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.:

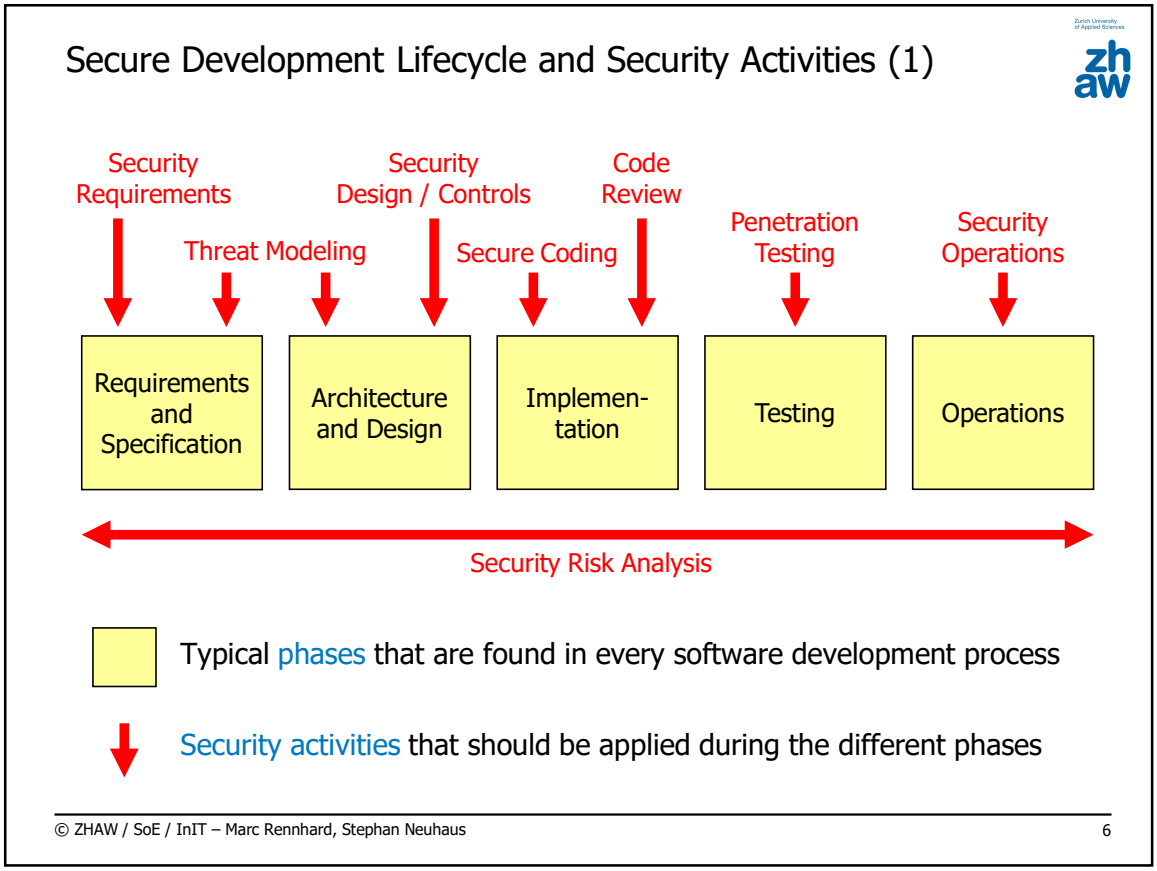
- Microsoft SDL: Microsoft Security Development Lifecycle
  - Microsoft-internal mandatory since 2004, as a response to security-related problems in Microsoft product
  - *https://www.microsoft.com/en-us/sdl*
- BSIMM: The Building Security In Maturity Model
  - Defined by a consortium of several companies, 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).

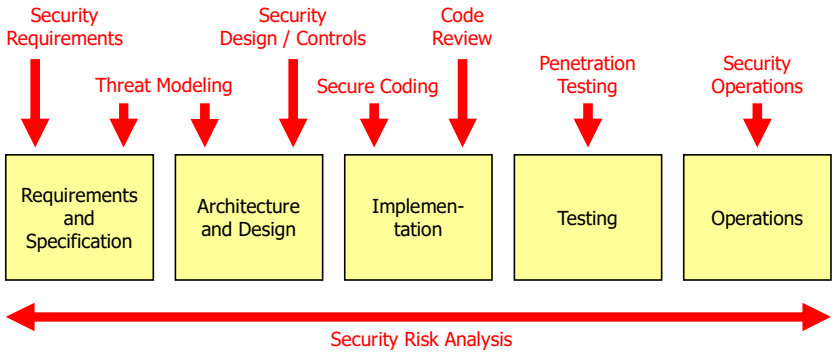
- For all activities, we then discuss best practices (taken from the processes above or from somewhere else) that have proven to work well in practice.
- Combined, the presented activities provide all building blocks for a secure development process.



Secure Development Lifecycle (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...

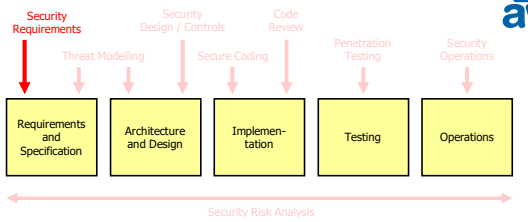
So you basically take your favorite software development process and transform it into a secure development lifecycle (SDL) by applying the security activities appropriately

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 (1)



- 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 using checklists with the goal is to define «some obvious basic security requirements», e.g.:
  - If an application transmits credit card information from client to server → this may result in a security requirement that *«Communication must be done over a cryptographically protected channel»*
  - If an application provides different areas for different users → this may result in a security requirement that *«An access control mechanism must be used, and authorization must be checked with every single access»*

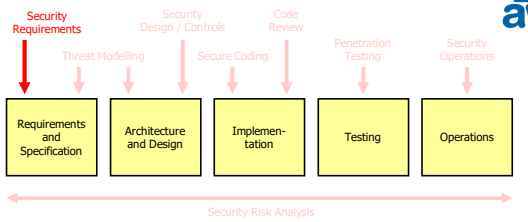
Security Requirements set the Basis for the Security of the System

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.

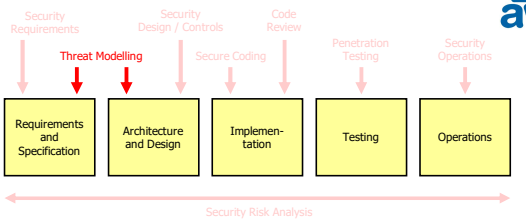
Security Requirements (2)



- However, these initial security requirements are usually not enough to achieve the desired security level
  - Because there are usually several important security requirements that are difficult to derive from functional requirements and that are typically not included on generic checklists
  - Therefore, additional security requirements should be defined based on the results of the threat modeling security activity (see next)

- Security requirements should be specific enough so they can be clearly understood, but should not include the technical details how to implement the requirement
  - Because the details how a requirement is fulfilled in a reasonable way depends on technology decisions which follow during later phases (overall architecture & design, frameworks, programming languages,...)

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 and 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.

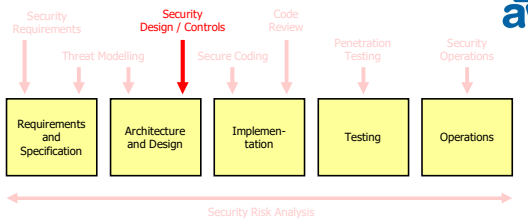
Threat Modeling is Powerful, but also highly Critical!

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 (1)



- The security design / controls include all specific security measures to secure the system adequately

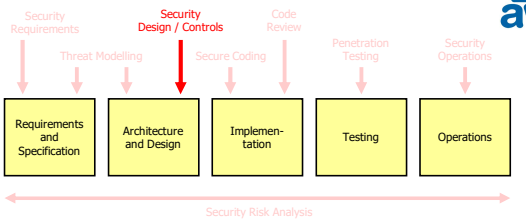
- The security design / controls are based on the determined security requirements, i.e., the goal of the security design / controls is to define suitable security mechanisms to implement all security requirements
  - E.g., if there's a security requirement that *«Strong user authentication must be used»*...
  - ...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 Requirements – Threat Modeling – Security Design / 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 be the security requirements with the goal to define suitable / reasonable security mechanisms to implement (fulfil) the security requirements.

To summarize, one can say that threat modeling is an important basis for the security requirements, which are an important basis for security design / controls.

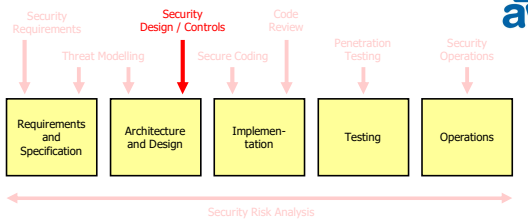
Security Design / Controls (2)



- Depending on the security requirements that must be fulfilled, completely different security design / controls decision can be made, as illustrated with three examples:

- Security requirement 1: *«Use protection measures to prevent buffer overflow vulnerabilities»*
  - Decision (assuming C is used as programming language): *«Check the length of target buffers before writing to them and make sure not to write beyond the end of the buffer; don't use insecure functions such as strcpy and use the more secure alternatives such as strncpy, and turn on compiler and OS security features (stack canaries, ASLR)»*
  - So in this case, the specific security measures to fulfill the requirement are mainly guidelines how to write code, how to compile it and how to run the program, and there's no impact on the system architecture and design

Security Design / Controls (3)



- Security requirement 2: *«An access control mechanism must be used, and authorization must be checked with every single access»*
  - Decision (assuming Jakarta EE is used as underlying technology): *«Use the role-based access control mechanism included in Jakarta EE (i.e., the Jakarta EE Security API) to enforce access control»*
  - So in this case, the specific security mechanism to fulfill the requirement is to use a specific security control provided by the used technology, which has a relatively small impact on the system architecture and design
- Security requirement 3: *«Administrative areas in the webshop must isolated from public and customer areas as much as possible to minimize the risk that normal users can abuse powerful admin functionality»*
  - Decision: *«Use a completely separate web application for administrative functions (that only shares the database with the actual webshop) and in addition allow access to this web application only from the company-internal admin network or over VPN-access that is only provided to administrators»*
  - So in this case, the specific security mechanism has a major impact on the overall architecture and design and also on networking (firewalls, VPNs)

Security Requirements should not include Technical Details

The second security requirement above shows nicely why security requirements should be «technology-agnostic»: The security design / controls decision is heavily influenced by the technology choice (Jakarta EE in this case) and it simply would be pointless to add more technical details to the security requirement to make sure that the choice about the specific security measures is left to the security design / controls activity where underlying technology decisions can be taken into account.

This is of course somewhat idealized and in practice, security requirements are not always formulated in a technology-independent way, in particular if the team determining the security requirements knows already what technologies will be used, and in this case, there's usually no big harm when security requirements are a bit too specific. However, as a general recommendation, always try to formulate security requirements in a way so that the specific technical solution is not already predetermined by the requirement, so that those that decide on the specific security design / controls measures have maximal flexibility and can take the underlying system architecture and design (and technology choices) into account.

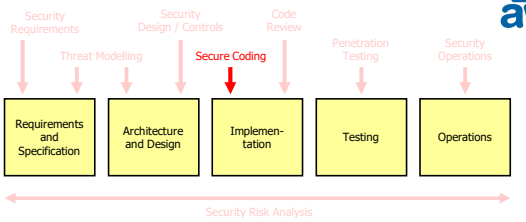
Security Requirements must be defined early

Security requirements 2 and 3 also show nicely why such requirements must be specified early, before the system architecture and design are defined in detail.

If the second requirement is not there before the design is done and implementation starts, it may easily happen that it won't clearly be defined how access control should be done specifically in the application. As a result of this, the developers may end up implementing their own solution without following a clear strategy or without considering the mechanisms provided by the underlying technology (Jakarta EE in this case). If multiple developers are involved, it may happen that access control will be done differently in different areas of the application and it's almost guaranteed that this will lead to security problems.

If the third requirement is not there, it's likely that «one web application» will be developed without splitting the application so that a separate one is used for admin functionality. If it is then decided later during development that splitting the application is needed for security reasons, this will require a significant reengineering effort, which costs time and money.

Secure Coding



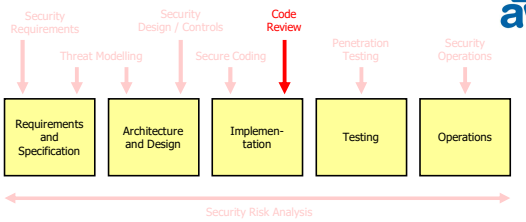
- Secure coding is about implementing the source code in a secure way
- Secure coding consists of two main aspects:
  - Implementing the security design / controls as specified so they are correctly enforced during runtime
  - Making sure no security bugs are introduced – either in the context of the security design / controls, but also in general (e.g., by introducing buffer overflow vulnerabilities)
- Secure coding is mainly a question of «being careful» when writing code (i.e., be security-aware and ask yourself what could go wrong) and of understanding the technology you are using (e.g., what security features the used framework/library/language provides or not)
  - In addition, there are some helpful tools and technologies that should be used (secure coding checklists, security features added by compilers, never ignore compiler warnings (e.g., about insecure or deprecated functions),...)

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

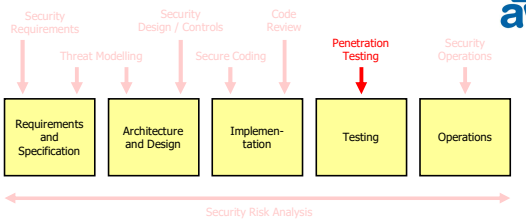


- The goal of code reviews is to detect security bugs introduced during the implementation of the code
  - As these account for approx. 50% of software security problems, good code review can uncover up to about 50% of all security problems
  - The other 50% are security design flaws that are virtually 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

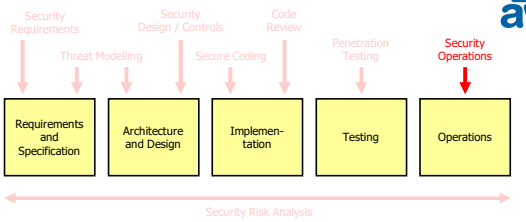
Penetration Testing

- 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 requirements are fulfilled in the implemented system
  - To check that no security bugs were introduced during programming
- Automated tools are available and should be used, but they are not as powerful as a skilled human penetration tester

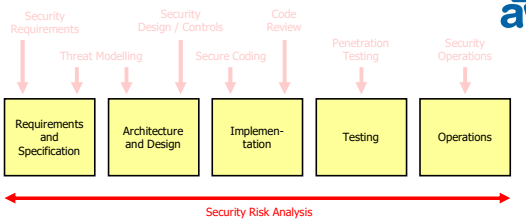
Security Operations



- 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...

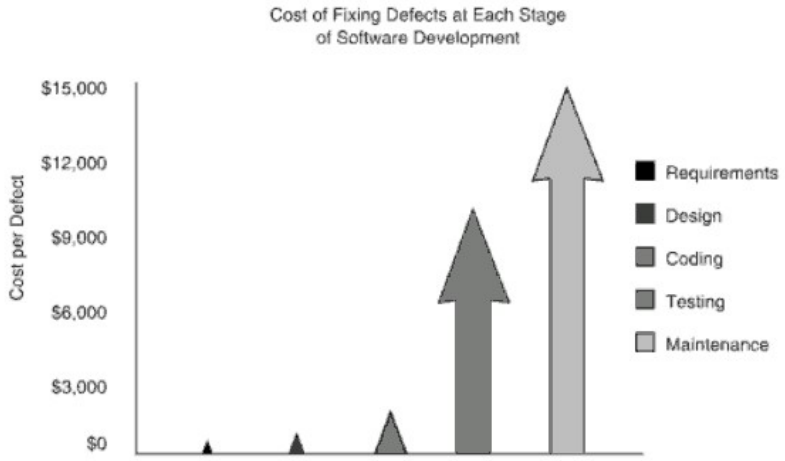
...but the long-term goal should be to adopt all security activities because they complement each other!

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 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
- All security activities:
  - Chapter 2: Secure Development lifecycle (overview of all activities)
  - Chapter 5: Fundamental Security Principles (have an impact in all activities)

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 controls, which therefore have to be discussed first.

What will NOT be covered in this Module?

- While this module discusses the entire SDL and all security activities (except security operations), the individual activities can – obviously – only be covered to a certain degree
- Specifically, this means, e.g., the following:
  - The focus will be on programming languages (and corresponding secure coding aspects) you already know: Java and a bit of C
  - When considering example applications, we focus on application types you already know: mainly web applications and mobile apps
  - Likewise, when discussing security requirements and security design / controls, we focus on a relevant subset of them (without claiming completeness) and do this mainly in the context of the application types mentioned above
  - When attacking systems during penetration testing and doing threat modeling, we focus on some of the most important attacks and threats (without claiming completeness) in the context of these application types
- However, what you learn will definitely be good enough for you to start applying an SDL in practice
  - But to get truly proficient, you first of all have to practice (a lot) and must get additional knowledge and understanding of threats, attacks, security design aspects, secure coding guidelines etc. in the context of the software type you are developing and the technology you are using

Summary

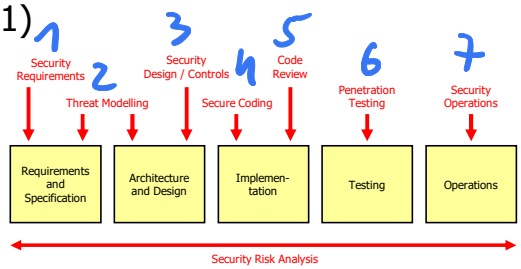
- Employing a Secure Development Lifecycle (SDL) is the only reasonable way towards secure (enough) software
  - Experience tells us that reactive security measures don't work well
- The SDL we are discussing in this module is not a new software development process but consists of a set of security activities 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

Exercises

Prüfungsaufgabe!!!

Exercise 1: Security Activities (1)

Assign the tasks below to the correct security activity (e.g., Security Requirements, Threat Modeling,...) of the secure development lifecycle



2 • Identify potential attackers or attack groups (e.g., script kiddies, organized cyber criminals, nation states,...) that may be interested in attacking the system

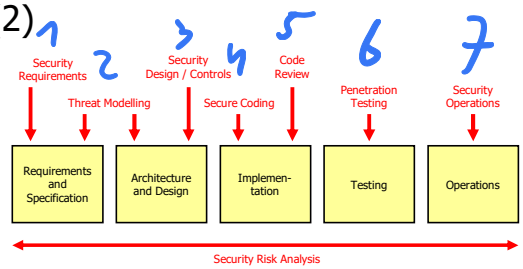
7 • Once a month, use a password cracking tool to check if there are users that are using weak passwords

5 • Before pushing code to the Git repository, a source code analyzer that is integrated in the IDE is used to analyze the code with respect to security bugs

4 • Use a checklist with guidelines to prevent typical security-relevant programming mistakes in iOS apps

Exercise 1: Security Activities (2)

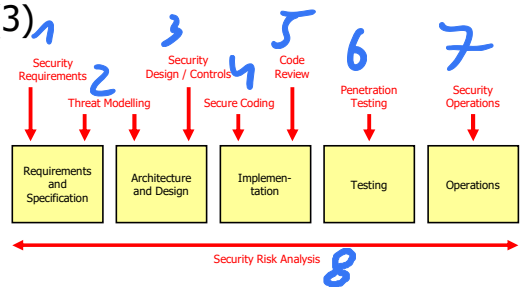
Continued...



- Decide whether to use login codes based on SMS messages or Google Authenticator as the 2nd authentication factor
- Hire a company to find security defects in an e-shop web application by interacting with the running system
- Configure the compiler so that safeguards to prevent exploitation of buffer overflow vulnerabilities are turned on
- Analyze the security architecture of a military communication system with the goal to identify security design flaws

Exercise 1: Security Activities (3)

Continued...



- Based on a list of identified security design flaws, determine additional security properties that should be considered
- Employ a security monitoring system to detect suspicious communication patterns (e.g., scanning traffic) between the control systems of a power plant
- Use a methodology to rate the criticality of vulnerabilities that are detected during a penetration test
- Depending on a specified security requirement, choose an appropriate access control mechanism

Exercise 2: Security Defects and Security Activities (1)

- Assume the following code is part of a server program and is used to check the password submitted by a user over the network
  - Also, assume that the array *receivedPwd* is long enough to hold any user password and that it has been pre-filled with 0-bytes

```
1  read(socket, receivedPwd, sizeof(receivedPwd)-1);
2  comp = memcmp(receivedPwd, correctPwd, strlen(receivedPwd));
3  if (comp != 0)
4     return (BAD_PASSWORD);
```

- Your tasks:
  - Identify the defects in this code (hint: there are 3 defects)
  - For each of them: Which security activity should be helpful to detect it?

- Defect 1: Return value of *read* is not checked. This is more poor programming practice than a security problem, but it may lead to non-predictable program behavior and should be prevented.
  - This is a simple programming bug that can easily be detected by a source code analyzer during Code Review

Example

This example was taken from *Gary McGraw, Software Security – Building Security In, Addison-Wesley Software Security Series*. It does the following:

- Reads data entered by a user into *receivedPwd* (at most sizeof(*receivedPwd*)-1 bytes)
- Compares the entered input with a stored password (*correctPwd*)
- Returns *BAD_PASSWORD* if the entered password does not match the stored one

Assume that *receivedPwd* is declared as a local variable within the function, so *sizeof* indeed returns the size of the array in bytes.

Exercise 2: Security Defects and Security Activities (2)

```
1  read(socket, receivedPwd, sizeof(receivedPwd)-1);
2  comp = memcmp(receivedPwd, correctPwd, strlen(receivedPwd));
3  if (comp != 0)
4     return (BAD_PASSWORD);
```

- Defect 2: Entered password is directly compared with the correct password, so the application likely stores passwords in plaintext
  - This is a security design flaw which can be found during Threat Modeling
- Defect 3: Using *strlen(receivedPwd)* means the comparison succeeds not only if the entered password matches the stored one, but also if a prefix of the stored password is entered. It even succeeds if the user enters a password of length 0.
  - This is a more complex programming bug that may be uncovered during Penetration Testing

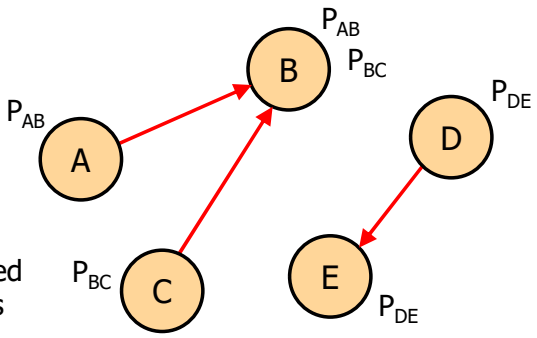
Exercise 2: Security Defects and Security Activities (3)

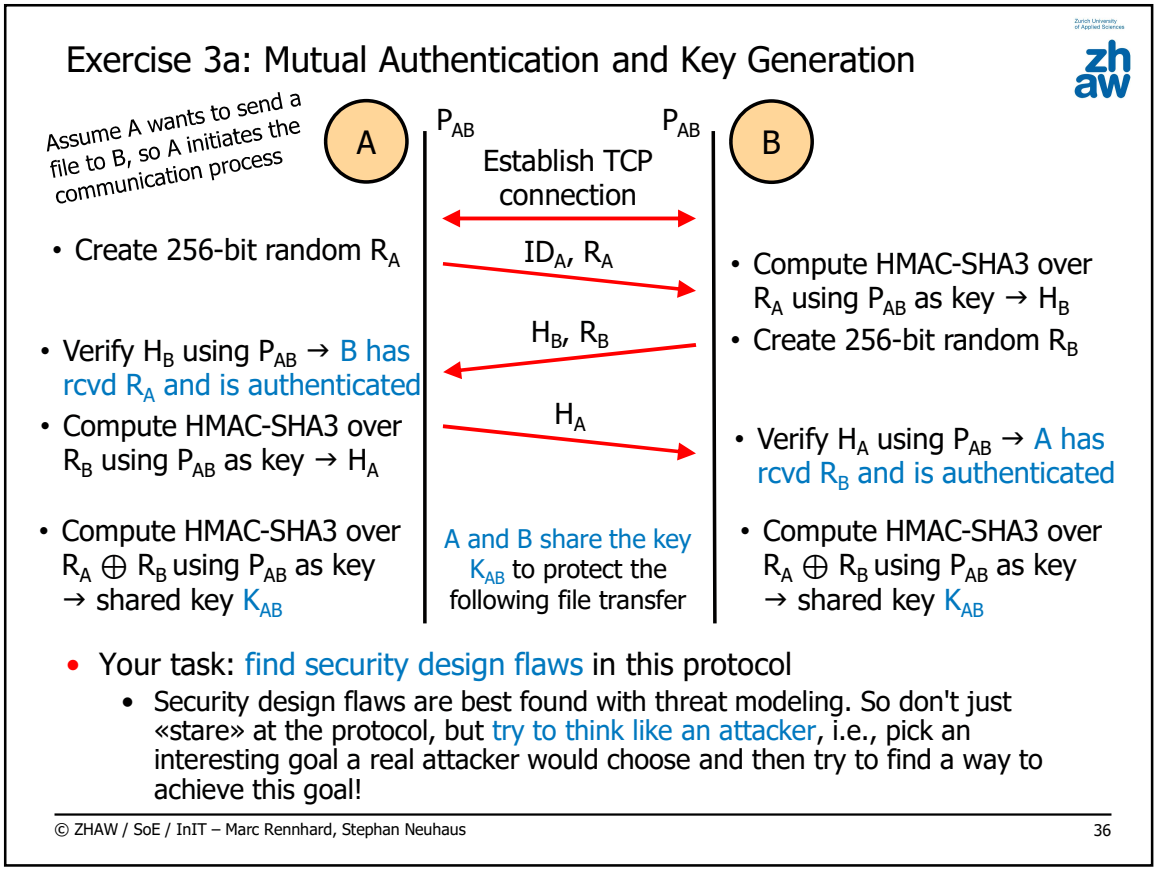
```
1  read(socket, receivedPwd, sizeof(receivedPwd)-1);
2  comp = memcmp(receivedPwd, correctPwd, strlen(receivedPwd));
3  if (comp != 0)
4     return (BAD_PASSWORD);
```

- Defect 3: The password can be determined character by character (and can possibly be used to login as the same user into other systems)
  - Try a, then b, then c..., if login works the 1st character is found (e.g., s)
  - Try sa, then sb, then sc..., if login works the 2nd character is found (e.g., e)
  - Eventually delivers the entire password (max. *password-length * 256 tries*)
  - If an attacker attempts this, this can be detected by a monitoring system that counts failed login attempts and that escalates this when a threshold is reached → Security Operations

Exercise 3: Finding Security Design Flaws (1)

- We consider a simple secure peer-to-peer file transfer system
  - Participants can send each other files (from sender to recipient)
  - To do this securely, any two participants that want to exchange files have established a common password
    - Assume this was done securely and that each pair uses a different password
    - This password is used for mutual authentication and key generation to protect the file transfer
  - The systems supports multiple sessions, i.e., a recipient can handle multiple senders at a time
  - Assume that we are still early in the development process, so the protocol has been designed but implementation has not started yet → perfect time to analyse this design for security problems!



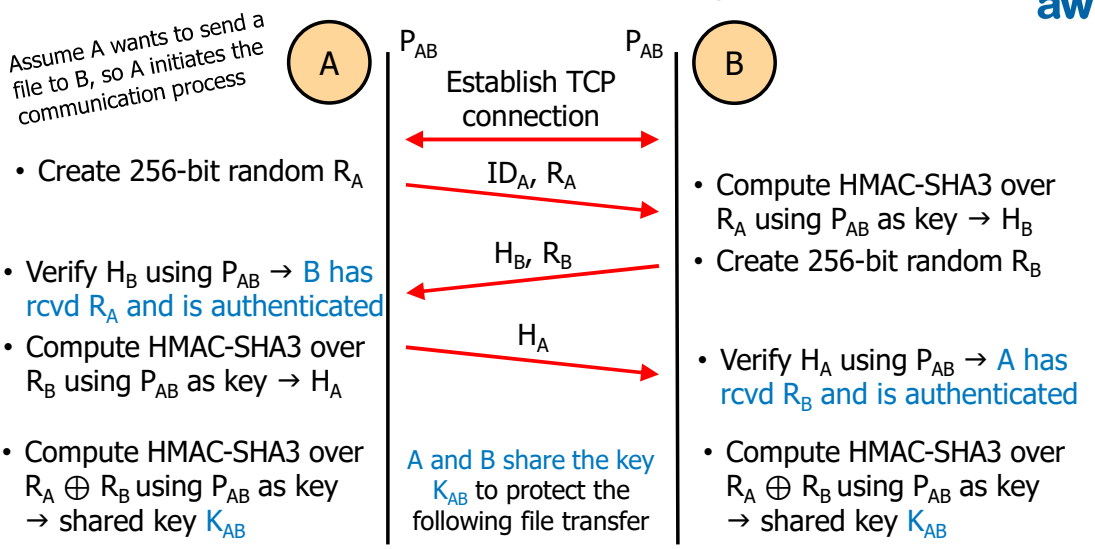


Protocol Simplifications

Note that the protocol does not include all details to keep the example relatively simple. For instance, the mechanism to support multiple sessions is not included. For the analysis, assume that such a mechanism is included and secure. This means for instance that reply attacks are not considered to be an issue.

The hash function can use any hash length (e.g., 256 bits), but it is assumed that the same hash length is used with all HMAC-SHA3 operations. Also, it is assumed the identities of the participants (ID_A etc.) are publicly known.

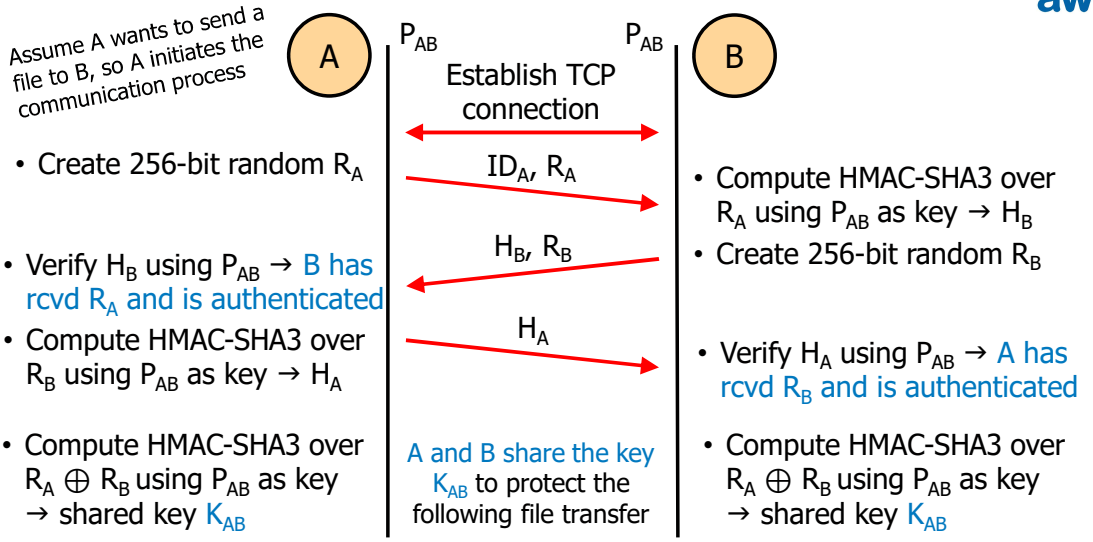
Exercise 3a-1: Mutual Authentication and Key Generation



- First attack goal: Determine the shared key K_{AB}
  - Assume that you are a Man-in-the-Middle (MITM) that can eavesdrop on all messages that are exchanged, that can modify the messages in transit, and that can communicate with all participants

Exercise 3a-1: Solution

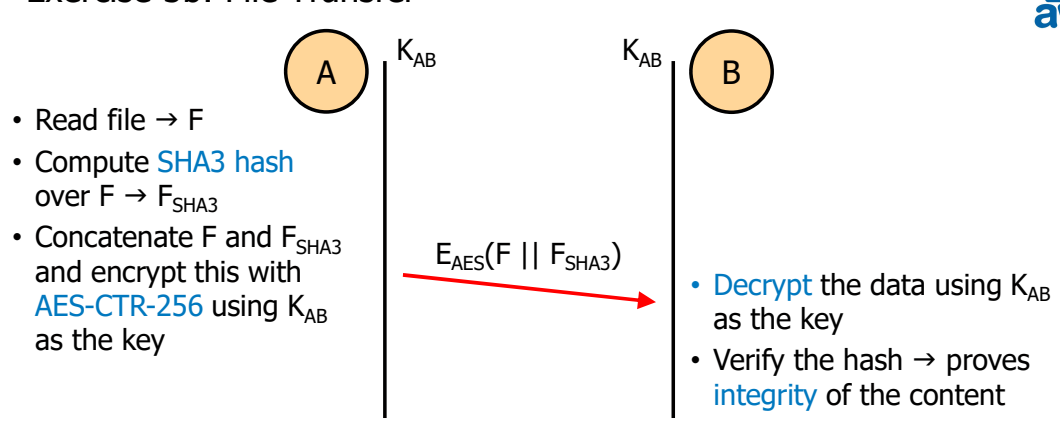
Exercise 3a-2: Mutual Authentication and Key Generation



- Second attack goal: Determine the password P_{AB}
  - Assume that you are a Man-in-the-Middle (MITM) that can eavesdrop on all messages that are exchanged, that can modify the messages in transit, and that can communicate with all participants

Exercise 3a-2: Solution

Exercise 3b: File Transfer



- Read file → F
- Compute SHA3 hash over F → F_{SHA3}
- Concatenate F and F_{SHA3} and encrypt this with AES-CTR-256 using K_{AB} as the key

- Decrypt the data using K_{AB} as the key
- Verify the hash → proves integrity of the content

- Your task: find security design flaws in this protocol
  - Again, pick an interesting goal a real attacker would choose and then try to find a way to achieve this goal!
  - Assume that you are a Man-in-the-Middle (MITM) that can eavesdrop on all messages that are exchanged, that can modify the messages in transit, and that can communicate with all participants

Exercise 3b: Solution

Exercise 3: Conclusions

- All identified defects are security design flaws
  - They are typically not detected by looking at code and code analysis tools usually can't detect them
  - Instead, they are typically detected by manually analyzing design documents (protocol design, use cases, flow diagrams, etc.)
- In addition, it's important to use threat modeling to look for security design flaws, i.e., that you think like an attacker – this will significantly increase the probability that defects are detected
- Also, it's important to analyze the design with respect to security as early as possible, before the design is implemented
  - Because if you detect the flaws late in the development lifecycle, it will be much more expensive to fix them
  - This underlines that the «early security activities» are especially important, as they help to not only to detect defects, but as they can prevent them so they never find their way into the code