FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Information Security

NI-IBE

Software Development Security

Autor:
Tomáš PATRO

November 28, 2020

Abstract

Software development security is an essential and fundamental part of developing a piece of software. The real-life industry projects' experiences show us that security is often a part of each phase of the Software Development Life Cycle (SDLC).

In this article, we look at the SDLC definition and describe each phase of the process. We put each phase into the context of development security. Based on the knowledge about the SDLC, we look at the three different processes used by the major players in the industry. The aim of these processes is to formalize activities and approaches to develop the software securely. As an addition, in the last section, we talk about the best practices for software development security.

Keywords – software, security, development, sdlc, process, processes, application, management, risk, best, practices

Contents

Introduction **4**

1 Software Development Life Cycle (SDLC) **5**
- 1.1 Planning . . . 5
- 1.2 Defining . . . 6
- 1.3 Designing . . . 6
- 1.4 Building . . . 6
- 1.5 Testing . . . 6
- 1.6 Deployment . . . 7
- 1.7 Operation and Maintenance . . . 7
- 1.8 SDLC Summary . . . 7

2 Secure Software Development Processes **7**
- 2.1 Security Development Lifecycle (SDL) [8] . . . 8
  - 2.1.1 Security as Supporting Quality . . . 8
  - 2.1.2 Well-defined Process . . . 9
  - 2.1.3 Good Guidance . . . 9
  - 2.1.4 Management Perspective . . . 9
- 2.2 Comprehensive, Lightweight Application Security Process (CLASP) [8] . . . 9
  - 2.2.1 Security at Center Stage . . . 9
  - 2.2.2 Limited Structure . . . 10
  - 2.2.3 Role-based . . . 10
  - 2.2.4 Rich in Resources . . . 10
- 2.3 Touchpoints [8] . . . 11
  - 2.3.1 Risk management . . . 12
  - 2.3.2 Black vs. white . . . 12
  - 2.3.3 Flexibility . . . 12
  - 2.3.4 Examples . . . 12
  - 2.3.5 Resources . . . 12
- 2.4 Processes Conclusions . . . 12

3 Software Development Security Best Practices **13**
- 3.1 Best Practices – Set 1 . . . 13
- 3.2 Best Practices – Set 2 & 3 . . . 14

Conclusion **17**

References **18**

Introduction

The software engineers pointed their fingers at the inability to handle the computer systems' complexity as early as 1968. It happened during the first software engineering conference organized by NATO. [1]

The security during the development very much depends on the complexity of the overall system. The more complex the system is, the more we emphasize the importance of development security. However, the experience from the industry shows us that security during the development is often based on intuition and somewhat incomplete heuristics.

In this article, we will look at the Software Development Life Cycle (SDLC) in the context of the development security. We will also acquaint the reader with two different approaches to enforce a secure development – security process frameworks and usage of the best practices. Based on the gained information, these approaches show their potential and may apply to a lot of different software projects. However, it is also important to note that these are not the only viable options for development security. There are many other mechanisms on how to ensure development security; we will talk about only two of them.

1 Software Development Life Cycle (SDLC)

In this section, we will talk about the software development life cycle. It is a fundamental term in software engineering, and it is good first to gain a better knowledge of this process, so we can put it into the context of the development security.

Every controlled process has some form of the life cycle, which determines the particular steps we take from the beginning to the end of the process. If we have a defined process, we can also define the resources and additional information connected to each step of the process. It is not different from the development of a piece of software.

The software development life cycle also defines several steps that determine the process of development of the software. These steps are the application of standard business practices to building software applications. The process is typically divided into six to eight steps. Some project managers may combine, split, or omit some steps, depending on the project's scope. [2]

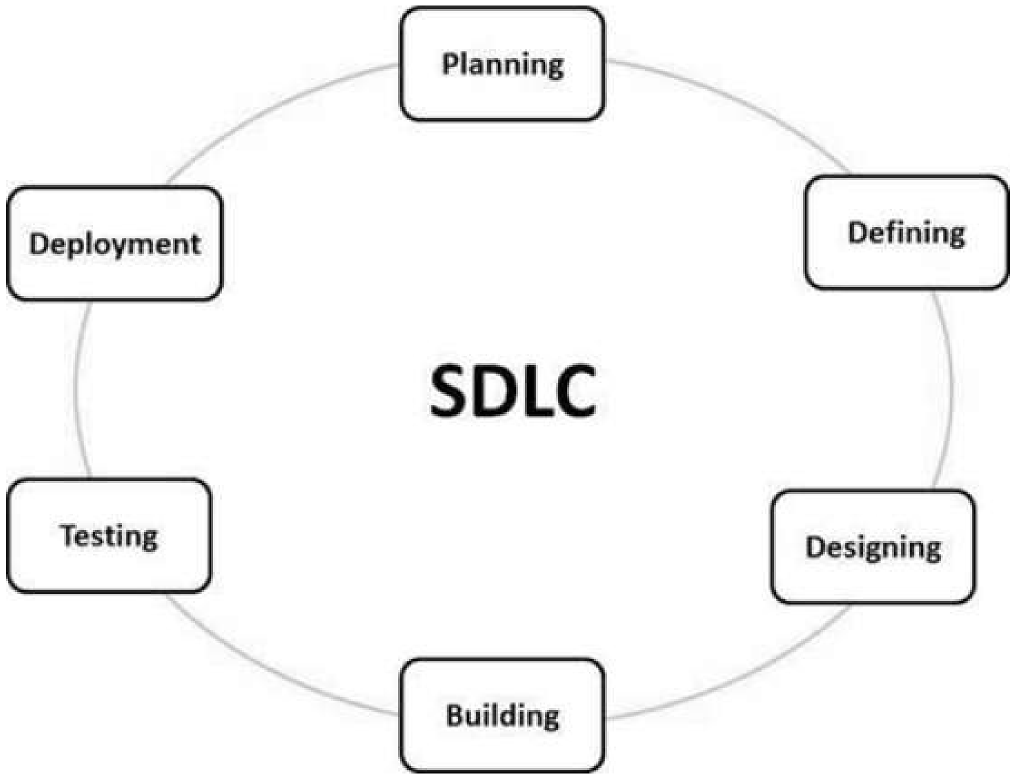

Figure 1: A typical SDLC consists of these stages [3]

1.1 Planning

Planning is often tightly connected with the requirements analysis. The senior team members often perform it. The inputs mostly come from the customer, sales department, market surveys, and domain experts in the industry. This information is later used to plan the project approach and conduct the feasibility study. [3]

Another part of the planning may also be quality assurance requirements. These requirements often involve the identification of the risks associated with the project. The risks are often included in the feasibility study's technical part, which may also contain an overview of the security risks. [3]

1.2 Defining

Once the requirements analysis is done, the next step is to clearly and formally document the product requirements, which are later discussed and approved by the customer or the market analysts. There is often a need to redefine these requirements based on customer feedback or feedback from other sources. The redefining can also be connected to the searching for potential security threads and risks. [3]

1.3 Designing

The design involves the architects to come out with the best architecture for the product to be developed. The product of this step is often some form of a design specification. This specification is then reviewed by all important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, etc. [3]

Some aspects of the design include architecture, user interface, platforms, programming, and communications. The security aspect is also often a part of this step. It defines the measures to be taken to secure the application and can involve things like SSL encryption, password protection, and secure storage of user credentials. [2]

1.4 Building

In this stage, the actual code is written, and the product is built. Developers must follow the coding guidelines defined by their organization, and programming tools like compilers, interpreters, debuggers, etc., are used to generate the code. The guidelines may include a set of best practices that often help write a better and more secure code. [3]

The coding process also includes many other tasks. The developers often face tasks like finding and fixing errors and critical glitches. These aspects can often hold up the development process. However, it is also important for the code's overall security to address these issues in the development process. [2]

Documentation can also be a guideline of the application's basic features. It can have a form of written documentation like user guides, troubleshooting guides, edge cases analysis, etc. [2]

1.5 Testing

This step is often a subset of all the stages in the SDLC. However, this step refers to the testing only stage of the product where we detect products' security issues, bugs, glitches, and other types of defects. The defects need to be addressed and reprogrammed accordingly. This step should end only if the product reaches the company's quality standards. [3]

Much of the testing can also be automated. This also includes security testing like static or dynamic code analysis, etc. [2]

1.6 Deployment

The deployment involves a formal release of the product in the appropriate market. Sometimes the product deployment happens in stages as per the business strategy of the company. The product may be first released in a limited segment and tested in the real business environment (user acceptance testing). Based on the feedback, the product may be adjusted accordingly. There is also a space for finding potential bugs or security issues not detected by the testing phase. [3]

Many companies prefer to automate the deployment phase. The automation may be complex and needs to be finely orchestrated not to bring security defects to the deployment process. [2]

1.7 Operation and Maintenance

At this point, the product is developed and used in the field. However, the operation and maintenance phase is still important. In this step, the users discover and report bugs and security issues that weren't found during coding and testing. These errors need to be resolved, which may spawn new development cycles. [2]

Often, a part of this phase is also some form of service-level agreement (SLA) between the customer and provider. This agreement documents what services the provider will furnish and defines the service standards the provider is obligated to meet. This may also include security measures that will be taken by the service provider. Typically, this includes the drafting and consensus on antipoaching, IT security, and nondisclosure agreements. [4]

1.8 SDLC Summary

Based on the steps described above, we may conclude that development security is included in all the SDLC steps (phases). We can ask ourselves the question of how to approach security in the development process. There may be many answers to this question. One possible solution may be to define a set of standards/processes that should be followed throughout the SDLC.

2 Secure Software Development Processes

In this section, we will look at the three processes which are used to ensure the security during the SDLC. The focus is put on three forefront representatives, namely Microsoft's *Security Development Lifecycle* (SDL) [5], OWASP's *Comprehensive, Lightweight Application Security Process* (CLASP) [6], and McGraw' *Touchpoints* [7].

In Section 1 we stated that the security issues are present in all phases of the SDLC. However, the construction of secure software is still largely a matter of guidelines, best practices, and undocumented expert knowledge in many companies. [8]

Several advances have been made in the definition of the process for secure software development by the major players in the field like Microsoft or OWASP. The information about the processes are taken from the *Bart De Win's* paper *On the secure software development process: CLASP, SDL and Touchpoints compared* [8].

2.1 Security Development Lifecycle (SDL) [8]

In 2002, Microsoft made a commitment to follow the so-called "trustworthy computing" principle. As a result, the company defined the *Security Development Lifecycle (SDL)*. SDL is a process definition that is composed of a set of activities, which complement Microsoft's development process. These activities are particularly aimed to address security issues during the development process. The following aspects characterize the process:

1. Security as a supporting quality
2. Well-defined process
3. Good guidance
4. Management perspective

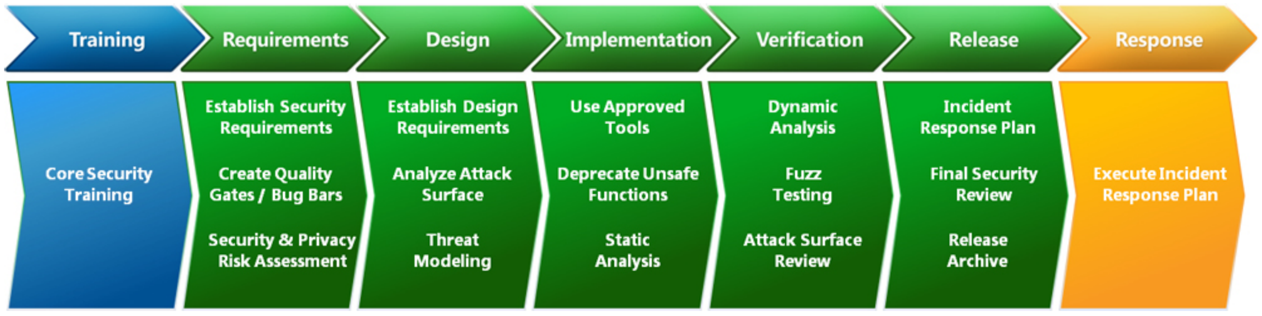

Figure 2: The seven phases of the Security Development Lifecycle Process. [9]

2.1.1 Security as Supporting Quality

The primary goal of SDL is to function as a supportive add-on to the software construction process. It reaches this functionality by increasing the quality of functionality-driven software. This is done by improving its security approach.

Security activities are most often related to functionality-based construction activities. For example, threat modeling starts from architectural dependencies with the external systems. However, the architecture itself could, in fact, reduce such threats in the first place. The SDL defines many similar approaches to enrich the quality of the development security.

2.1.2 Well-defined Process

From Figure 2, we can see that SDL is well-organized into the phases similar to the SDLC described in Section 1. Although the stages are security-specific, it is straightforward to map them to standard software development phases – in the section 1 we concluded that security is often included in all steps of the SDLC.

Furthermore, several activities have continuous characteristics. This includes threat modeling and education. Thus, the SDL process incorporates support for revising and improving intermediate results.

2.1.3 Good Guidance

SDL achieves good guidance by specifying the method that must be used to execute activities. On average, these specifications are concrete and often somewhat pragmatic. For example, the attack surface reduction is guided by a flow chart. Another example can be threat modeling, which is described as a set of sub-processes. As a result, the execution of an activity is quite achievable, even for less experienced people.

2.1.4 Management Perspective

Often in the companies, we can observe that security as quality has to be managed in order to be realized in practice. SDL takes a management perspective for the description of many activities. This is a very convenient feature given the inherent complexity of the security.

2.2 Comprehensive, Lightweight Application Security Process (CLASP) [8]

CLASP is a *lightweight* process for building secure software. It consists of 24 top-level activities, which are general enough to be tailored to the development process in progress. Some of the key features include:

1. Security at the center stage
2. Limited structure
3. Role-based
4. Rich in resources

2.2.1 Security at Center Stage

As shown in Figure 3, the CLASP framework is mainly proposed for the construction of software in which security takes a central role. The set of activities is fairly broad since they are defined and conceived primarily from a security-theoretical perspective.

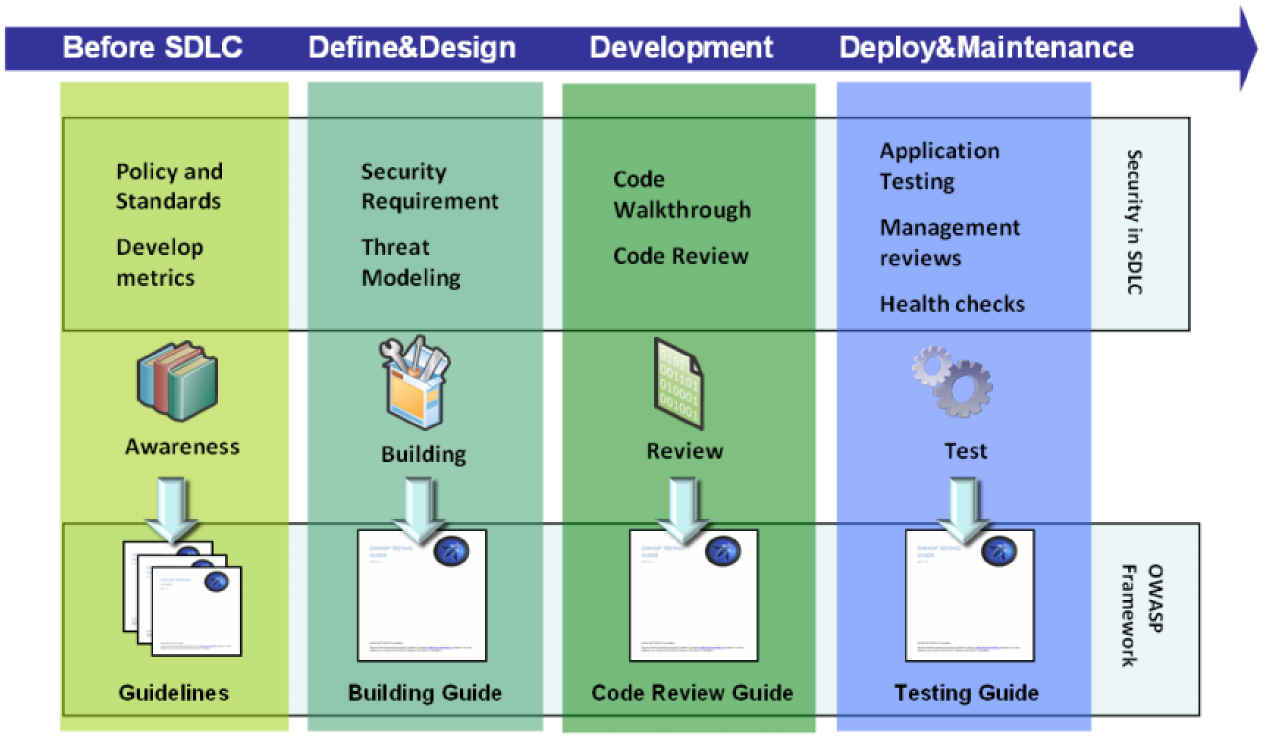

Figure 3: SDLC & OWASP Guidelines. [10]

2.2.2 Limited Structure

CLASP framework offers a certain level of flexibility. It is achieved by the independence of the activities that have to be integrated into the development process and its operating environment. It is up to a particular company to choose the activities to be executed and choose the order of execution. Moreover, the execution frequency is specified per individual activity. The coordination and synchronization of the activities may thus not be straightforward. Besides, the CLASP framework defines two road maps ("legacy" and "greenfield"), which give some guidance on how to combine the activities into a coherent and ordered set.

2.2.3 Role-based

We already mentioned activities, which are the building blocks of the CLASP. The framework also consists of the roles responsible for the finalization and the quality of the results of an activity. These roles can also have a direct impact on the security approach. Roles are used as an additional perspective to structure the set of activities.

2.2.4 Rich in Resources

CLASP provides and an extensive set of security resources used as support in the implementation of the activities. For example, one of the resources is a list of 104 known security vulnerabilities in application source code.

2.3 Touchpoints [8]

Let's introduce Gary McGraw, the author of the *Seven Touchpoints for Software Security*, first:

"Gary McGraw is the CTO of Cigital, Inc., a software security and quality consulting firm providing services to some of the world's best-known companies for a decade. Dr. McGraw is a globally-recognized authority on software security-featured frequently as a keynote speaker at events coast-to-coast as well as internationally." [11]

Touchpoints describe a set of security best practices that come from the extensive industrial experience of McGraw. The best practices (activities) are grouped together in seven so-called touchpoints. Some of the aspects of Touchpoints include:

1. Risk management
2. Black vs. white
3. Flexibility
4. Examples
5. Resources

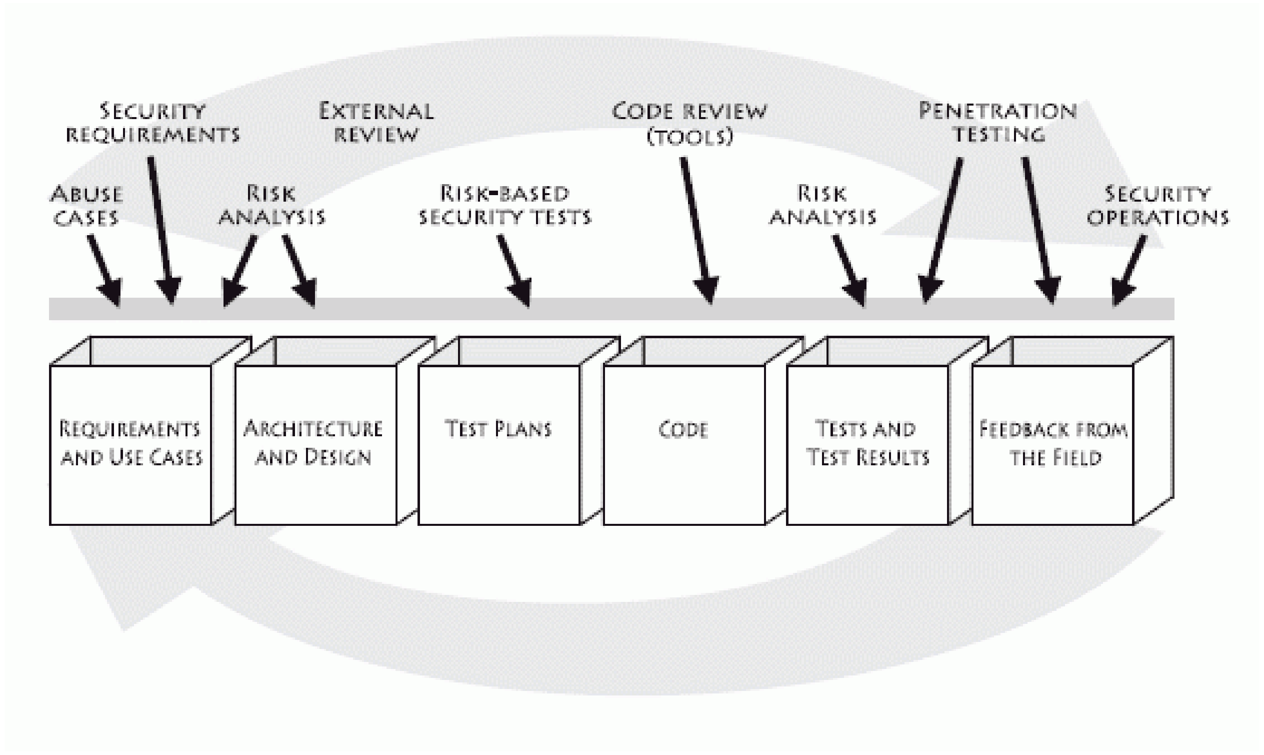

Figure 4: Seven touchpoints (best practices) of software security. [11]

2.3.1 Risk management

Risk management is an important part of software security. Touchpoints acknowledge its importance and try to bridge the gap by elaborating the *Risk Management Framework (RMF)* that supports the Touchpoints activities.

2.3.2 Black vs. white

Touchpoints come with a mix of black-hat and white-hat activities. Both types are equal in the means of necessity. Black-hat activities aim for attacks, exploits, and breaking software – penetration testing. On the other hand, white-hat activities are more constructive in nature. They cover design, control, and functionality – code review.

2.3.3 Flexibility

Since the Touchpoints are formalized in the form of best practices, we can tailor them and use them for software that already undergoes development. Besides, the documentation provides prioritization of different touchpoints. Thus, the companies are able to gradually introduce the touchpoints, starting from the most important ones.

2.3.4 Examples

Touchpoints bring lots of examples. For example, while describing the abuse cases, there is an example giving the reader a good feel about what they might look like in a particular situation.

2.3.5 Resources

Part of the Touchpoints is dedicated to security knowledge where resources are part of it. The resources further aid the execution of activities. Touchpoints provide references to resources and explain how to use them. For example, attack patterns are provided in order to be used in the elicitation of abuse cases.

2.4 Processes Conclusions

There is always a question about which process or methodology to use for your project. The answer is that it very much depends on a particular situation. We described three widely used processes and their main characteristics. If you want to know more and read about the comparison of these three processes, we recommend to read *Bart De Win's* paper *On the secure software development process: CLASP, SDL and Touchpoints compared* [8]. De Win and his colleagues summarized and compared these processes looking at a lot of different characteristics from education and awareness through release and deployment to support.

3 Software Development Security Best Practices

This last section will look at a subset of the practical best practices for secure software development. We will look at the list of the best practices described by *Ansar-Ul-Haque Yasar*, and his colleagues in the paper *Best practices for software security: An overview* [12]. Yasar et al. made a list of the best practices from the different sources. Yasar et al. also state the following:

"Software security spread covers the whole Software Development Life Cycle phases. Each of the phases should be carefully observed for building secure software."

We can relate to this statement as we described it in Section 1. Yasar et al. chose three example sets of best practices and classified them based on the phases of the SDLC.

3.1 Best Practices – Set 1

Davis et al. proposed the first set in the paper *Processes for producing secure software* [13]. The set is composed of these practices, which should be executed in this order:

1. Abuse cases
2. Security requirements
3. Risk analysis
4. External review
5. Risk-based security tests
6. Static analysis (tools)
7. Risk analysis
8. Penetration testing
9. Security breaks

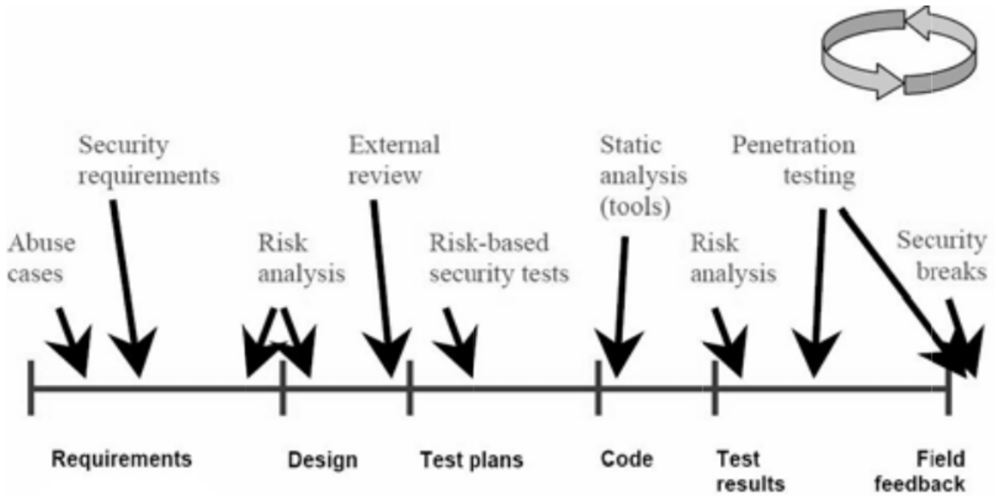

Figure 5: Best practices for the software security represented by software artifacts. [13]

The SDLC is divided into the so-called self-explanatory subgroups, i.e., software artifacts describing the activities performed during the phases. The practices mentioned above are afterward mapped to these artifacts, as we can see in Figure 5.

3.2 Best Practices – Set 2 & 3

The other two sets are rather composed of straightforward recommendations. We will list them and add a simple explanation to each of them. The second set is proposed by *Razvan Peteanu* in the paper *Best Practices for Secure Development* [14]. Peteanu proposed high-level best practices for secure development:

1. **Relevancy and weakest links** – protect the related information and secure the places which may be more prone to attacks.
2. **Clarity** – we should define every aspect of the solution we provide.
3. **Quality** – the quality of security should be evenly maintained with the quality of the overall product.
4. **Involve all stakeholders** – all relevant stakeholders should be involved in the security process (level of involvement should also be defined).
5. **Technology processes** – the processes are related to each other, and there should be a technology able to invoke them.
6. **Fail safe operation** – the product should be able to recover from the errors.
7. **Defense in depth** – we should be able to implement defensive mechanisms in a multi-layered environment.
8. **Principle of appropriate privileges** – everyone should be given privileges to operate the system accordingly considering the development phase and not only the final product.
9. **Interacting with users** – related to the user testing phase. The users often find bugs in the product, and we should address these findings.
10. **Trust boundaries** – we should define the trust boundaries to enforce a reliable security mechanism. For example, we can define a trusted computing base or a trusted path.
11. **Third-party software** – at some point, we may use a third-party software. We should always be cautious when using vendors and potentially also test this software.
12. **Manipulation of sensitive data** – this manipulation should be handled with proper care, and its security should be our top priority.
13. **Attack first on paper** – if possible, it is a good recommendation to try breaking the security architecture during the design phase.

14. **Think "outside of the box"** – an important concept which tells us to always look for alternative approaches in points of views when testing and trying to break the product.

15. **Be humble** – in the terms when we should continue improving the product even after the release.

16. **Declarative vs. programmatic** – this is the most "technical" best practice. It tells us that security should be enforced in the programming either during the execution or outside the program control.

17. **Reviews are your best allies** – also related to the programming. Code reviews should be an essential part of the coding process.

The third and last set is proposed by *David A. Wheeler* in the article *Secure programming for Linux and Unix HOWTO*. This set is the most technical one and aimed mainly for the coding part:

1. **Validate all input** – every external and internal input to the system should be carefully handled.

2. **Avoid buffer overflow** – we should be careful when working with data structures and not cause stack or heap overflow problems.

3. **Secure the interface** – we should design the interface simple enough to understand it easily.

4. **Separate data and control** – illegal or invalid execution may lead to the loss of important data.

5. **Minimize privileges** – we should carefully consider every permissions/privileges assignment and properly design the policies.

6. **Minimize the functionality of a component** – it is easier to find and fix possible security flaws in a system composed of smaller components than in a monolithic system.

7. **Configure safely and use safe defaults** – we should make the initial system safe as it eases the potential reconfiguration in the future.

8. **Load initialization values safely** – the system often loads these values to set the defaults. It relates to the previous point.

9. **Fail safe** – we should always define a straightforward and safe recovery process from an error.

10. **Avoid race conditions** – multi-threading and multi-processing should be handled carefully by the experienced developers.

11. **Trust only trustworthy channels** – always evaluate if the source of data is reliable and trustworthy.

12. **Setup a trusted path** – always evaluate a process path and look for the possible sources of errors to avoid the leakage of secure information.

13. **Use internal consistency** – this refers to the usage of coding standards and application of company policies during the development process.

14. **Self-limit resources** – as abnormal termination may loose some information.

15. **Prevent cross-site malicious content** – avoid data from unreliable sources which may lead to the leakage of other users' data.

16. **Foil semantic attacks** – implement procedures to prevent social engineering on your users like phishing, etc.

17. **Be careful with data types** – many bugs and security flaws are caused by invalid usage of data types.

18. **Carefully callout to other resources** – some resources like library routines may require some additional system information to operate.

19. **Send information back judiciously** – minimize feedback for unreliable users.

The best practices we summarized in the article can be used as a guideline or a checkbox. The advantage of these recommendations is their flexibility and freedom to adapt them to a particular case. Based on the project's complexity and scope, you can use them as an addition to your security process framework or as a core guideline for your development process.

Conclusion

In the article, we first defined, in Section 1, the *Software Development Life Cycle (SDLC)* as one of the fundamental parts of software engineering. We looked at the different steps (phases) of the cycle and briefly put them into the software development security context.

The SDLC description aimed to acquire the reader with the life of the software and its different phases. This section's outcome should be the question of how to ensure the security of development in all phases of the SDLC.

We addressed the question from the first section in the second one where we described three different security processes used in the software development, namely Microsoft's *Security Development Lifecycle* (SDL) [5], OWASP's *Comprehensive, Lightweight Application Security Process* (CLASP) [6], and McGraw' *Touchpoints* [7].

In Section 2, we also described each of the processes' key characteristics and features. In Section 3 we presented three different sets of security best practices. Each set looked at the security from a different level of abstraction. The article's overall purpose was to acquaint the user with the possible ways to ensure security during the development – using the security process frameworks, security best practices, or combining them.

References

1. RANDELL, Brian. The 1968/69 nato software engineering reports. *History of Software Engineering* [online]. 1996, vol. 37 [visited on 2020-11-28]. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.45.8087&rep=rep1&type=pdf#page=41.
2. JEVTIC, Goran. *What is the Software Development Life Cycle?* [online]. © 2019 [visited on 2020-11-28]. Available from: https://phoenixnap.com/blog/software-development-life-cycle.
3. TOTURIALSPOINT. *SDLC - Overview* [online]. © 2020 [visited on 2020-11-28]. Available from: https://www.tutorialspoint.com/sdlc/sdlc_overview.htm.
4. ROUSE, Margaret. *service-level agreement (SLA)* [online]. © 2006–2020 [visited on 2020-11-28]. Available from: https://searchitchannel.techtarget.com/definition/service-level-agreement.
5. M. HOWARD, S. Lipner. The Security Development Lifecycle (SDL): A Process for Developing Demonstrably More Secure Software. *Microsoft Press*. 2006.
6. OWASP. *Comprehensive, lightweight application security process* [online]. © 2020 [visited on 2020-11-28]. Available from: https://owasp.org/.
7. MCGRAW, Gary. Software security. *IEEE Security & Privacy*. 2004, vol. 2, no. 2, pp. 80–83.
8. DE WIN, Bart; SCANDARIATO, Riccardo; BUYENS, Koen; GRÉGOIRE, Johan; JOOSEN, Wouter. On the secure software development process: CLASP, SDL and Touchpoints compared. *Information and Software Technology*. 2009, vol. 51, no. 7, pp. 1152–1171. ISSN 0950-5849. Available from DOI: https://doi.org/10.1016/j.infsof.2008.01.010. Special Section: Software Engineering for Secure Systems.
9. BRUNO, Luigi. *The Security Development LifeCycle* [online]. © 2013 [visited on 2020-11-28]. Available from: https://social.technet.microsoft.com/wiki/contents/articles/7100.the-security-development-lifecycle.aspx.
10. KNOBLOCH, Martin. *Developing Secure Applications with OWASP* [online]. © 2020 [visited on 2020-11-28]. Available from: https://owasp.org/www-pdf-archive/Developing_Secure_Applications_with_OWASP.pdf.
11. MCGRAW, Gary. *Software Security* [online]. © 2006 [visited on 2020-11-28]. Available from: http://www.swsec.com/.
12. YASAR, Ansar-Ul-Haque; PREUVENEERS, Davy; BERBERS, Yolande; BHATTI, Ghasan. Best practices for software security: An overview. 2008. Available from DOI: 10.1109/INMIC.2008.4777730.
13. DAVIS, N.; HUMPHREY, W.; REDWINE, S. T.; ZIBULSKI, G.; MCGRAW, G. Processes for producing secure software. *IEEE Security Privacy*. 2004, vol. 2, no. 3, pp. 18–25. Available from DOI: 10.1109/MSP.2004.21.
14. PETEANU, Razvan. Best Practices for Secure Development. 2003.