AppSec Europe, Poland (2009)
11. Pascal, M.: Handbook of Science and Technology for Homeland Security. Classes of Vulnerabilities and Attacks. Wiley, New York (2007)
12. Perlroth, N.: A Tough Corporate Job Asks One Question: Can You Hack It? New York Times Online 20, July 2014
13. Ross, S.J.: Whiz Bang 2000. ISACA J. **6**, 4–5 (2014)
14. Wichers, D.: Breaking the Waterfall Mindset of the Security Industry. In: OWASP AppSec USA, New York (2008)