

Secure Development Lifecycle

Prof. Dr. Marc Rennhard, Dr. Stephan Neuhaus
Institut für angewandte Informationstechnologie InIT
ZHAW School of Engineering
rema | neut @zhaw.ch

Content

- **Overview** of the secure development lifecycle
- **Security activities** during secure software development
- How to **start adopting** a secure software development process
- Exercises

Goals

- You understand the fundamental idea of using **different security activities** to get a **secure development lifecycle**
- You know the different security activities, during which phases they are applied, and can provide **brief explanations** about their purpose
- You realize that the discussed approach to build secure software does not introduce a novel software development process, but that it can be applied to any process to **transform it into a secure development lifecycle**

Secure Development Lifecycle – Overview

The Need for Secure Software Development

- We have seen in the previous chapter that reactive approaches such as **penetrate and patch** or using **network security devices** do not work well to achieve secure software
- The only solution to really make software secure is by putting a **stronger focus on security** during the entire development process
 - This means employing a **secure development lifecycle (SDL) / a secure development process**
 - An SDL in general means that **security activities** are applied during different phases of the software development process

Secure Development Lifecycle (SDL) / Secure (Software) Development Process

These are just two expressions that mean the same: That security must be considered during the entire software development process.

Specific Processes for Secure Software Development

Several specific processes have been proposed, e.g.:

- Microsoft SDL: Microsoft Security Development Lifecycle
 - Microsoft-internal mandatory since 2004, as a response to security-related problems in Microsoft product
 - <https://www.microsoft.com/en-us/sdl>
- BSIMM: The Building Security In Maturity Model
 - Defined by a consortium of several companies, based on analyzing projects of > 30 companies
 - <https://bsimm.com>
- CLASP: Comprehensive Lightweight Application Security Process
 - OWASP project concerned with security during the software development lifecycle (basically a collection best practices)
 - https://www.owasp.org/index.php/CLASP_Concepts
- SAMM: Software Assurance Maturity Model
 - OWASP project that additionally defines maturity levels for the different disciplines
 - http://www.owasp.org/index.php/Category:Software_Assurance_Maturity_Model

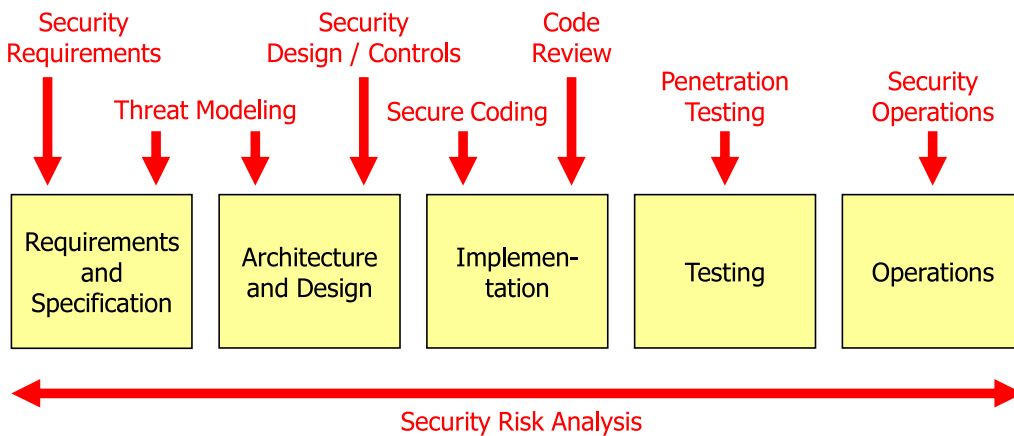
All of these processes are similar to each other and propose similar security activities that should be used during software development.


- The differences are in the actual details about how individual activities should be carried out, e.g., what checklists to use during threat modeling.
- This means that one does not have to stick to a particular process and one can easily mix components from different processes.


In this module, we therefore won't follow a particular one of these processes, but focus on the security activities (which can be found in any of them).

- For all activities, we then discuss best practices (taken from the processes above or from somewhere else) that have proven to work well in practice.
- Combined, the presented activities provide all building blocks for a secure development process.

Secure Development Lifecycle and Security Activities (1)



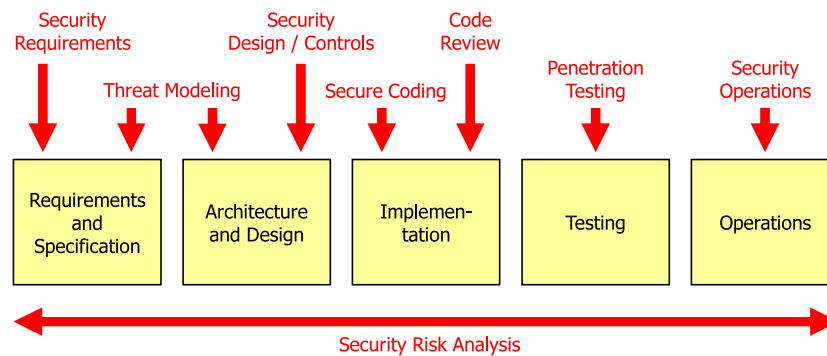
 Typical **phases** that are found in every software development process

 **Security activities** that should be applied during the different phases

Secure Development Lifecycle (SDL)

Basically, an SDL means applying different security activities during different phases of a software development process. So, for instance, when defining requirements, one should also define security requirements. Or when defining the architecture and design, one should also define the security design and the security controls. And so on.

Secure Development Lifecycle and Security Activities (2)



- A key concept of this approach based on security activities is that it can be **applied to any software development process!**
 - So it's not bound to a specific underlying software development process
- This is possible because the activities **focus on the individual phases** and not on the entire software development process
 - These phases can basically be found in every software development process (waterfall / iterative / agile / ...)

Security Activities are Independent of the Software Development Process

The image above with the development phases and security activities may be misleading as it looks a bit like the traditional waterfall model. But the approach based on security activities is by no means bound to the waterfall model or any other specific process and the image should only express that the activities are applied to the individual phases during software development (and not to the process in its entirety), and these phases can be found in every process (although they may have different names).

For instance, the *security requirements* activity is directly associated with the *requirements and specification* development phase. The activity «doesn't care» about the other phases and in what order or how many times they are executed, so it «doesn't care» about the overall development process. It only «cares» about the *requirements and specification* phase. The same holds for the other security activities: they are only associated with the corresponding phase of the development process and «don't» care about the other phases or the overall development process..

This means it doesn't matter whether you use the waterfall model, an iterative process, an agile process, or anything else as the basis because the security activities can always be applied to the corresponding phases of the used process. The main difference between classic processes (e.g., waterfall) and modern ones (e.g. iterative and agile) is that in classic processes, each phase is typically done just once, which means each security activity is also done just once. And in modern processes, the phases are typically done several times (in multiple iterations), which means the corresponding security activities are also done multiple times.

Applying the Security Activities to Specific Processes

- Examples of applying the security activities:
 - **Waterfall process:** Every phase is done once, which means each security activity are also done **once**
 - E.g., when the functional requirements are defined in the beginning, then the complete security requirements should also be formulated
 - **Iterative / agile processes:** The phases are repeated several times, so the security activities are also carried out **repeatedly**
 - But just like with functional aspects, only «to a degree» as it is reasonable during a particular iteration
 - E.g., when the first functional requirements are defined in the first iteration, then the corresponding first security requirements should be formulated
 - When additional functional requirements are added during a later iteration, then add the corresponding new security requirements and so on...

So you basically take your favorite software development process and **transform it into a secure development lifecycle (SDL)** by applying the security activities appropriately

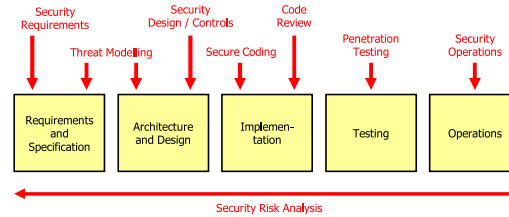
Example: Unified Process (UP)

- Security requirements are defined during iterations where «normal» requirements are defined (mainly inception and elaboration phases)
- Threat Modeling will be performed as long as the system design grows, focusing on the novel design components during each iteration
- The security design will grow and be refined as the system architecture and design develops during multiple iterations
- Code reviews are carried out along the development of the code
- ...

Security Activities

Security Requirements

- Defining security requirements is **the first security-related activity** that is carried out
 - Defining the right security requirements is highly important as they provide the **basis to end up with a secure system**
- Initially, this is usually done based on the functional requirements and the goal is to define «**some obvious security requirements**», e.g.:
 - If an application **transmits credit card information** from client to server → this results in a security requirement that *communication must be done over a cryptographically protected channel*
 - If an application provides **different areas for different users** → this results in a security requirement that *an access control mechanism must be used*
- However, these initial security requirements are usually **not enough** to achieve the desired security level
 - Therefore, additional security requirements should be **defined based on the results of the threat modeling security activity** (see next)
 - And also during further iterations in case an iterative processes is used



Security Requirements set the Basis for the Security of the System

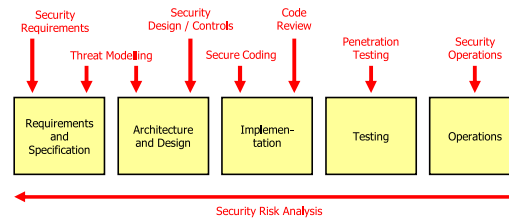
Just like functional requirements are highly important to end up with the right functionality, security requirements are highly important to end up with the right security measures and the right level of security. If important security requirements are forgotten, corresponding security measures most likely won't be integrated, which means we most likely end up with an insecure system.

Security Requirements based on Functional Requirements

At first, security requirements are usually driven by functional requirements. So, for instance, if credit card information is transmitted between client and server, then this directly leads to a security requirement that this should only happen over a cryptographically protected channel. However, as will be discussed in detail in later chapters, this usually won't be enough to really secure a system (because several important security requirements will likely be missed) and therefore, additional security requirements should be defined based on the results of the threat modeling security activity.

Threat Modeling

- The purpose of threat modeling is to **identify security design flaws** based on the security requirements or security controls that have already been defined
 - Or to put it differently: The goal is to find out whether the already defined security requirements and security controls are good enough to achieve the required security level
- Threat modeling is typically done as follows:
 - Imagine to be an attacker, i.e., **look at the system from the point of view of an attacker** – this puts you in the right mindset for threat modeling!
 - Based on this, **identify possible threats** against the system
 - Based on the threats, **identify vulnerabilities** in the current system design
 - If vulnerabilities have been detected: «Pass this information» to the security requirements activity to **define additional security requirements** to mitigate the vulnerabilities
- Threat modeling is a powerful, but also **critical activity** during an SDL
 - Because only the threats that are identified are likely to be prevented by adequate security requirements and security controls



Point of View of the Attacker

During threat modeling, always try to think like an attacker. I.e., ask yourself "if I were the attacker, what would my attack goals be? And how could I achieve this?" This helps a lot as it puts you in the right mindset to think about threats and attacks.

Focus of Threat Modeling

The focus of threat modeling is on finding security design flaws, i.e., on finding conceptual security problems based on the already defined security requirements and security controls. Or to put it differently: The goal is to find out whether the already defined security requirements and security controls are good enough to achieve the required security level. It's not about finding security bugs that happened because of implementation errors (for which there is the Code Review activity).

As an example, the following could happen during threat modeling of a web application:

- As a possible threat, «getting non-legitimate access to the admin area» is identified.
- Based on this, the current system design and the already identified security requirements and security controls are analyzed.
- If neither the security requirements nor the current security controls clearly show that there must be an access control mechanism where every single access to the web application must be checked, then this is identified as a vulnerability (security design flaw).
- As a result of this and to mitigate the vulnerability, a corresponding security requirements will be defined. But strictly speaking, this is then again part of the security requirements activity and this shows that in practice, the activities «security requirements» and «threat modeling» are often closely associated and done hand-in-hand.

Threat Modeling is Powerful, but also highly Critical!

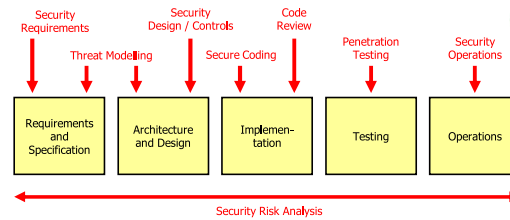
The power of threat modeling is that it «forces» us to view at the system from the attacker's point of view, which helps a lot to identify a wide variety of possible threats and attacks against the system. But it's also a critical activity because when not done correctly, then many potential attacks are probably missed and as a result of this, it's likely that we also do not define corresponding security requirements and security controls (because of a lack of awareness of the threats).

Not only Attacks, but also Attackers

During threat modeling, one should not only think about threats and attacks, but also about realistic attackers that may be interested in attacking the system. The reason for doing this is that the expected types of attacks depend on the attacker's capabilities. So if we have a system where an attacker can expect a significant financial benefit (e.g., a system that processes financial transactions), then we can expect powerful attackers and based on this, we certainly should consider sophisticated attacks.

Security Design / Controls

- The security design includes all security measures that are required **to secure the system adequately**
- The security design is based on the determined security requirements, i.e., the goal of the security design is to define **suitable security controls / security mechanisms** to implement all security requirements
 - E.g., if there's a security requirement that *strong user authentication must be used...*
 - ...then a reasonable security design / control decision could be, e.g., to use *two-factor authentication using a password combined with a biometric fingerprint*
- Security controls **should be chosen reasonably** with respect to the threat / risk they reduce
 - E.g., it is pointless to spend CHF 10'000 per year to protect from a threat that is expected to result in CHF 2'000 damage a year



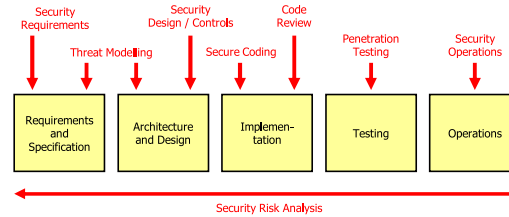
Security Requirements – Threat Modeling – Security Controls

It's important to stress that these three activities are closely associated. Security requirements can be defined «on their own», which is often done at the beginning of a project to get the first security requirements. But to truly get «all» relevant security requirements, one has to use threat modeling; so threat modelling is an important basis for the security requirements. And finally, the security design / controls are driven by the security requirements with the goal to define suitable / reasonable security mechanisms to implement (fulfil) the security requirements.

So basically, we can say that threat modeling is an important basis for the security requirements, which are an important basis for security design / controls.

Secure Coding

- Secure coding is about **implementing the source code in a secure way**



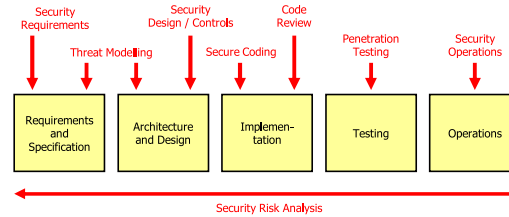
- Secure coding consists of two main aspects:
 - Implementing the **security design / controls as specified** so they are correctly enforced during runtime
 - Making sure **no security bugs are introduced** – either in the context of the security design / controls, but also in general (e.g., by introducing buffer overflow vulnerabilities)
- It's important to **use the right technology and third party libraries** that provide a good basis for a secure implementation
 - And you have to **understand the technology** you are using and what security features it provides and what it does not provide

Secure Coding

It's paramount that no security bugs are introduced during the actual implementation. If you have designed a very secure authentication mechanism but fail to implement this in a security bug-free way, then the mechanism is not of much value.

Code Review

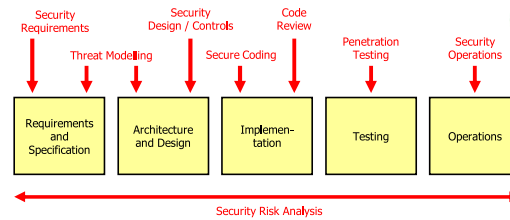
- Code review means **inspecting the code** and search for security problems



- The goal of code reviews is to detect **security bugs** introduced during the implementation of the code
 - As these account for approx. 50% of software security problems, good code review can uncover up to about 50% of all security problems
 - The other 50% are security design flaws that are virtually impossible to uncover by looking at code (and that should be uncovered during threat modeling)
- Code review is usually done using **automated code analysis tools**
 - **Manual code review** can be reasonable for some «very security critical» sections, but is usually too expensive for large amounts of code

Penetration Testing

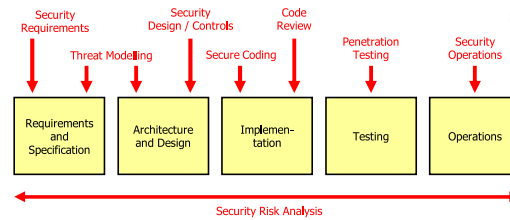
- Penetration testing means **taking the attackers view** with the goal to find and exploit vulnerabilities in the running system



- Valuable as it provides information of the **security of a complete system «in reality»**
- In the context of an SDL, a penetration test serves two purposes:
 - To verify whether the **security design / controls are correctly implemented**
 - To check that **no security bugs** were introduced during programming
- **Automated tools** are available, but they are not as powerful as a skilled human penetration tester

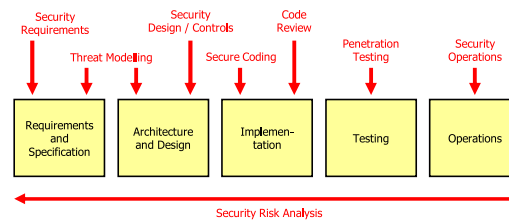
Security Operations

- Security operations includes all **security-related activities** that take place while the system is **in operation**
 - Updating the system (patching), backups, system & network monitoring,...
- Once a (valuable) system is in operation, it's almost guaranteed that **attack attempts will happen**
- Therefore, you should observe what's going on by **monitoring a fielded system** for various reasons
 - To learn about attack attempts and possibly vulnerable areas
 - This information should be fed back into the development process to decide about corrective actions
 - To detect a system compromise – because it may be that your preventive security measures are going to fail
 - Without monitoring, it's very difficult to detect that a system has been compromised



Security Risk Analysis

- Security risk analysis is a **horizontal activity** that complements the other security activities



- The purpose of security risk analysis is to **rate the risk of problems** that were detected when carrying out other security activities
- Examples:
 - If vulnerabilities are detected during **threat modeling**, risk analysis serves to assess the criticality of the individual vulnerabilities
 - If vulnerabilities are detected during **penetration testing**, risk analysis also serves to quantify the criticality of the vulnerabilities
- Risk rating is important to **decide what to do with a vulnerability**
 - If the risk is low, one can decide to do nothing about it
 - If the risk is high, effective countermeasures should be implemented

How to Start Adopting an SDL

Incremental Adoption of an SDL

- When moving towards a more secure development process, there's **no need to start applying all security activities at once**
 - You can start, e.g., by employing code reviews (which should already help a lot as it may detect up to 50% of all security problems)
 - A reasonable next step would be performing threat modeling (as it also covers security design flaws)
 - Penetration tests can also be used on their own and provide you with immediate feedback about the «real security» of a system
 - ...
- **Every added security activity will increase the security** of the resulting software...

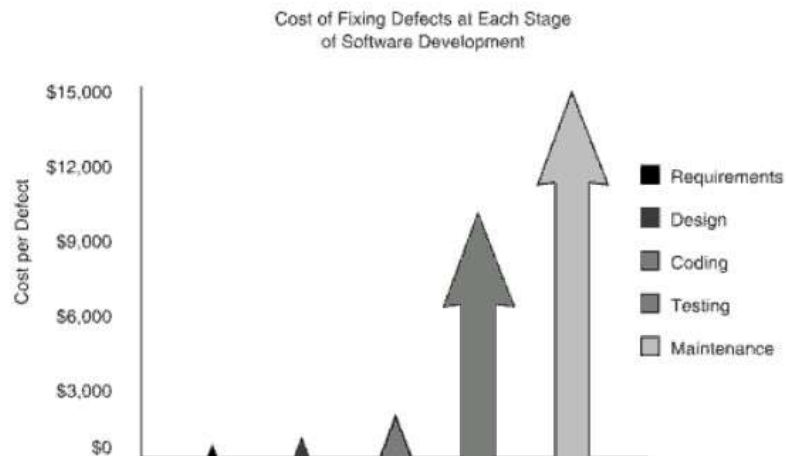
...but the long-term goal should be to adopt all security activities because they complement each other!

Security Activities complement each other

That's an important point. When you are leaving out one of the activities, something will be missing. So, for instance, when not using threat modeling, then it's very likely that your system is not going to provide protection from all realistic attacks. Or if you leave out penetration testing, then you'll never have real feedback about the security of the system in operation. Similar arguments can be brought up for all the other activities.

Fixing Earlier is Better (1)

- Just like with functional software defects, **finding and fixing security-related defects early in the development process is cheaper**



- Therefore, the **«early security activities»** are especially important
 - They help to not only to detect defects, but can prevent them

Fixing Earlier is Better

Of course, if there's a simple programming bug (whether it's security-relevant or not), then it's usually not expensive to fix this even late in the development process. So the saying «fixing earlier is better» mainly applies to fundamental software design problem. For instance, if it turns out late during software development that the designed and implemented access control mechanism is not suited at all to achieve the desired level of security, then it will most likely be quite expensive to fix this as redesigning and changing an access control mechanism affects many system components, which means that significant portions of the code must be changed.

Early Activities Prevent Defects

That's an important point. The late activities are security testing-oriented (especially code review and penetration testing), so they help to *detect* security problems. But the early activities, especially security requirements, threat modeling and security design / controls help to *prevent* security problems in the sense that if these activities are done with care, then the likelihood of bad surprises during security testing (and costly redesigns of security components) is relatively small.

The illustration above was directly extracted from the book Gary McGraw, *Software Security – Building Security In*, Addison-Wesley Software Security Series.

Fixing Earlier is Better (2)

- But in reality, security is **often considered only towards the end of the development process** (if it is considered at all)
 - For instance, many companies do not really consider security during development, but **only do a penetration test** just before the release date, hoping not much is found so they can start productive operation
 - Sometimes, this is simply done so the company can blame the (external) penetration testers in case of a security breach → «Cover your Ass Security»
- Doing only a penetration test is **better than not doing anything at all**, but what happens if serious security design problems are uncovered?
 - Usually, there will be **quick fixes** without truly solving the underlying problem, especially if the necessary effort (time, money) would be significant
 - This likely means the problem **will show up again** (somewhere else in the code or at the same place through a variation of the attack)...
 - ...and we are back in the «penetrate and patch» cycle
- To avoid this and to get truly secure software it is therefore paramount to also **adopt the early security activities in the lifecycle!**

Security Testing vs. Unit-/Functional Testing

In a way, security testing is at a similar stage as functional testing was many years ago. It was recognized that early and continuous Unit-/functional testing is beneficial and correspondingly, this is often done in software projects. However, security testing still often is only done late (if at all), often in the form of penetration testing.

Which Security Activities in which Chapters in this Module?

- **Security Requirements and Threat Modeling**
 - Chapter 9: Security Requirements Engineering and Threat Modeling
- **Security Design / Controls and Secure Coding**
 - Chapter 3: Software Security Errors
 - Chapter 4: Java Security
 - Chapter 5: Secure Design Principles
 - Chapter 7: Developing Secure Traditional Web Applications
 - Chapter 8: Developing Secure Modern Web Applications
 - (And a lot of this was covered in module IT-Sicherheit!)
- **Code Review**
 - No lecture, will be discussed in Security Lab
- **Penetration Testing**
 - Chapter 6: Web Application Security Testing
- **Security Operations**
 - Not part of this module (but part of Software and System Security 2)
- **Security Risk Analysis:**
 - Chapter 10: Security Risk Analysis

Which Security Activities in which Chapters

This is simply an overview that shows in which chapters in this module the different security activities will be discussed. Note that the security activities are not discussed «in order». In particular, the first two security activities, security requirements engineering and threat modeling, are only discussed towards the end of the semester. But that's reasonable because these two security activities require a lot of know how about attacks and security design, which therefore have to be discussed first.

Summary

- Employing a **Secure Development Lifecycle (SDL)** is the only reasonable way towards secure (enough) software
 - Experience tells us that reactive security measures don't work well
- The SDL we are discussing in this module is not a new software development process but consists of a set of **security activities** that can **be applied to virtually any existing development process**
 - This is possible because the activities **focus on individual phases** which can be found in every software development process
 - With iterative / agile processes, the security activities are applied repeatedly
- When moving towards an SDL, the security activities can be **adopted incrementally**
 - But the long term goal should be to eventually employ all security-related activities because they complement each other