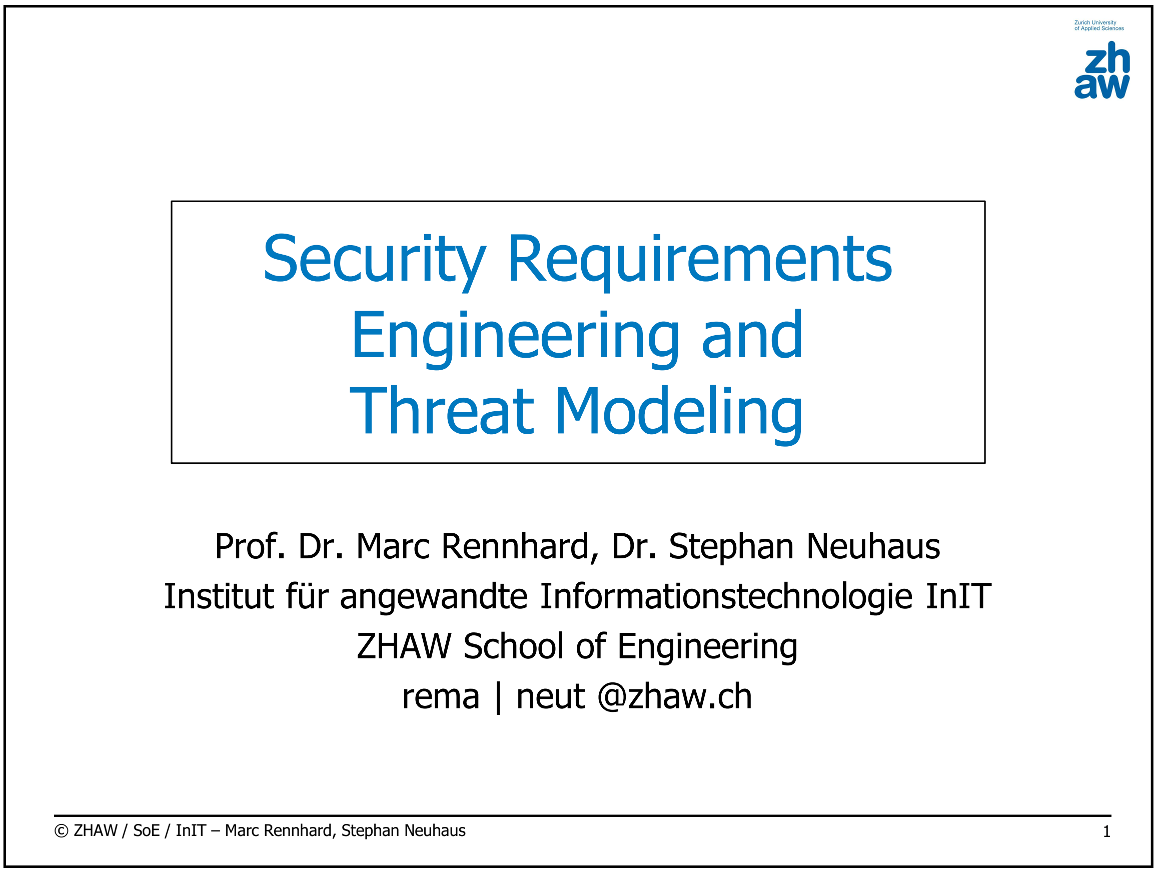


Zurich University of Applied Sciences

Security Requirements Engineering and Threat Modeling

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

This topic is covered, e.g., in the following books:

- *Michael Howard and Steve Lipner, The Security Development Lifecycle, ISBN 978-0735622142* (Chapter 9: Stage 4: Risk Analysis)
- *Gary Mc Graw. Software Security: Building Security In. ISBN 978-0321356703* (Chapter 5: Risk Analysis, Chapter 8: Abuse Cases)
- *Brook S.E. Schoenfield. Securing Systems. ISBN 978-1482233971*
- *Adam Shostack. Threat Modeling: Designing for Security. ISBN 978-1118809990*

There are also some online resources:

- Security Quality Requirements Engineering (SQUARE) methodology: *https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=484884*
- Microsoft's Threat Modeling Approach, which is part of the Microsoft Security Development Lifecycle process: *https://www.microsoft.com/en-us/SDL/about/default.aspx*
- *Tom Olzak. A Practical Approach to Threat Modeling. http://docplayer.net/16289507-A-practical-approach-to-threat-modeling.html*

The slides in this chapter partly contain information from these books and articles.

Content

- Introduction to security requirements engineering and threat modeling
- The process of security requirements engineering and threat modeling based on the STRIDE method
- Documenting threats and security requirements
- Appendix with further methods for security requirements engineering and threat modeling: attack trees, abuse cases, attack patterns

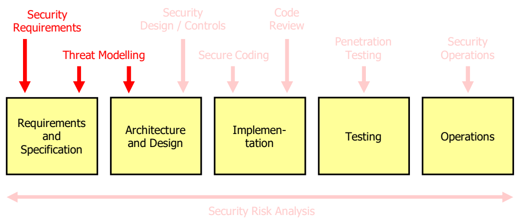
Goals

- You know and understand the security requirements engineering / threat modeling process and can apply it to analyze the security of software systems and to derive security requirements

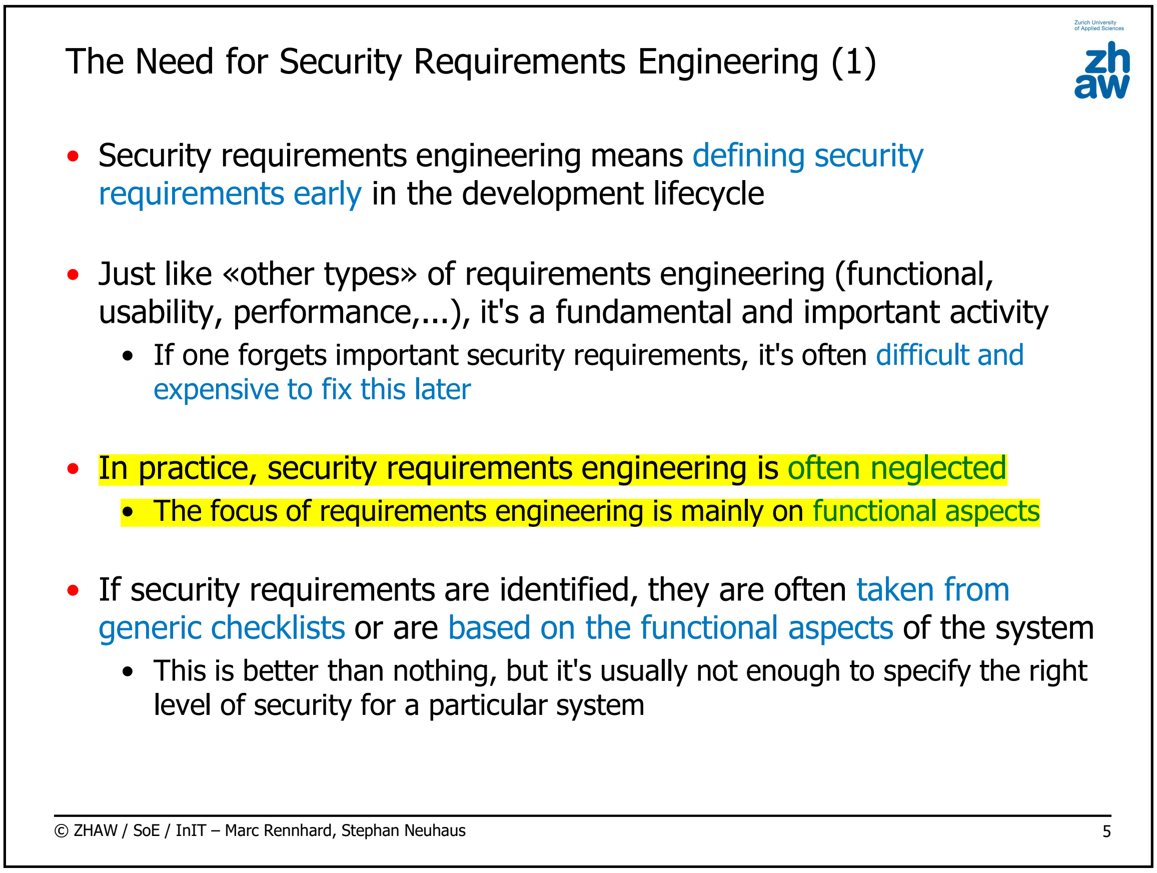
- You know and understand the STRIDE method and can apply it to identify threats / attacks against a system

- You know how to document threats and security requirements

- Security activities covered in this chapter:



Introduction



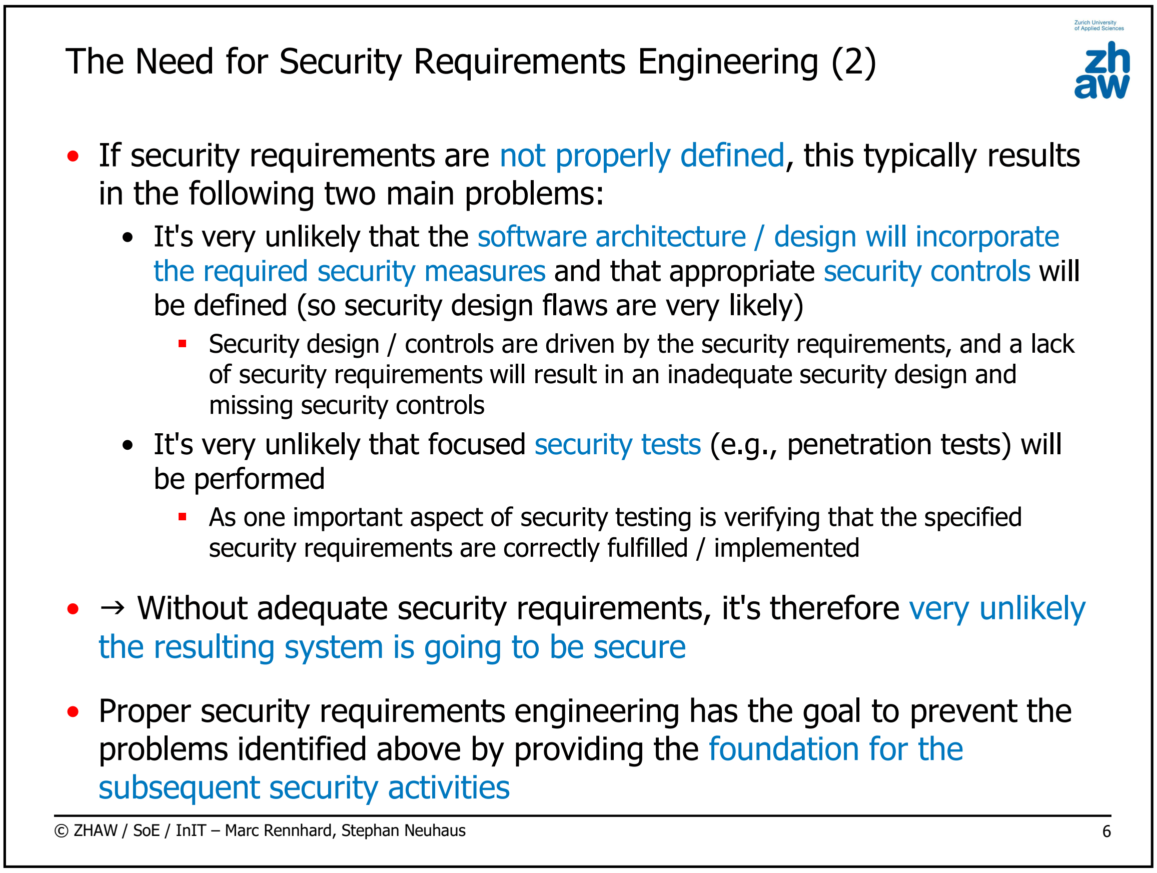
The Need for Security Requirements Engineering (1)

- Security requirements engineering means defining security requirements early in the development lifecycle
- Just like «other types» of requirements engineering (functional, usability, performance,...), it's a fundamental and important activity
  - If one forgets important security requirements, it's often difficult and expensive to fix this later
- In practice, security requirements engineering is often neglected
  - The focus of requirements engineering is mainly on functional aspects
- If security requirements are identified, they are often taken from generic checklists or are based on the functional aspects of the system
  - This is better than nothing, but it's usually not enough to specify the right level of security for a particular system

Checklists

Below you find a selection of checklists that may be helpful to specify security requirements and security controls. These lists can be a start (and may actually be enough to secure relatively simple standard applications that have limited value for potential attackers), especially if you are not yet experienced, but they usually only cover the most typical security measures and usually only focus on often-used application types such as web applications. In addition – being generic checklists – they are never a really good fit to the specific application *you* are developing and most likely won't include some requirements that you need in your specific application.

- *https://www.newnettechnologies.com/posters/sans-securing-web-applications.pdf*
- *https://www.sensedeep.com/blog/posts/stories/web-developer-security-checklist.html*
- *https://github.com/Probely/security_checklist/blob/master/Checklist.pdf*
- *https://www.security4startups.com/controls-checklist/*



The Need for Security Requirements Engineering (2)

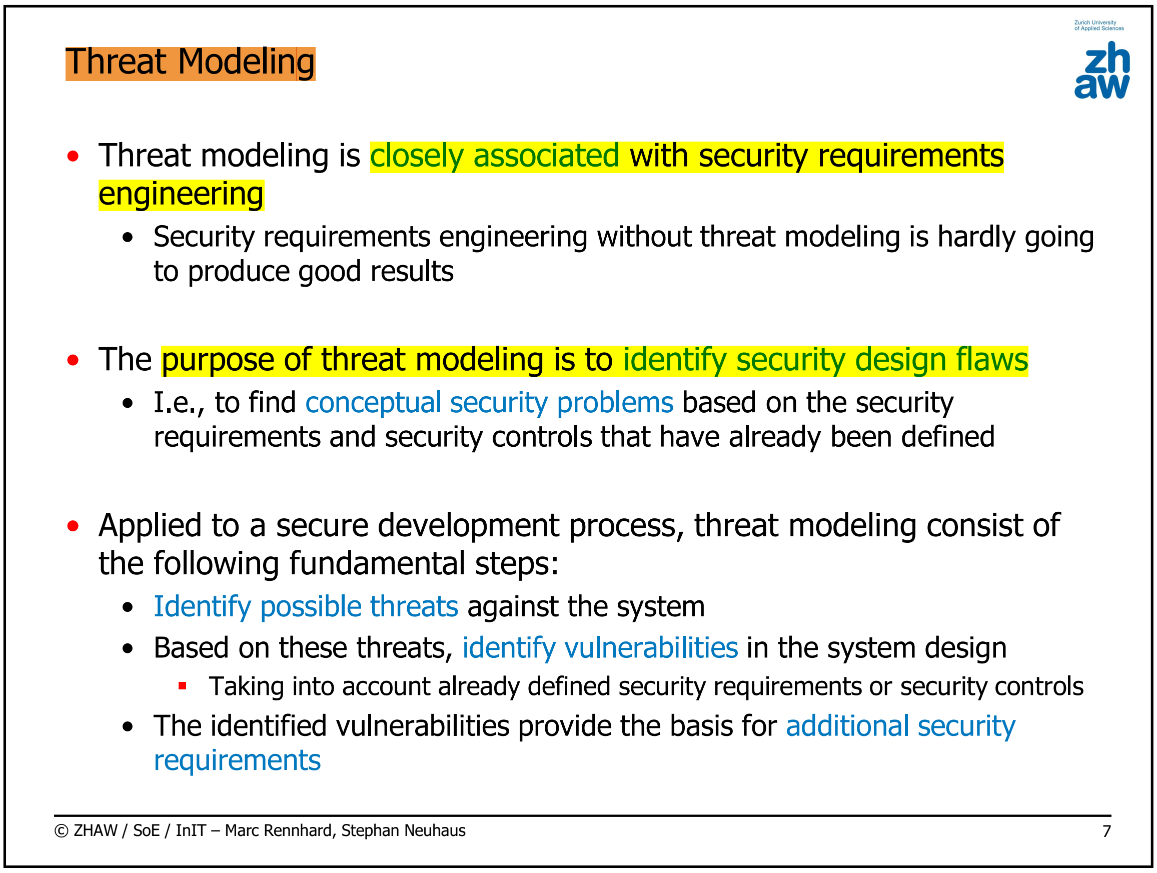
- If security requirements are not properly defined, this typically results in the following two main problems:
  - It's very unlikely that the software architecture / design will incorporate the required security measures and that appropriate security controls will be defined (so security design flaws are very likely)
    - Security design / controls are driven by the security requirements, and a lack of security requirements will result in an inadequate security design and missing security controls
  - It's very unlikely that focused security tests (e.g., penetration tests) will be performed
    - As one important aspect of security testing is verifying that the specified security requirements are correctly fulfilled / implemented
- → Without adequate security requirements, it's therefore very unlikely the resulting system is going to be secure
- Proper security requirements engineering has the goal to prevent the problems identified above by providing the foundation for the subsequent security activities

Security Controls are driven by Security Requirements

The purpose of security requirements is to provide the basis for security controls. So, if important security requirements are missing, then it's likely that no corresponding security controls will be integrated into the architecture / design. Or by using the long-standing credo of requirements engineering: «If you don’t know what you want, it’s hard to do it right».

Security Testing to verify Security Requirements

This implies that security requirements should be verifiable (e.g., by analyzing the code, by doing a penetration test or by assessing organizational procedures), so they should be formulated accordingly.



Threat Modeling

- Threat modeling is closely associated with security requirements engineering
  - Security requirements engineering without threat modeling is hardly going to produce good results
- The purpose of threat modeling is to identify security design flaws
  - I.e., to find conceptual security problems based on the security requirements and security controls that have already been defined
- Applied to a secure development process, threat modeling consist of the following fundamental steps:
  - Identify possible threats against the system
  - Based on these threats, identify vulnerabilities in the system design
    - Taking into account already defined security requirements or security controls
  - The identified vulnerabilities provide the basis for additional security requirements

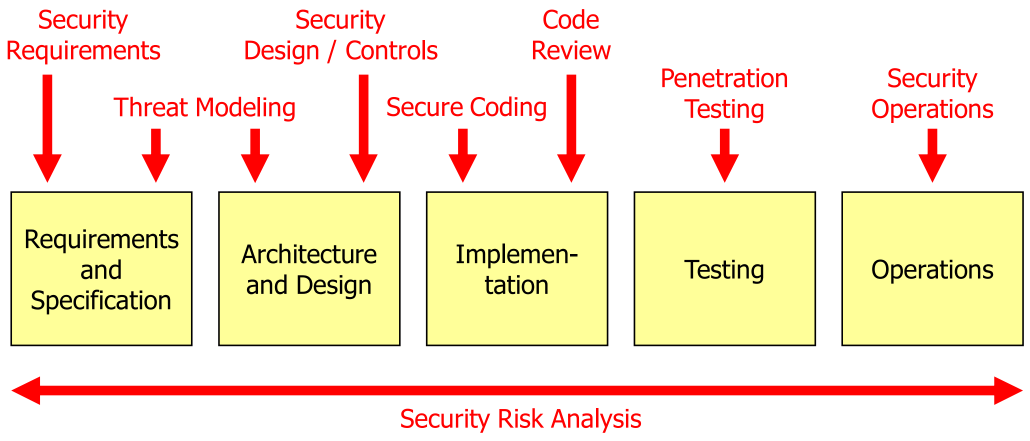
Threat Modeling as a standalone Activity

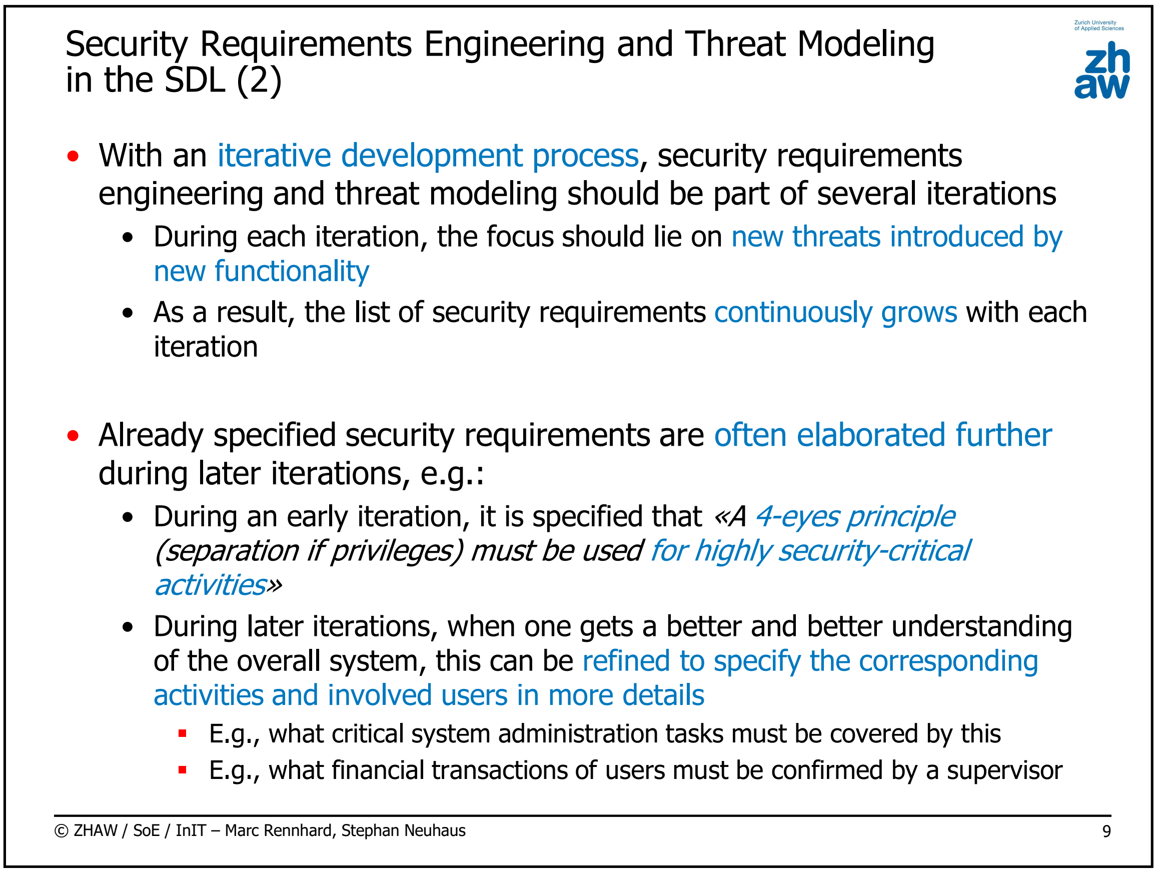
Besides as an activity during a secure development process, threat modeling can also be used as an activity on its own, e.g., to perform a conceptual security analysis of a completed system (design) as an external security expert.

But the result will be typically the same: A list of identified system vulnerabilities that are usually rated with respect to their risk / criticality.

Security Requirements Engineering and Threat Modeling in the SDL (1)

- The focus of security requirements engineering and threat modeling is during the early phases of the software development process
  - This confirms the importance of these activities as they are responsible to provide the foundation for a secure system and as they influence the following security activities





Security Requirements Engineering and Threat Modeling in the SDL (2)

- With an iterative development process, security requirements engineering and threat modeling should be part of several iterations
  - During each iteration, the focus should lie on new threats introduced by new functionality
  - As a result, the list of security requirements continuously grows with each iteration
- Already specified security requirements are often elaborated further during later iterations, e.g.:
  - During an early iteration, it is specified that *«A 4-eyes principle (separation if privileges) must be used for highly security-critical activities»*
  - During later iterations, when one gets a better and better understanding of the overall system, this can be refined to specify the corresponding activities and involved users in more details
    - E.g., what critical system administration tasks must be covered by this
    - E.g., what financial transactions of users must be confirmed by a supervisor

When to Start with Security Requirements Engineering and Threat Modeling

With security requirements engineering, one can start as soon as the first functional requirements have been defined, which is typically done in the first iteration. This most likely won't result in a complete set of requirements, but it will already provide a basis, and this basis can then already be taken into account when starting with the overall system design.

With threat modeling, one can also start very early, as some threat modeling techniques such as attack trees can be used even if one only has a first general understanding of the features of the system to be developed. However, one typically starts with threat modeling once a first high-level system design has been derived, which should include the most important actors and the main system components. But this is also when you really should start doing threat modeling, because depending on the results of threat modeling you may have to adapt the overall system design, so you should learn this early enough before investing too much time into a possibly wrong design.

Security Requirements Engineering and Threat Modeling during each Iteration

This is important to do, in particular if the extensions done in a new iteration add completely new aspects/features to a system. As an example, consider the following:

- A web application is developed and so far, functionality for «standard authenticated users» of the application has been designed and implemented that authenticate using username and password.
- In the next iteration, admin functionality should be added to the web application, which enables new functionality to do, e.g., user management such as listing all users, creating users, changing the passwords of users etc.
- If no new round of security requirements engineering / threat modeling is done, it's very likely that the functionality is simply added to the already existing web application, using the same authentication mechanism that is used for normal users.
- But when really thinking about the new functionality from a security point of view, then one most likely comes to the conclusion that the new functionality introduces new attack opportunities for attackers as they may now try to abuse the new admin functionality to get access to all user identities, to create fake accounts, to reset passwords of users etc., which definitely motivates a new round of security requirements engineering / threat modeling that considers the new functionality.
- Based on this, one likely comes to the conclusion that only using a password-based authentication is likely not secure enough for admin access, so there may be a new security requirement to use a second authentication factor for admins. Also, one may also derive that it's not a good idea to integrate the admin function directly into the existing web application, as otherwise it may be possible that normal users of the application may find a vulnerability (e.g., with respect to access control) that gives them admin access. So, a second conclusion could be to use a separate web application for admin access to better isolate it from normal users of the application.

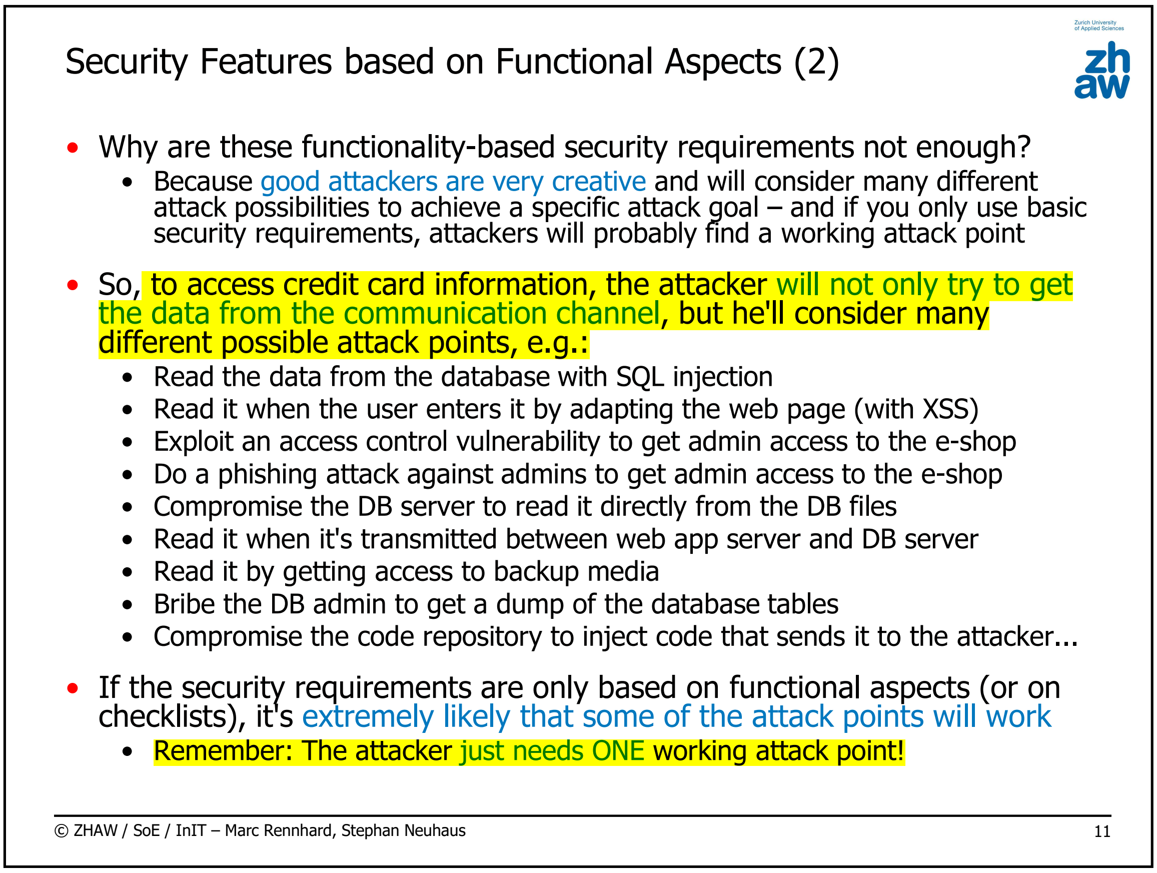
Overall, this example shows that whenever new functionality is added in an iteration, especially new functionality that introduces new actors and new functionality that's interesting for attackers, one should always also do a new round of security requirements engineering and threat modeling.

Security Features based on Functional Aspects (1)

- One approach often used in practice is to take the functional aspects of the system and define security requirements based on them
  - So instead of doing real threat modeling, one only considers the functionality of the system as a basis

- Examples in an e-shop application
  - Functionality «credit cards are transmitted» → security requirement *«Sensitive information such as payment details must only be transmitted over cryptographically secured communication channels»*
  - Functionality «there's a special area for sales personnel to view and modify orders» → security requirement *«Access control must be enforced to make sure only sales users can access the sales area»*

- In general, there's nothing wrong with doing this to get a first set of important basic security requirements – but they are usually by no means enough to achieve a good security level



Security Features based on Functional Aspects (2)

- Why are these functionality-based security requirements not enough?
  - Because good attackers are very creative and will consider many different attack possibilities to achieve a specific attack goal – and if you only use basic security requirements, attackers will probably find a working attack point
- So, to access credit card information, the attacker will not only try to get the data from the communication channel, but he'll consider many different possible attack points, e.g.:
  - Read the data from the database with SQL injection
  - Read it when the user enters it by adapting the web page (with XSS)
  - Exploit an access control vulnerability to get admin access to the e-shop
  - Do a phishing attack against admins to get admin access to the e-shop
  - Compromise the DB server to read it directly from the DB files
  - Read it when it's transmitted between web app server and DB server
  - Read it by getting access to backup media
  - Bribe the DB admin to get a dump of the database tables
  - Compromise the code repository to inject code that sends it to the attacker...
- If the security requirements are only based on functional aspects (or on checklists), it's extremely likely that some of the attack points will work
  - Remember: The attacker just needs ONE working attack point!

© ZHAW / SoE / InIT – Marc Rennhard, Stephan Neuhaus 11

Security Features based on Functional Aspects

The list above could easily be extended to 20 or more different ways to get access to the credit card information. And if the developers define security requirements only based on functional aspects of the system, it's very likely some attack points won't be prevented adequately. And remember: the attacker just needs one attack point that works to succeed!

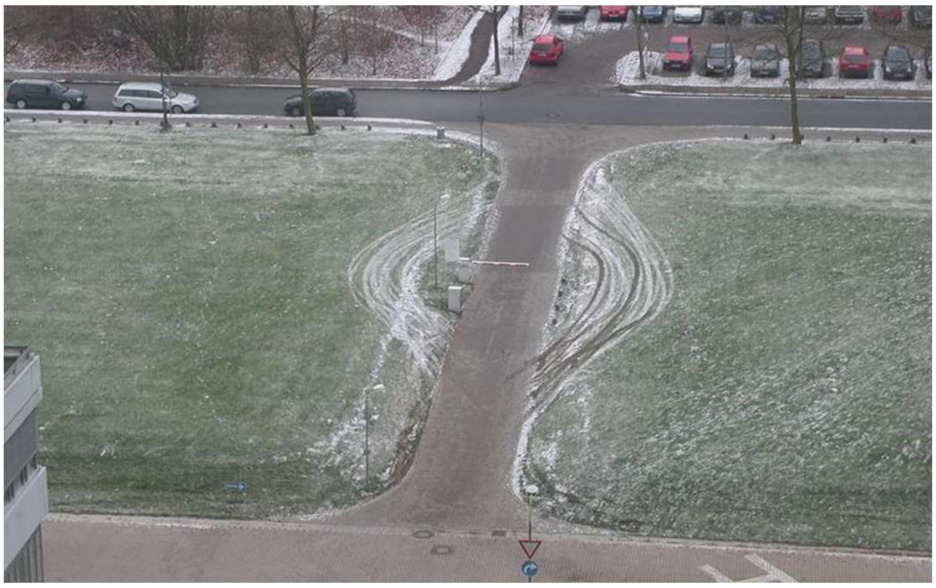
Experienced Security Engineers and Standard Applications

Of course, of the security engineer that defines the security requirements is very experienced, then he'll likely be able to derive many reasonable security requirements just because of his experience. And also, if a security engineer has defined security requirements in ten different standard web applications in the past, then he'll likely be able to define a reasonable set of security requirements for an eleventh more or less standard web application without doing a lot of threat modeling. This implies that such experienced security engineers may actually come up with a complete list to reasonably protect from the threat that an attacker wants to get the credit cards. But even such experienced security engineers should and will typically do threat modeling, because even seemingly standard web applications may have their peculiarities and in the case of novel scenarios or system/application types, it's unlikely also for experienced security engineers to come up with the right security requirements without doing threat modeling.

Two additional advantages of doing real threat modeling are that (1) it gives the team confidence that all important security requirements have been determined and (2) that it delivers a documentation of the threats and corresponding vulnerabilities (assuming documentation is seriously done) so that others can verify this, and it also provides a certain «proof» that you have taken the task of determining security requirements (hopefully) seriously.

Security Features based on Functional Aspects (3)

→ Attackers try to find ways to circumvent the security measures that are implemented, and if they find one, they will likely attack



Getting Beyond Security Features based on Functional Aspects

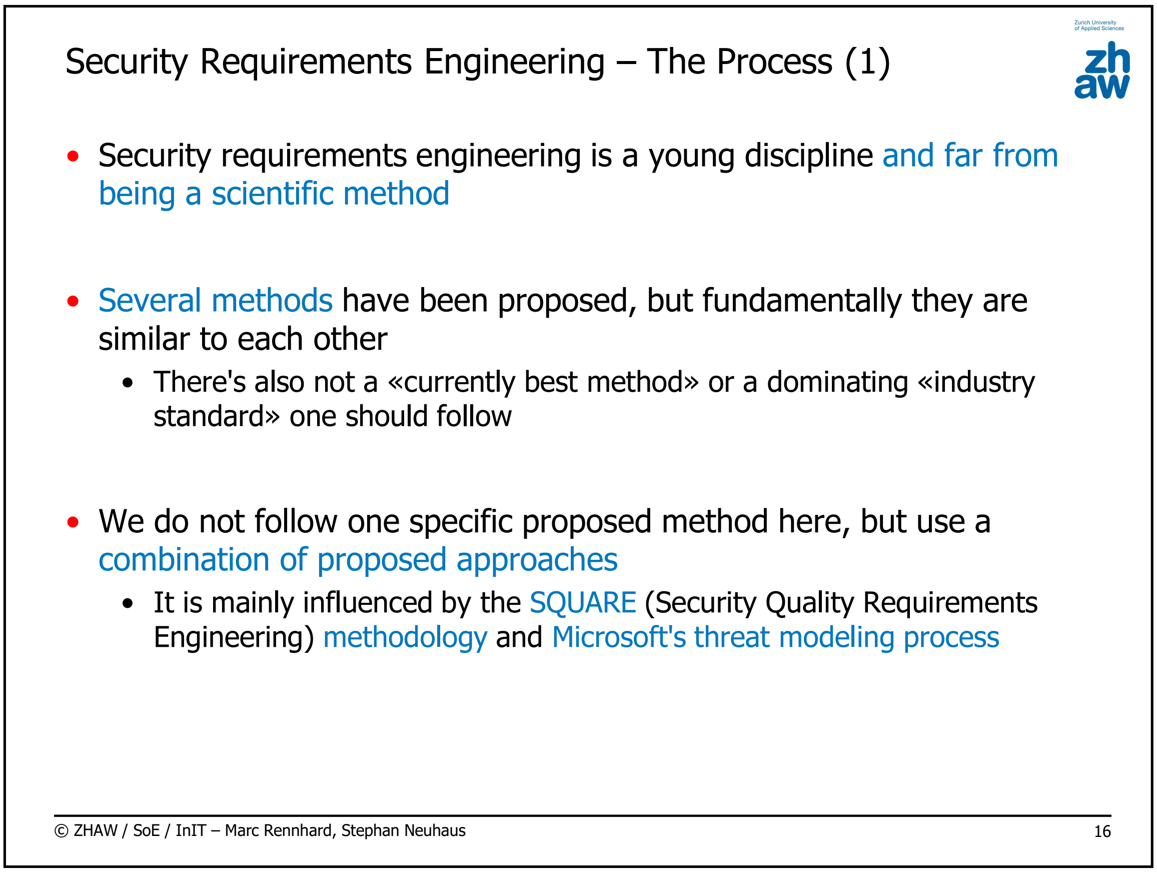
- The main goal of threat modeling is to uncover all relevant threats and attack points, not only those driven by functional aspects
- This also requires you to start thinking like an attacker, i.e., to «put on the black hat»

- During threat modeling, ask yourself: If I were attacking the system...
  - What would I want? (steal money, get administrator access, take the system offline, deface the website,...)
  - How could I accomplish this? (flood the target, elevate privileges, guess passwords, perform SQL injection,...)
    - Be creative and consider a wide range of attacks!
- → This will help you to look at software (and security) from an entirely different viewpoint
  - It takes your focus away from the functionality of the software and from already existing security measures and helps you to consider a wide range of possibilities to achieve an attack goal

«What», not «How»

- Security requirements should describe «what» has to be done, not «how» it should be done
  - How a requirement is going to fulfilled / implemented (technical details) will be determined during the Security Design / Controls security activity
  - 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, etc.)
- General rule: Security requirements should be specific enough so they can be clearly understood, but should not include the technical details how to implement the requirement
- Example of a reasonable security requirement:
  - *«The web application must employ an authorization mechanism and authorization must be checked on every single request»*
  - This is clearly understandable but the specific security mechanism to implement this is not determined by the requirement

Security Requirements Engineering – The Process



Security Requirements Engineering – The Process (1)

- Security requirements engineering is a young discipline and far from being a scientific method
- Several methods have been proposed, but fundamentally they are similar to each other
  - There's also not a «currently best method» or a dominating «industry standard» one should follow
- We do not follow one specific proposed method here, but use a combination of proposed approaches
  - It is mainly influenced by the SQUARE (Security Quality Requirements Engineering) methodology and Microsoft's threat modeling process

Security Quality Requirements Engineering (SQUARE) Methodology

https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=484884

Microsoft's Threat Modeling Approach

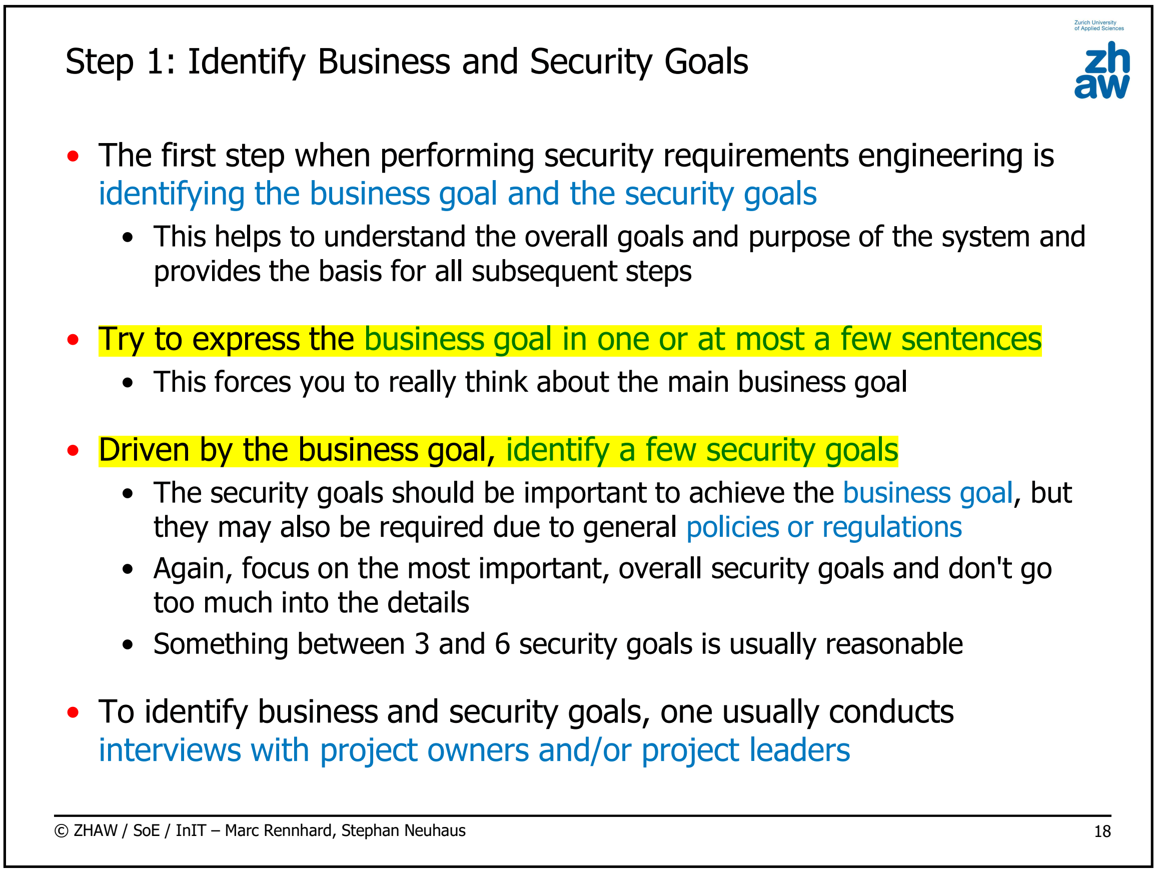
It's part of the Microsoft Security Development Lifecycle process: *https://www.microsoft.com/en-us/SDL/about/default.aspx*

Security Requirements Engineering – The Process (2)

The security requirements engineering process consists of 5 steps:

1. Identify the business and security goals of the system
2. Collect information about the system so you get a good understanding about its purpose and functions
3. Decompose the system to understand its internal workings and as a basis for the following step
4. Identify threats that are relevant for the system and rate the risk of these threats → if the risk of a threat is too high, then a vulnerability has been identified
5. Mitigate the threats where necessary by proposing adequate security requirements that bring down the risks to acceptable levels

$\}$ Threat modeling activities

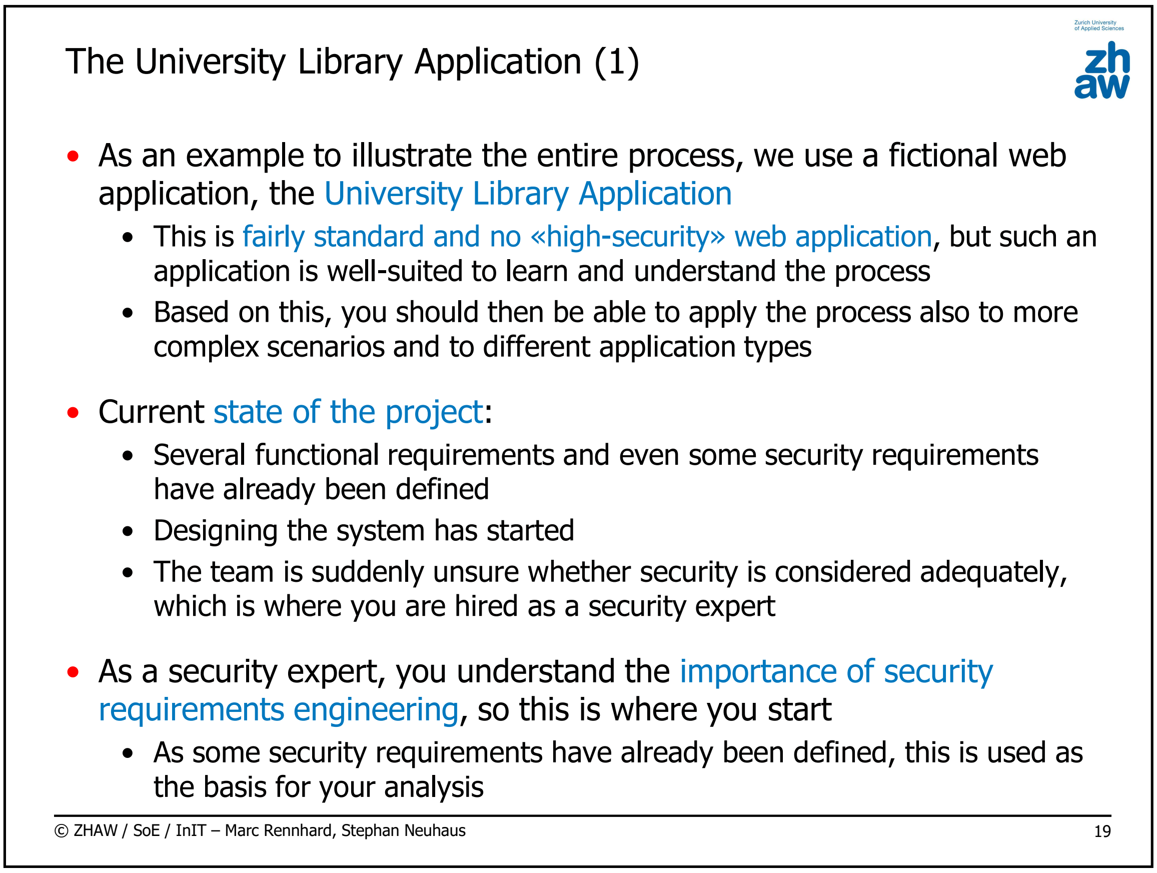


Step 1: Identify Business and Security Goals

- The first step when performing security requirements engineering is identifying the business goal and the security goals
  - This helps to understand the overall goals and purpose of the system and provides the basis for all subsequent steps
- Try to express the business goal in one or at most a few sentences
  - This forces you to really think about the main business goal
- Driven by the business goal, identify a few security goals
  - The security goals should be important to achieve the business goal, but they may also be required due to general policies or regulations
  - Again, focus on the most important, overall security goals and don't go too much into the details
  - Something between 3 and 6 security goals is usually reasonable
- To identify business and security goals, one usually conducts interviews with project owners and/or project leaders

Determining Business and Security Goals

Of course, it may be the software development team that is performing security requirements engineering / threat modeling is well aware of business and security goals. However, it may also be that special internal teams are consulted for this phase or an external company is given the task to perform threat modeling. In these cases, the persons performing the process may not be aware of these goals, so it’s important to consult the correct resources (e.g., project owners and project leaders) to get the information. In any case, it’s important that all personnel involved in threat modeling has a clear understanding of the business and security goals.



The University Library Application (1)

- As an example to illustrate the entire process, we use a fictional web application, the University Library Application
  - This is fairly standard and no «high-security» web application, but such an application is well-suited to learn and understand the process
  - Based on this, you should then be able to apply the process also to more complex scenarios and to different application types
- Current state of the project:
  - Several functional requirements and even some security requirements have already been defined
  - Designing the system has started
  - The team is suddenly unsure whether security is considered adequately, which is where you are hired as a security expert
- As a security expert, you understand the importance of security requirements engineering, so this is where you start
  - As some security requirements have already been defined, this is used as the basis for your analysis

The University Library Application

It is based on an example by OWASP (*https://www.owasp.org*) and has been modified for this course. The original example from OWASP is no longer online.

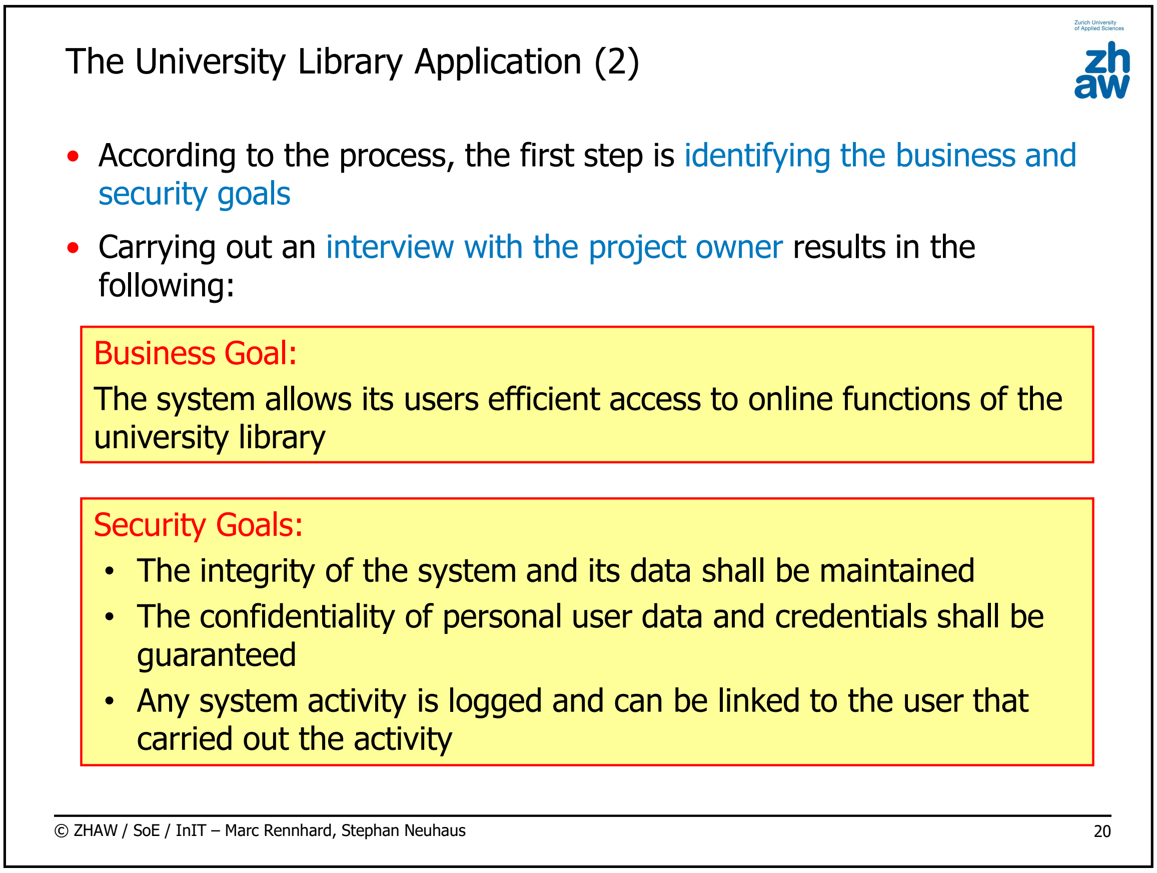
This seems all so simple - can it get more challenging?

At the end of this chapter, in particular if you are already quite experienced in the context of web application security, you may think that determining the security requirements and, based on them, making the corresponding security design / controls decisions is quite easy and obvious considering what was discussed in the chapter. Indeed, in the case of a fairly standard web application that is not specifically security-relevant as used as an example in this chapter, this is really not very complicated as the typical threats / attacks and corresponding security controls are well established, and assuming you already have a good understanding of web application security, you may even be able to come up with a good security design without going through the entire security requirements engineering and threat modeling process (although that is still recommended, as mentioned before). But the purpose of this chapter is to present the security requirements and threat modeling *process* so you get a good understanding about this process in general and that you can apply it on your own. And the best way to learn this process is by using an application type you already know (here: web applications) and by using a relatively simple example, so the focus can truly be set on the actual process and not on technical complexities of the underlying system.

When applying the process in practice, however, it can of course get much more complicated than presented in this chapter, in particular if the system you are developing is «less standard» and/or more security-critical. Assume, e.g., you are developing an industrial control systems where the attacks and attack goals may be quite different. Or think about IoT devices or devices for home automation, where the threats will again differ and where computing power is often sparse and standard security solutions may not be easily applicable. And also in the web application domain, there are scenarios with much higher security requirements than in the current example, for instance, if financial transactions are involved or if highly-sensitive personal information (e.g., medical data about patients) or sensitive business information is managed and processed. In all these «not so standard» or more security-critical cases, security requirements engineering, threat modeling and coming up with good security design / controls are very challenging and require a lot of know how, understanding, and creativity about realistic threats / attacks (in the corresponding application domain) and reasonable corresponding security solutions. And often, standard and well-known «off the shelf» security solutions are not enough in such scenarios and instead, one has to come up with clever security engineering solutions to achieve the desired security level.

But discussing such advanced systems goes beyond the scope of this module, as the primary goal of the module is that you understand the secure development lifecycle and that you can apply it in the context of typical scenarios that you already understand from a technical point of view. However, to get some insight into systems with high security requirements and that use clever and non-standard approaches to solve some security issues, you can have a look at the following documents (the first two describe projects where a significant portion of the overall security design was done in the context of research and development projects by the Institute of Applied Information Technology (InIT, https://www.zhaw.ch/init) at ZHAW)):

- Rennhard, Marc; Tschannen, Michael; Christen, Tobias. SecureSafe : a highly secure online data safe. https://digitalcollection.zhaw.ch/handle/11475/1746
- Lapagna, Kevin; Zollinger, Moritz; Rennhard, Marc; Strobel, Hans; Derché, Cyrille. Dokspot : securely linking healthcare products with online instructions. https://digitalcollection.zhaw.ch/handle/11475/8897
- vsftpd (Very Secure FTPD): https://security.appspot.com/vsftpd.html



The University Library Application (2)

- According to the process, the first step is identifying the business and security goals
- Carrying out an interview with the project owner results in the following:

Business Goal:

The system allows its users efficient access to online functions of the university library

Security Goals:

- The integrity of the system and its data shall be maintained
- The confidentiality of personal user data and credentials shall be guaranteed
- Any system activity is logged and can be linked to the user that carried out the activity

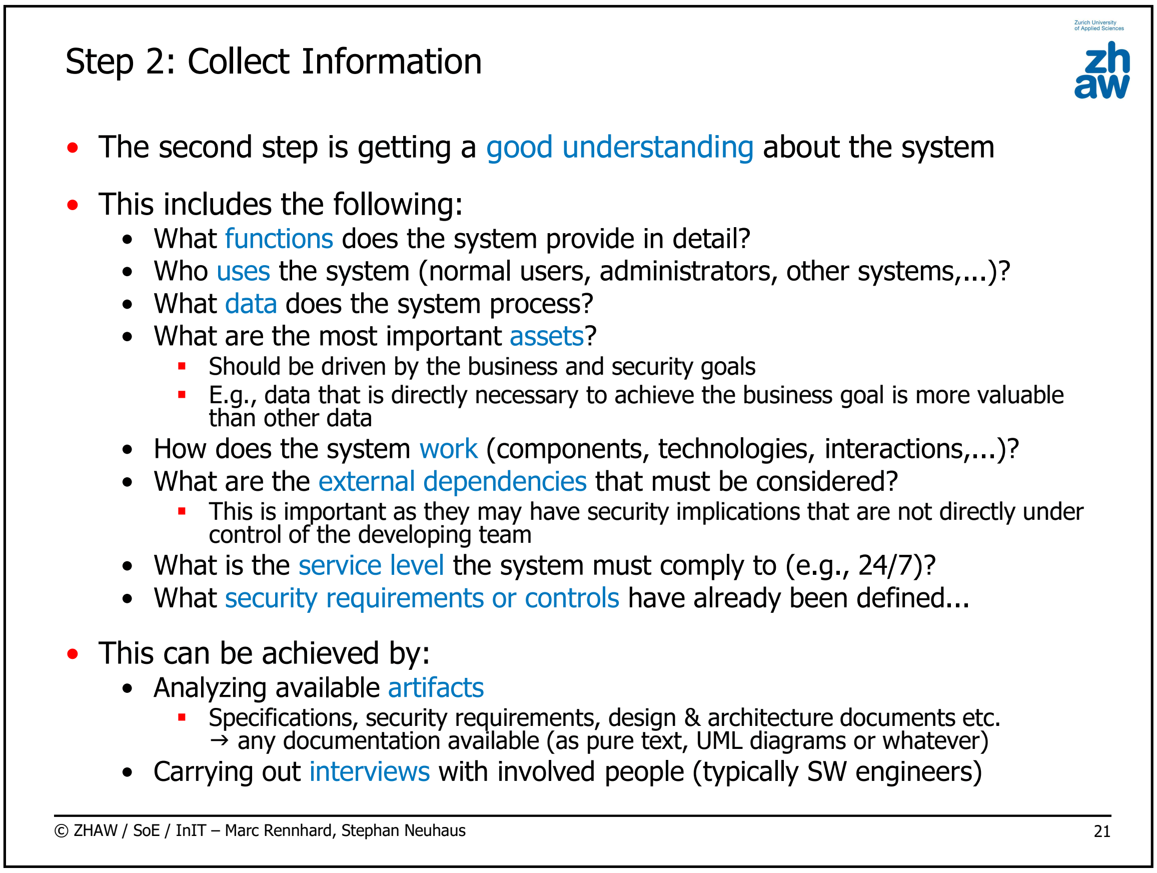
Security Goals

- The first security goal is clearly driven by the business goal: Only a correctly functioning system (system integrity) and correct data (data integrity) will allow efficient online access.
- The second security goal is mainly driven by university policies: Personal user data shall be kept private and credentials must not be leaked, but it's also needed to support the business goal as leaked credentials of librarians allow abusing the system (e.g., create a huge number of user accounts until the system becomes non-operational).
- The third goal was added to detect potential abuse of the system, e.g., by a malicious librarian or an attacker that managed to steal the credentials of a user, which helps to identify the cause of a security breach and which therefore helps to resume normal operation after an incident, which in turn supports the business goal.

Don't go into the Details

Neither the business nor security goals should go into the details. So don't start to enumerate important features in the business goal, such as «library users must be able to reserve books», as otherwise, the business goal quickly becomes a list of functional requirements. That's not needed as «efficient access to online functions» implicitly contains the important functions. Note also that the business goal does not have to include all functions of the system. For instance, the system may also provide statistical information to analyze library usage, but if this is not considered an important and central function, then the business goal should not include it and in addition, no corresponding security goals would be defined (assuming that statistical information is not security-critical).

Likewise, don't included detail in the security goals. So a security goal «send passwords only over protected network links» is way too specific because then, further similar goals would have to be added in the context of password handling and as a result of this, the security goals would become a list of security requirements. Instead, use high-level security goals such as the ones above, where, e.g., the second goal covers all security mechanisms needed to protect passwords. Note also that one should only include security goals that are truly relevant to achieve the business goal. Here, for instance, there's no security goal related to availability as apparently, there are no specific availability requirements (as will be described during step 2).


Step 2: Collect Information

- The second step is getting a good understanding about the system
- This includes the following:
  - What functions does the system provide in detail?
  - Who uses the system (normal users, administrators, other systems,...)?
  - What data does the system process?
  - What are the most important assets?
    - Should be driven by the business and security goals
    - E.g., data that is directly necessary to achieve the business goal is more valuable than other data
  - How does the system work (components, technologies, interactions,...)?
  - What are the external dependencies that must be considered?
    - This is important as they may have security implications that are not directly under control of the developing team
  - What is the service level the system must comply to (e.g., 24/7)?
  - What security requirements or controls have already been defined...
- This can be achieved by:
  - Analyzing available artifacts
    - Specifications, security requirements, design & architecture documents etc. → any documentation available (as pure text, UML diagrams or whatever)
  - Carrying out interviews with involved people (typically SW engineers)

More Information allows a more detailed Analysis

The more information that is available, the more thorough the subsequent steps can be performed.

- In early phases, there often exists only a general description and some functional requirements.
  - As a result of this, only basic security requirements can be derived (but with limited time effort).
- As the system architecture and design advances, more information will be available allowing for a more detailed analysis.
  - Which likely will produce further and more specific security requirements.

Security requirements engineering is therefore usually applied repeatedly while the overall system design is developed, which fits well with an iterative software development process.

The University Library Application (3)

Collecting information on our example application reveals the following:

- General information of the application:
  - The university library application is a web application to provide librarians and library users (students and staff) with online services
    - As this is the first version, the functionality will be limited
    - The web application uses a database application to store the data, which runs on a separate server
  - There will be three types of main users of the application:
    - Students, staff, librarians
  - Offered functionality:
    - Students and staff can log in, search for books (physical books and e-books), reserve physical books (to pick them up later), and download e-books
    - Staff members can additionally request new books
    - Librarians can log in, search for books, maintain the list of available books, view and delete book reservations, and add and delete users
  - Students and staff have individual login credentials, librarians share one account
  - High availability is not needed, downtimes during the weekend or even partial downtimes during workdays are not critical

The University Library Application (4)

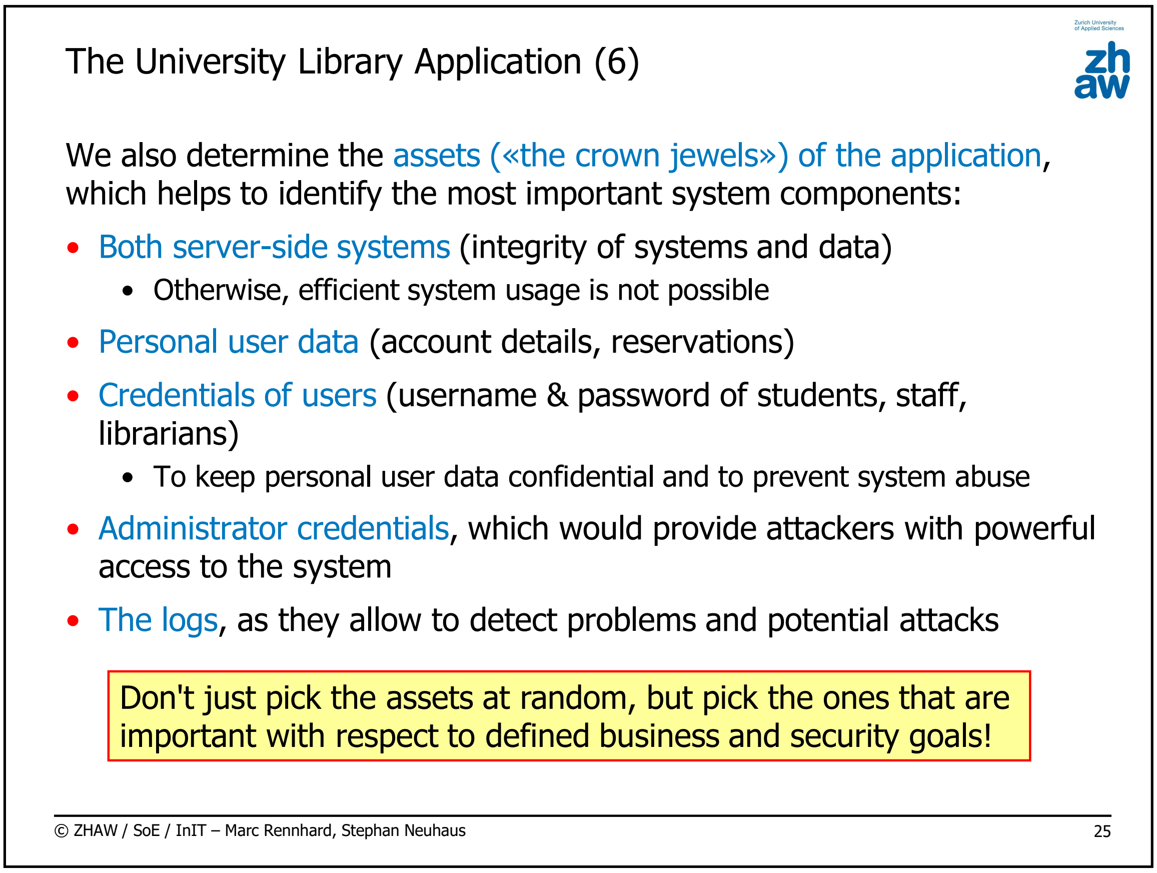
There are also some external dependencies for the application:

- The university library application runs on standard university servers for web applications
  - Linux, Java, Jakarta EE, Payara (we assume that Jakarta EE is used to develop web applications)
  - Hardened according to the university's server hardening standards, which are considered adequate for typical university services
    - Includes, e.g., the latest security patches for OS and software components
- The database server will be MySQL on a Linux server, also hardened according to university standards
- The server components are physically located in the same network as servers of other typical applications of the university, access to the network is protected by a packet filtering firewall

The University Library Application (5)

Some security requirements have also been defined and documented:

- R1: Students, staff and librarians can access the web application server only via a cryptographically protected channel (HTTPS / TCP port 443) and the web application server is authenticated using a certificate from a trusted certification authority
- R2: Direct access to the web and DB servers for configuration and log monitoring is only allowed to administrators and only via SSH (TCP port 22)
- R3: Administrators must have at least security clearance «medium» (employed ≥ 5 years, regular background checks)
- R4: Only TCP ports 22 / 443 are reachable from the outside
- R5: All performed activities are logged locally on the web application server and linked to the user, all queries are logged on the database server, and the SSH daemons log successful logins and login failures
- R6: Students and staff have individual login credentials (username & password), librarians share one account
- R7: Standard servers of the university must be used (Linux, Java, Jakarta EE, Payara, MySQL), which are hardened according to the university's hardening standards



The University Library Application (6)

We also determine the assets («the crown jewels») of the application, which helps to identify the most important system components:

- Both server-side systems (integrity of systems and data)
  - Otherwise, efficient system usage is not possible
- Personal user data (account details, reservations)
- Credentials of users (username & password of students, staff, librarians)
  - To keep personal user data confidential and to prevent system abuse
- Administrator credentials, which would provide attackers with powerful access to the system
- The logs, as they allow to detect problems and potential attacks

Don't just pick the assets at random, but pick the ones that are important with respect to defined business and security goals!

Assets are driven by Business and Security Goals

The assets listed above are driven by the business and security goals we defined earlier:

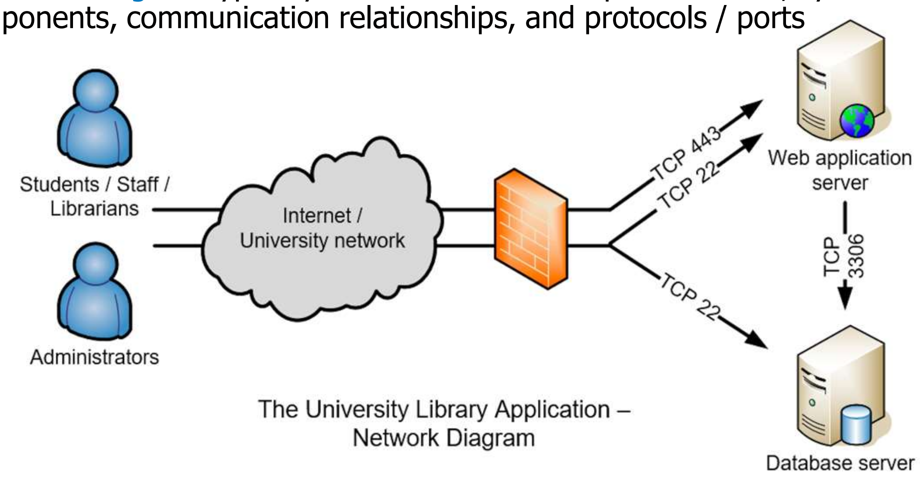
- The business goal ("The system allows its users efficient online access to the university library") and first security goal ("The integrity of the system and its data shall be maintained") imply that the server-side system components (integrity of systems and data) are important assets.
- The second security goal ("The confidentiality of personal user data and credentials shall be guaranteed") implies that personal user data and user and librarian credentials are important assets.
- Administrator credentials are important assets, as owning them allows to threaten the business goal and all three security goals, because with the administrator credentials, access to all application components and all data in the database is likely possible.
- Finally, the logs are assets as they are needed to fulfill the third security goal ("Any system activity is logged and can be linked to a user").

Step 3: Decompose the System

- The next step is to decompose the system into its components
  - To understand its internal workings and as a basis for the following step
- There are multiple ways to do so, e.g., by drawing a network diagram or a data flow diagram
- It's important to include all relevant information
  - Different users and main system components
  - Interactions between users and system components and between system components
  - Interactions with external systems or applications
  - Also make sure to include all assets (identified in step 2)
- Choose a reasonable level of detail that allows to efficiently determine threats and vulnerabilities during step 4
  - Often, it's not necessary to include very many details as this will only make the following step much more complex without producing significantly better results

Network Diagram

- A network diagram typically includes the most important users, system components, communication relationships, and protocols / ports



The University Library Application – Network Diagram

- Usually, not directly used to identify threats, but well-suited to get a good overview of the entire system and also to make sure all involved persons have the same and correct understanding of the system
- If threat modeling is done by an external team, it's a good idea to show the network diagram to the development team for verification

Network Diagram

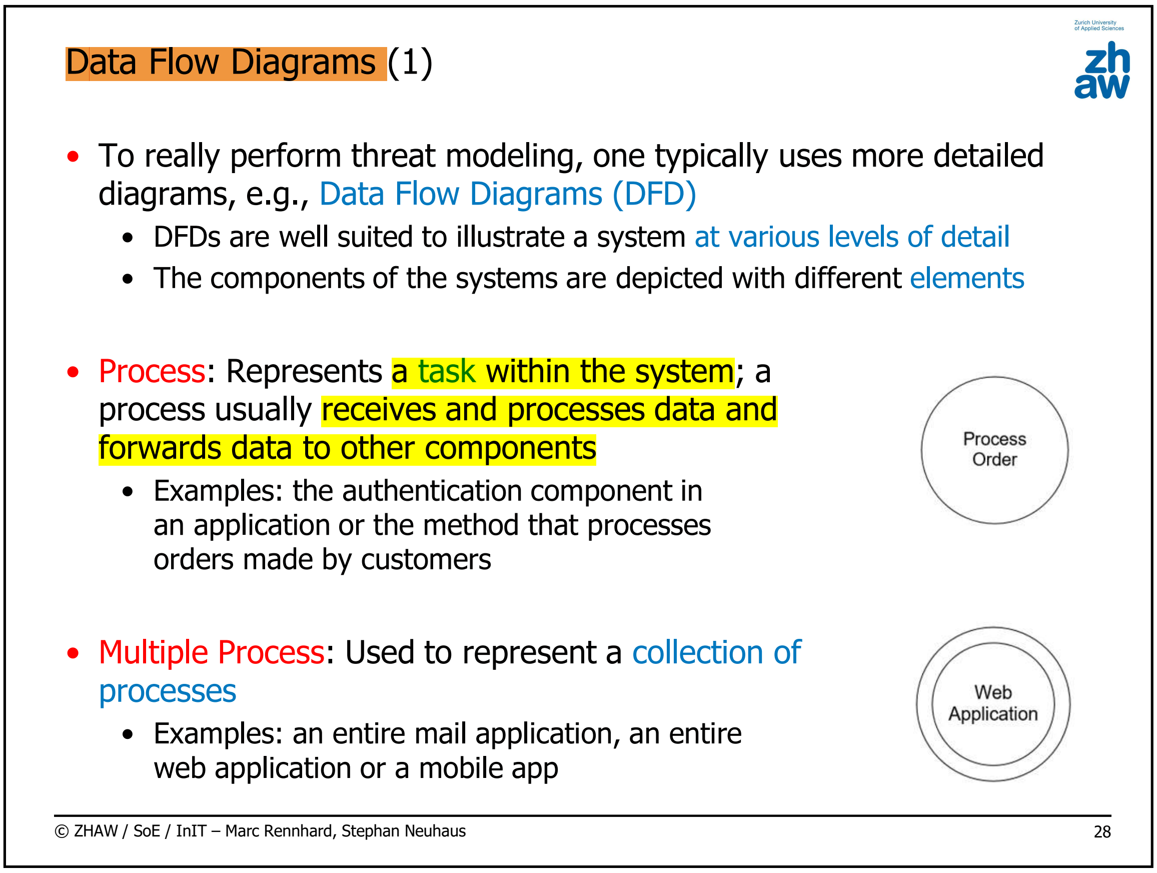
It may of course be well possible that such diagrams are available as system documentation and have been collected during step 1. In general. It's always a good start to begin with a network diagram as it shows – although on a high-level – the most important components and how they interact. This diagram can then be used as a basis to draw more detailed data flow diagrams (see next slide).

Network Diagrams for Threat Modeling

Basically, a network diagram could be used to think about threats and security requirements. For instance, in the diagram illustrated above, one could derive the following:

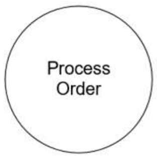
- The web application server is the entry point into the application, so it must authenticate users before allowing access to protected areas
- Credentials are sent across the network, which could be sniffed by attackers, so they must be protected (as is already the case with HTTPS / SSH)
- All servers in the network can access the database server, so the database application must authenticate the web application before granting access

But in general, network diagrams usually only serve as a first overview of the overall system and of the main system components. To really perform threat modeling, one uses more detailed diagrams, e.g., data flow diagrams.

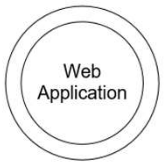


Data Flow Diagrams (1)

- To really perform threat modeling, one typically uses more detailed diagrams, e.g., Data Flow Diagrams (DFD)
  - DFDs are well suited to illustrate a system at various levels of detail
  - The components of the systems are depicted with different elements
- Process: Represents a task within the system; a process usually receives and processes data and forwards data to other components
  - Examples: the authentication component in an application or the method that processes orders made by customers



- Multiple Process: Used to represent a collection of processes
  - Examples: an entire mail application, an entire web application or a mobile app

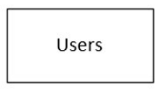


Process vs. Multiple Process

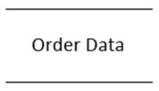
In practice, it's often not so clear if «something» is a process or a multiple process. But in the end, it doesn't matter too much in practice where you draw the line between the two, because when applying STRIDE on the elements, processes and multiple processes are treated equally

Data Flow Diagrams (2)

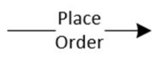
- External Entity: Used to represent any entity outside the system that interacts with the system
  - Examples: users, administrators, but also external components used by / interacting with the system (e.g., a 3rd party web service)

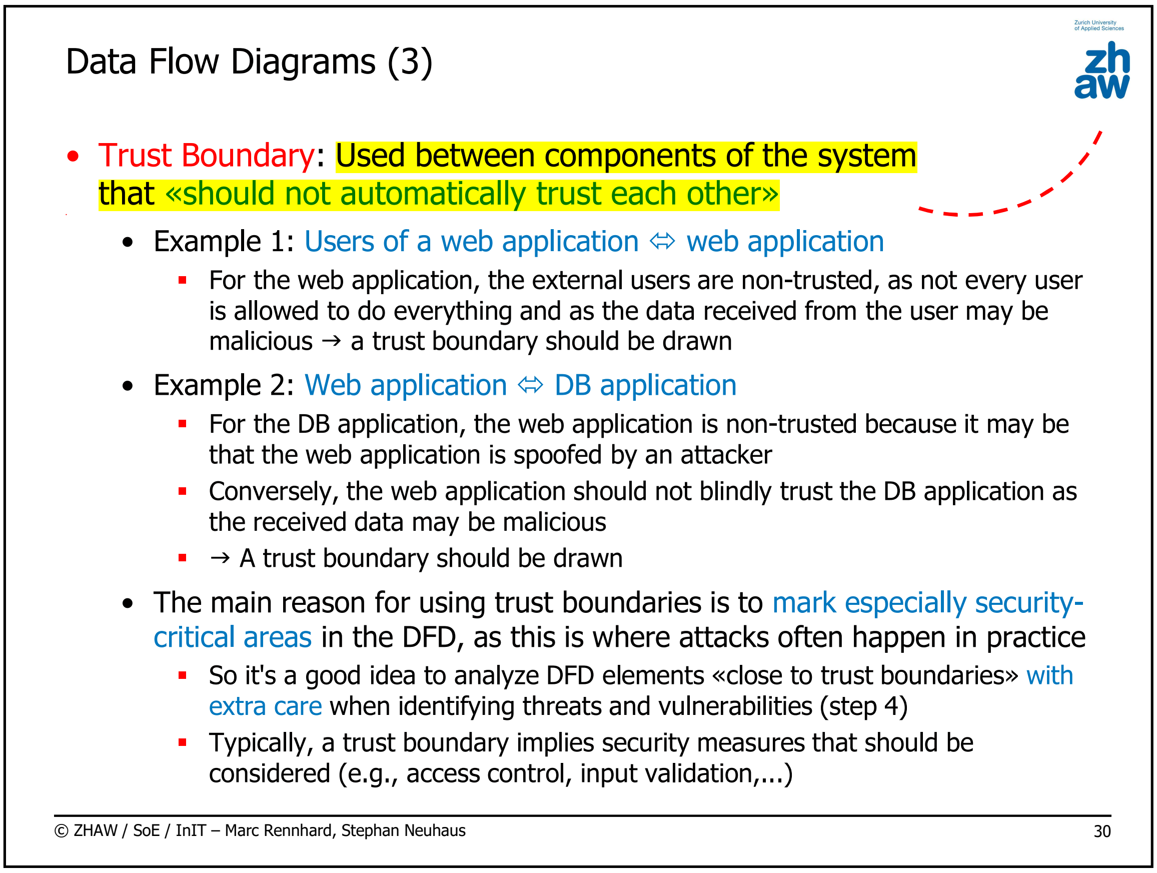


- Data Store: Used to represent locations where data is stored
  - Examples: database files, configuration files, log files



- Data Flow: Represents interactions within the system; the arrow indicates the direction of the data flow
  - Examples: data sent over a network or data exchanged between components of a mail server





Data Flow Diagrams (3)

- Trust Boundary: Used between components of the system that «should not automatically trust each other»
  - Example 1: Users of a web application ⇔ web application
    - For the web application, the external users are non-trusted, as not every user is allowed to do everything and as the data received from the user may be malicious → a trust boundary should be drawn
  - Example 2: Web application ⇔ DB application
    - For the DB application, the web application is non-trusted because it may be that the web application is spoofed by an attacker
    - Conversely, the web application should not blindly trust the DB application as the received data may be malicious
    - → A trust boundary should be drawn
  - The main reason for using trust boundaries is to mark especially security-critical areas in the DFD, as this is where attacks often happen in practice
    - So it's a good idea to analyze DFD elements «close to trust boundaries» with extra care when identifying threats and vulnerabilities (step 4)
    - Typically, a trust boundary implies security measures that should be considered (e.g., access control, input validation,...)

© ZHAW / SoE / InIT – Marc Rennhard, Stephan Neuhaus 30

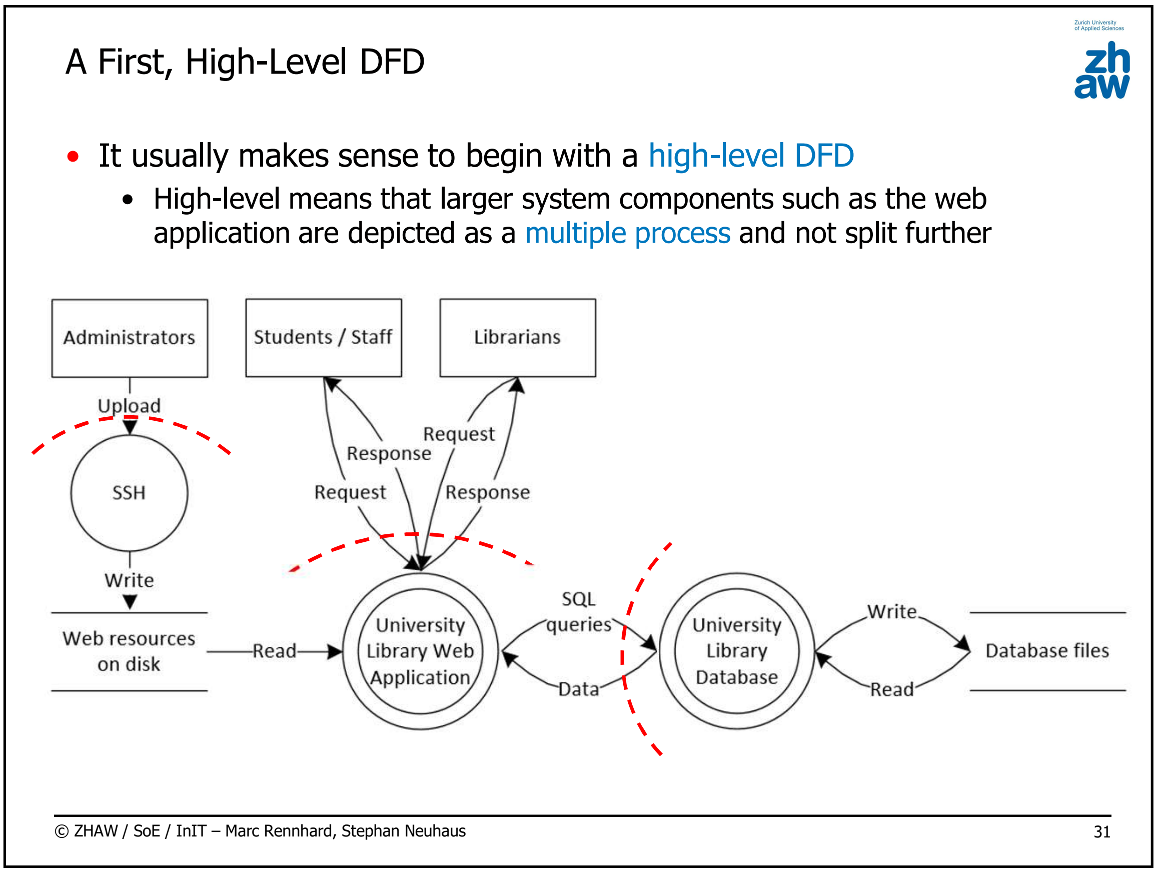
Trust Boundary

They usually delineate data moving from “low to high trust” areas and vice versa. For instance, an anonymous user creates a request to a higher-privileged process (e.g., a web application) for consumption. As the web application most likely has powerful (high) privileges (e.g., perform a shopping transaction in an e-shop system), it is necessary to closely inspect this trust boundary to make sure that anonymous users can only do what they should be allowed to do. But the “return channel” must also be considered to make sure the web application does not leak sensitive (high trust) information to anonymous users, e.g., showing the shopping history of a specific user to an anonymous user. External entry points are therefore typical candidates for trust boundaries, except in the case where the system serves purely informational purposes without distinguishing between different categories of users.

Trust Boundaries also exist between systems. For instance, a web application may access a database, but most likely, the web application should not be able to do everything (e.g., changing the DB schema) and there may be auditing data in the DB that should not be leaked. So again, a trust boundary makes sense to “keep this in mind” during the subsequent analysis.

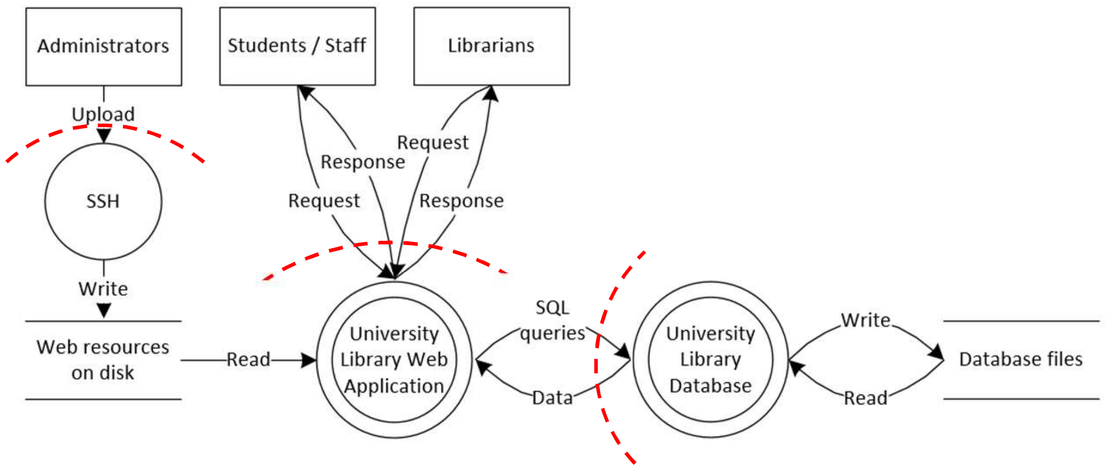
Depending on the level of detail used in a DFD, trust boundaries may also be used between processes running on the same system. For instance, if an application is split – for security reasons – into multiple independent running local processes where each process has only exactly the rights needed to perform its task (the Postfix mal server is a good example of this), then using such boundaries between the individual processes also makes sense, because it may be that one of these processes has been compromised by an attacker, who then tries to abuse a higher-privileged process to increase his own privileges (as maybe this higher-privileged process runs with root rights (for whatever reason), which may be abused by sending specifically crafted data to it).

In general, trust boundaries usually imply that some sort of access control and input validation must take place and that measures to prevent information leakage are provided. And in general, DFD elements «close to trust boundaries» should be analyzed with extra care as they correspond to “entry points from the outside” into the application, where attackers likely will invest a lot of effort to find vulnerabilities.



A First, High-Level DFD

- It usually makes sense to begin with a high-level DFD
  - High-level means that larger system components such as the web application are depicted as a multiple process and not split further



Details left out to Keep it Simple

Note that the DFD above has already left out some details, to keep everything reasonably presentable on the slide above (but these details should be included in a «real» DFD used for the library application, as they are associated with assets. This includes administrative access to the database via SSH, the SSH logs and also the logs on the web application and database servers. In addition, the SSH process uses credentials stored on the device during authentication, so this should be included as well in a complete DFD.

Assets

Note that the DFD above includes all assets identified during step 2 (except those associated with the elements that were left out for simplicity):

- Both server-side systems (integrity of systems and data) – they are obviously present in the DFD.
- Personal user data (account details, reservations) – that's stored in the database files.
- Credentials of users (username & password of students, staff, librarians) – that's also stored in the database files.
- Administrator credentials, which would provide attackers with powerful access to the system – that would be stored in files that were left out for simplicity in this drawing.
- The logs, as they allow identify problems quickly and resume normal operation – they were also left out for simplicity.

Trust Boundaries

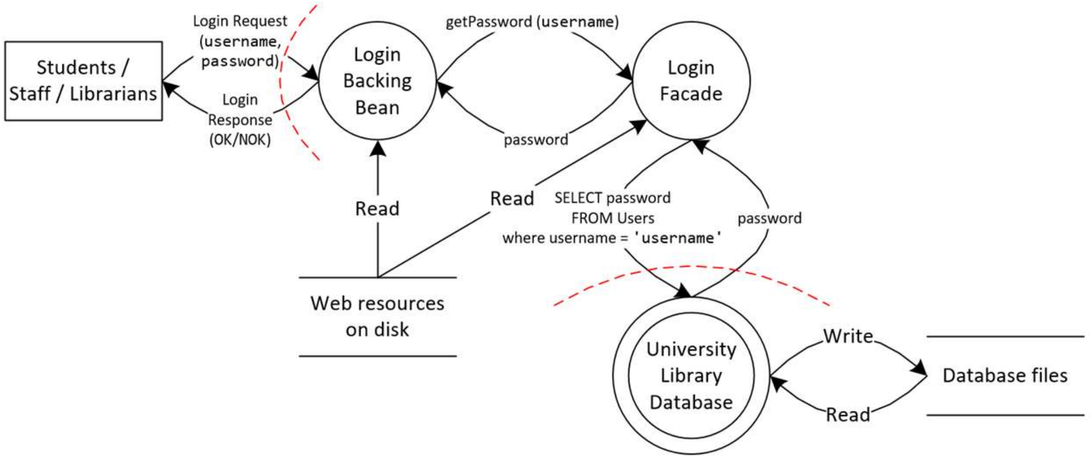
There are obvious trust boundaries between the users and the system components, illustrated by the top two boundaries. But there's another one between the web server and the database server, as from the point of view of the database application, the received data is non-trusted as not every system or application is allowed to access the database. And also, the web application should not blindly trust the data it gets from the database, as it may be malicious (e.g., it may contain JavaScript code in the context of a stored XSS attack).

Where are the Users' Computers / Browsers / Tools?

Maybe you are wondering why there are no devices of the users included, e.g., the browsers used by students & staff. Well, they are not explicitly included, but they are implicitly part of the external entities. So, the external entity *Students / Staff* also includes the browser to access the web application. And as we are starting with a high-level DFD, merging the browsers with the users makes sense. Note also that as attacks that target the browsers of the users are likely not very realistic given the value of the target and the assumed attackers and their goals (details follow later in this chapter). However, in other cases or if one wants to explicitly include the browsers, they of course can be included, and they could be included as additional external entities (e.g., in the case of the students, as their computers are probably out of control for the university) or they could be included as a multiple process (e.g., with staff users and librarians, assuming their browsers can be controlled at least to a certain degree, e.g., by providing these users administrated computers). Note that even if the browsers are not drawn in the DFD, they may still be considered during step 4 of the process. For instance, an attacker could try to spoof a librarian by compromising his browser and – while the librarian is logged in – send his own requests over the authenticated session with the web application. So, attacks that involve a compromised browser can still be considered, but mainly from the viewpoint of the effect they have on the attacked application (here a spoofing attack against the library application) and less from the viewpoint of the initial attack that compromised the browser in the first place.

Drilling Down

- One can also drill down deeper
  - Assume we want to analyze the login process in detail
  - In this case, it's reasonable to draw a DFD that focusses on the login process and that no longer includes the web application as a multiple process
  - Therefore, we include the login backing bean and the login facade (assuming JSF is used), as these are the two core components involved during login



Students / Staff / Librarians
Login Request (username, password)
Login Response (OK/NOK)
Login Backing Bean
getPassword (username)
password
Login Facade
Read
Read
Web resources on disk
SELECT password FROM Users where username = 'username'
password
University Library Database
Write
Read
Database files

Drilling Down

DFDs can be drawn at every possible level of details. Use the level that is most suited for your threat modeling effort but in general, starting at a high-level and making sure that the important trust boundaries (and data flows across these boundaries) are present is a good idea. Once threat modeling has been done at this level, you can always drill down further if necessary (especially if the system includes highly security-critical functions).

The main advantage of using a more detailed DFD is that it may help you to identify threats and attacks (see step 4 of the process) that you may overlook when just looking at the high-level DFD. For instance, looking at the Users (Students/Staff/Librarians), the Login Request data flow and the Login Backing Bean process in the DFD above, then you are more likely to focus an attacks specifically in the context of the login request and its processing by the backing bean. One can then «easily» derive a possible attack to, e.g., get access as a librarian or as a staff user by omitting the password parameter from the login request. Depending on how the backing bean is implemented (e.g., if it fails to an insecure state if the password parameter is missing), this may grant the attacker authenticated access as a staff user. When only using the high-level DFD, then this «processing of username & password» is less obvious and therefore, corresponding threats may not be considered.

As another example that shows that a more detailed DFD can help to uncover important threats, consider a web application that uses a secret key or public key pair that is stored in a file in the file system to encrypt/decrypt sensitive data, e.g., to protect credit card information before it is stored in the database. If the web application is drawn as a multiple process («high-level»), it may happen that the corresponding file is not included as a separate data store, because it may be considered as an integral part of the web application (and therefore of the multiple process). As a result of this, corresponding threats against the data store (e.g., read or tamper with the keys) may not be considered as it is not visible in the DFD. However, if the web application is drawn in more detail to specifically analyze the security-critical credit card payment/handling process, then it's very likely that the data store will be included (because the mindset of the analyst drawing the DFD will be explicitly on this process, so it's much more likely he will consider and include all important details) and therefore, you'll also likely think about specific threats in the context of this data store. And based on this, you'll then likely come up with security requirements that specifically address such threats (e.g., use suitable file access permissions and make sure that only highly trusted administrators have the rights to access the corresponding file).

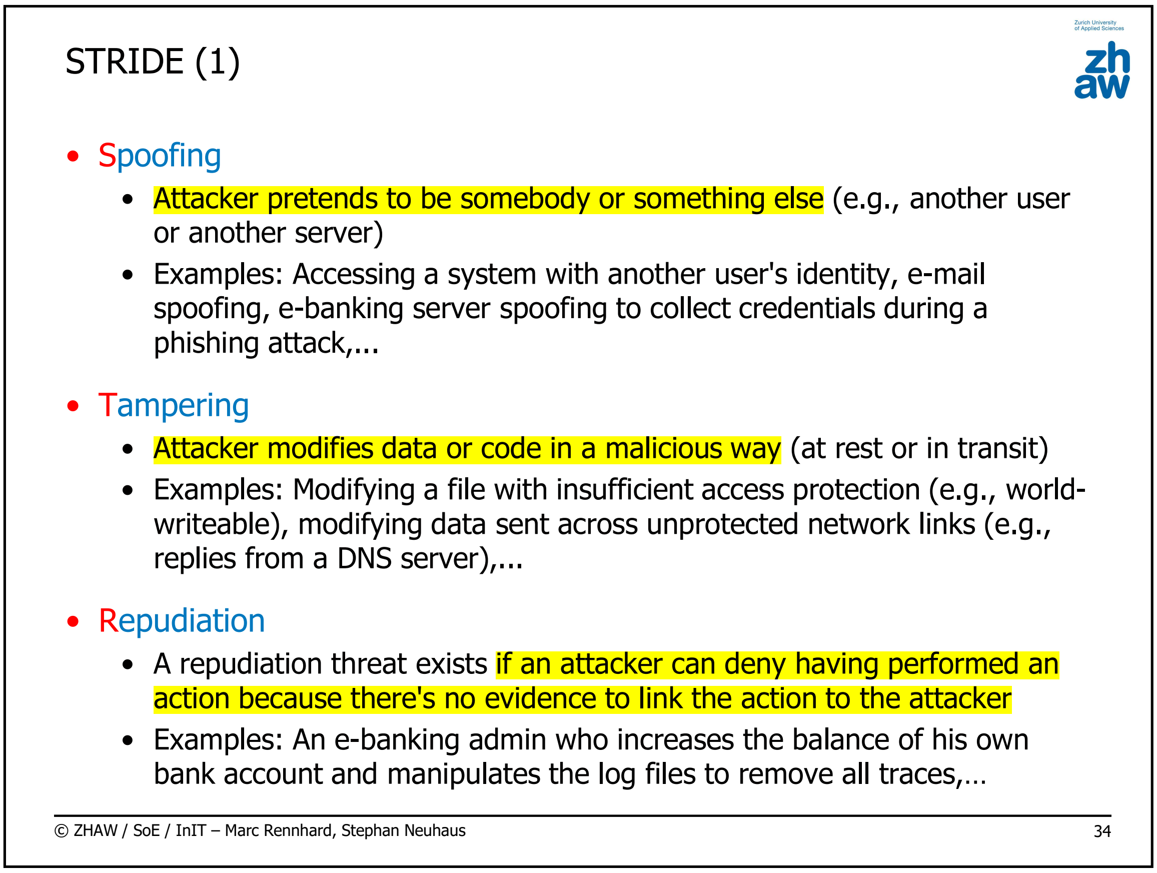
However, the general experience is that a high-level DFD as used on the previous slide (assuming that it also includes the elements that were omitted for space restrictions, such as, e.g., the data stores for the log data) is usually «good enough» and well suited to analyze a system with «typical security requirements», as more detailed DFDs also mean more work during the actual analysis and often do not really provide better results (in particular if the security engineers doing the analysis are experienced and make sure that the important components, such as the data store with the key pair in the example described above, are truly included in the high-level DFD). In some cases, however, a more detailed view is certainly appropriate, e.g., to analyze very security-critical processes (e.g., the entire payment process in an e-banking system), where it may be reasonable to include all involved components in detail in the DFD.

Step 4: Identify Threats, Risks, and Vulnerabilities

- The next step is to identify threats that are relevant for the system under analysis, to rate the risk of these threats, and to identify vulnerabilities if the risk of a threat is too high

- A good way to identify threats is by using the Microsoft STRIDE approach

- STRIDE basically is a checklist of six attack / threat categories:
  - Spoofing
  - Tampering
  - Repudiation
  - Information disclosure
  - Denial of service
  - Elevation of privilege



STRIDE (1)

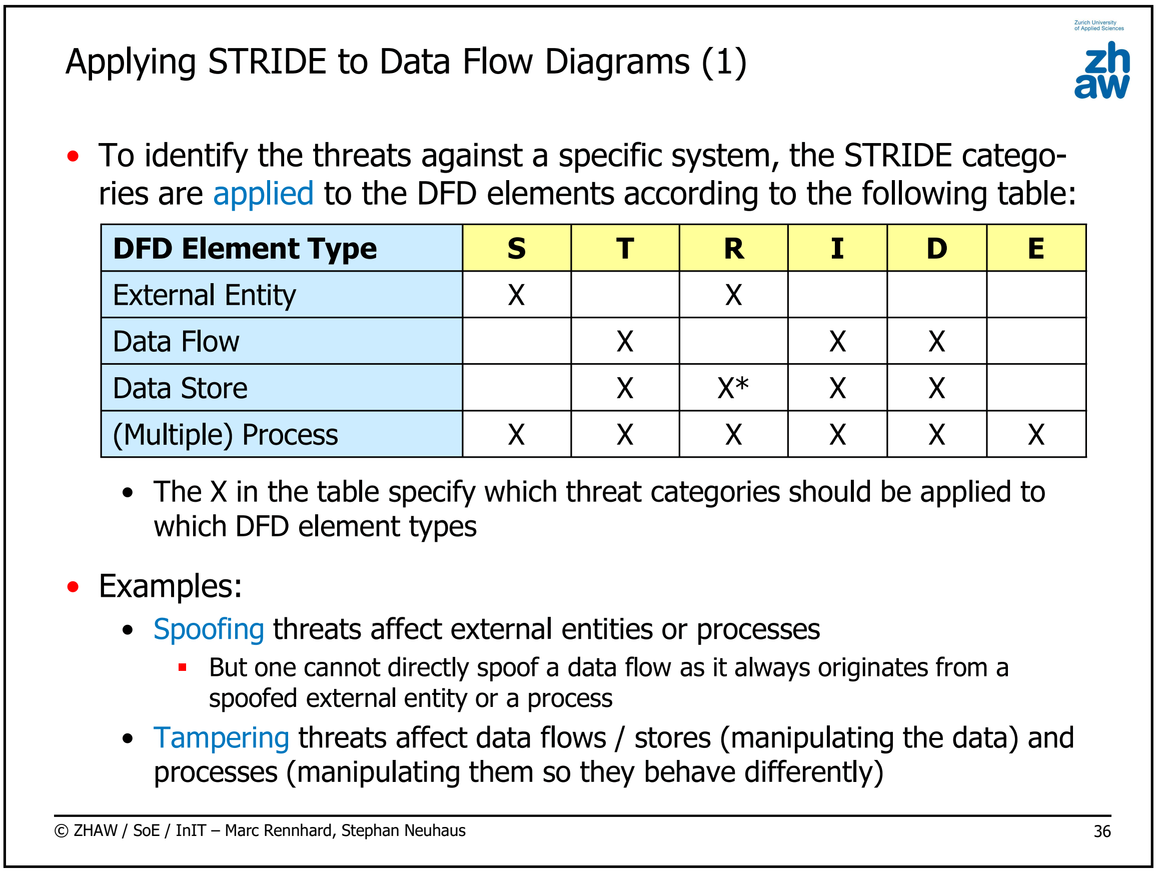
- Spoofing
  - Attacker pretends to be somebody or something else (e.g., another user or another server)
  - Examples: Accessing a system with another user's identity, e-mail spoofing, e-banking server spoofing to collect credentials during a phishing attack,...
- Tampering
  - Attacker modifies data or code in a malicious way (at rest or in transit)
  - Examples: Modifying a file with insufficient access protection (e.g., world-writeable), modifying data sent across unprotected network links (e.g., replies from a DNS server),...
- Repudiation
  - A repudiation threat exists if an attacker can deny having performed an action because there's no evidence to link the action to the attacker
  - Examples: An e-banking admin who increases the balance of his own bank account and manipulates the log files to remove all traces,…

STRIDE

For more on STRIDE, see: *https://msdn.microsoft.com/en-us/library/ms954176.aspx*

STRIDE (2)

- Information disclosure
  - Attacker gets access to information he's not authorized to read
  - Examples: Reading files with plaintext passwords, reading credit card data sent across unprotected network links, reading data from the database using SQL injection,…
- Denial of service
  - Attacker prevents legitimate users from accessing the system
  - Examples: Crashing an application or a system (triggering an exception, causing a buffer overflow), flooding a system with large amounts of network traffic,...
- Elevation of privilege
  - Attacker gets more privileges than he is entitled to
  - Examples: Exploiting a vulnerable program that runs with administrator rights, circumventing a poor access control mechanism,…



Applying STRIDE to Data Flow Diagrams (1)

- To identify the threats against a specific system, the STRIDE categories are applied to the DFD elements according to the following table:

DFD Element Type	S	T	R	I	D	E
External Entity	X		X			
Data Flow		X		X	X	
Data Store		X	X*	X	X	
(Multiple) Process	X	X	X	X	X	X

  - The X in the table specify which threat categories should be applied to which DFD element types
- Examples:
  - Spoofing threats affect external entities or processes
    - But one cannot directly spoof a data flow as it always originates from a spoofed external entity or a process
  - Tampering threats affect data flows / stores (manipulating the data) and processes (manipulating them so they behave differently)

© ZHAW / SoE / InIT – Marc Rennhard, Stephan Neuhaus 36

* If the data store contains logging or audit data, repudiation is a potential threat because by manipulating the data, the attacker could cover his tracks.

Trust Boundaries
Trust boundaries are not considered as they primarily serve to identify interesting areas in a DFD.

Relation between STRIDE threats and DFD elements

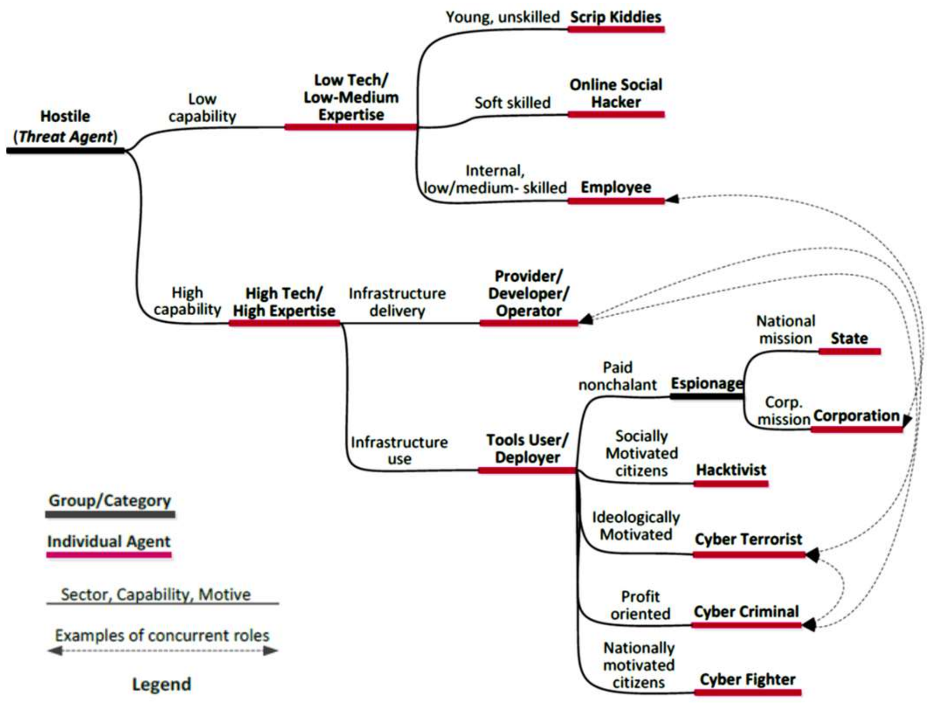
- Spoofing threats affect external entities (e.g., an attacker masquerading as a legitimate user) or processes (e.g., an attacker pretending to be an e-banking server).
- Tampering threats affect data flows (manipulating data in transit), data stores (manipulating stored data) and processes (manipulating a process such that it performs differently from being indented, e.g., by sending credentials to the attacker after a successful user login). Note that Tampering external entities is of course possible (e.g., manipulating a 3rd party system), but it's out of scope as "nothing can be done" about it. So either you have to trust that system "to be secure enough" or you shouldn't use it.
- Repudiation threats can stem from external entities (users that don't have to authenticate to perform critical activities or users that share credentials) and processes (typically compromised ones) that can access other systems without authentication to perform critical activities or that no longer perform logging correctly). If audit or log data is stored in a data store, it also gets affected by reputation threats as manipulating the stored data may allow an attacker to cover his tracks.
- Information disclosure threats affect data flows (getting access to sensitive data while in transit), data stores (accessing sensitive data in storage) and processes (non-validated input parameters that are processed).
- Denial of service threats affect data flows (legitimate data flows are prevented due to flooding by an attacker or by an insider who physically interrupts the network link), data stores (exhaust a log file by an attacker that triggers many log entries) and processes (an attacker sending a malformed packet to a server so it crashes).
- Elevation of privilege threats affect processes (e.g., an error in an authentication routine or a missing access control check).

Other Attacks against External Entities
Maybe you are asking why tampering or information disclosure are not relevant with external entities. To illustrate this, assume a system uses a backup service provided by a 3rd party. This would be drawn as an external entity. And of course, an attacker can potentially attack this service to get your backups, so this would be information disclosure. Still, such attacks are not considered with external entities because this is out of your own control. You should consider spoofing (e.g., make sure you send the backups to the right backup service) and repudiation (e.g., make sure you get some confirmation message from the service after each successful backup so the service cannot claim later that he never got anything), but you cannot do anything against the other attacks against the service. So you either trust the service (and for this, e.g., you may ask the service for an internal security analysis they have done) and then you should probably use it or you don't trust it and then don't use it. But it does not really make sense for you to consider the attack types TIDE in the context of this external service.

Threat Agents

- Before applying STRIDE to the DFD, one should think about potential attackers (threat agents) and what their attack goals are
  - This is a very important step and helps us to consider realistic attacks during the remainder of the process
- The basis to do this are the assets of the system
  - Depending on the value of these assets, different types of attackers will be attracted
  - If the value of the assets is low, one likely has to deal with script kiddies; if the value is high, one could face organized cyber criminals
- Depending on the identified potential attackers, one can make reasonable assumptions about the threats / attacks that may likely happen
  - Obviously, organized cyber criminals are capable of executing more advanced attacks than script kiddies
  - A powerful expected attacker means we require higher security standards

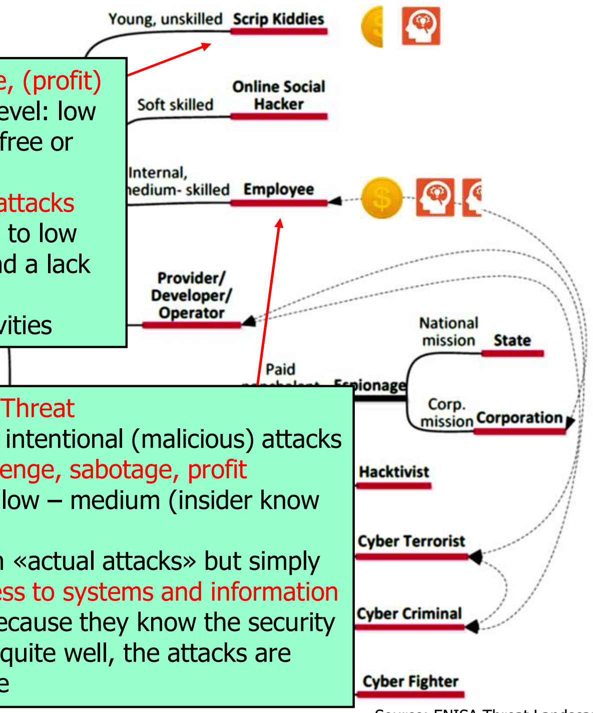
Threat Agents according to ENISA (1)



Source: ENISA Threat Landscape Report, 2015

ENISA = European Union Agency for Network and Information Security

Threat Agents according to ENISA (2)

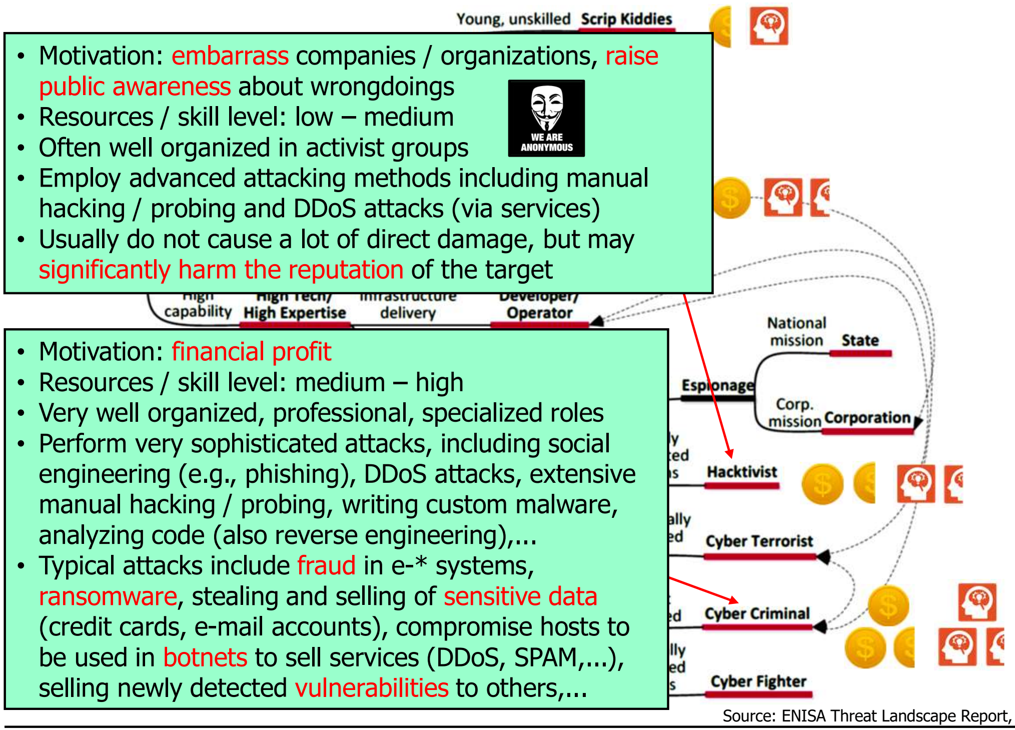


- Motivation: thrill, fun, fame, (profit)
- Resources: very low, skill level: low
- Often work alone and use free or purchased hacking tools
- Despite being low-skilled, attacks can have great impact due to low threshold of self-control and a lack of understanding of the consequences of their activities

- Also referred to as Insider Threat
- Includes unintentional and intentional (malicious) attacks
- Motivation (malicious): revenge, sabotage, profit
- Resources: low, skill level: low – medium (insider know how)
- Often, they do not perform «actual attacks» but simply abuse their legitimate access to systems and information
- Because of this and also because they know the security protections and processes quite well, the attacks are sometimes hard to mitigate

Source: ENISA Threat Landscape Report, 2015

Threat Agents according to ENISA (3)



- Motivation: embarrass companies / organizations, raise public awareness about wrongdoings
- Resources / skill level: low – medium
- Often well organized in activist groups
- Employ advanced attacking methods including manual hacking / probing and DDoS attacks (via services)
- Usually do not cause a lot of direct damage, but may significantly harm the reputation of the target

- Motivation: financial profit
- Resources / skill level: medium – high
- Very well organized, professional, specialized roles
- Perform very sophisticated attacks, including social engineering (e.g., phishing), DDoS attacks, extensive manual hacking / probing, writing custom malware, analyzing code (also reverse engineering),...
- Typical attacks include fraud in e-* systems, ransomware, stealing and selling of sensitive data (credit cards, e-mail accounts), compromise hosts to be used in botnets to sell services (DDoS, SPAM,...), selling newly detected vulnerabilities to others,...

Source: ENISA Threat Landscape Report, 2015

Threat Agents according to ENISA (4)

- Motivation: intelligence, competitive advantage
- Resources / skill level: very high (nation state: almost unlimited resources...)
- Targets include state secrets, military secrets, company secrets, military and critical infrastructures,...
- Will do virtually anything to achieve the attack goals, including buying highly expensive exploits on the black market, bringing their own people into target organization, «buying» insiders, cooperate with manufacturers to install backdoors in SW and HW,...
- Very hard to defend against a committed attacker

Key difference between nation states and cyber criminals:

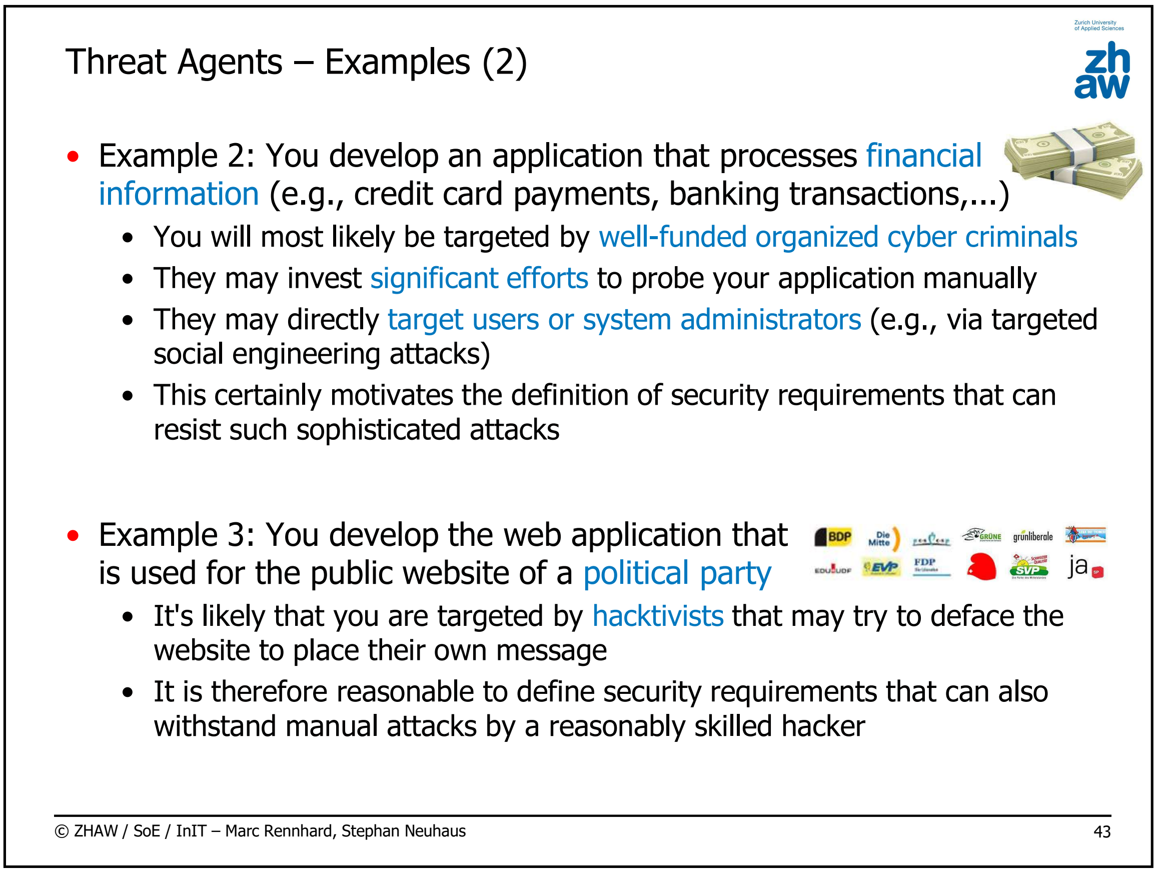
- Nation states have a specific target in mind and will go for it, at all costs
  - So even if the target has very high security standards, the chances are high that the attacker eventually succeeds
- Cyber criminals «just» want to achieve financial profit; in the end, they don't care, where they'll get it
  - So if a potential target has a high security standard, it's likely it won't be attacked successfully if attacker can achieve their goals by attacking other, less well protected targets

Source: ENISA Threat Landscape Report, 2015

Threat Agents – Examples (1)

- Example 1: You develop a web application for your local sports club

  - You most likely won't be the target of determined nation states or cyber criminals that expect huge profits
  - Still, script kiddies may try to deface the web site for fun and cyber criminals may be interested in using your system as a bot or to host illegal content
  - So it's likely your system will be probed for some easily exploitable vulnerabilities (low hanging fruit), most likely using automated tools
  - It is therefore reasonable to define security requirements that help resisting this kind of attack, but not much more (using a checklist with typical web application security measures may be good enough here)



Threat Agents – Examples (2)

- Example 2: You develop an application that processes financial information (e.g., credit card payments, banking transactions,...)
  - You will most likely be targeted by well-funded organized cyber criminals
  - They may invest significant efforts to probe your application manually
  - They may directly target users or system administrators (e.g., via targeted social engineering attacks)
  - This certainly motivates the definition of security requirements that can resist such sophisticated attacks
- Example 3: You develop the web application that is used for the public website of a political party
  - It's likely that you are targeted by hacktivists that may try to deface the website to place their own message
  - It is therefore reasonable to define security requirements that can also withstand manual attacks by a reasonably skilled hacker

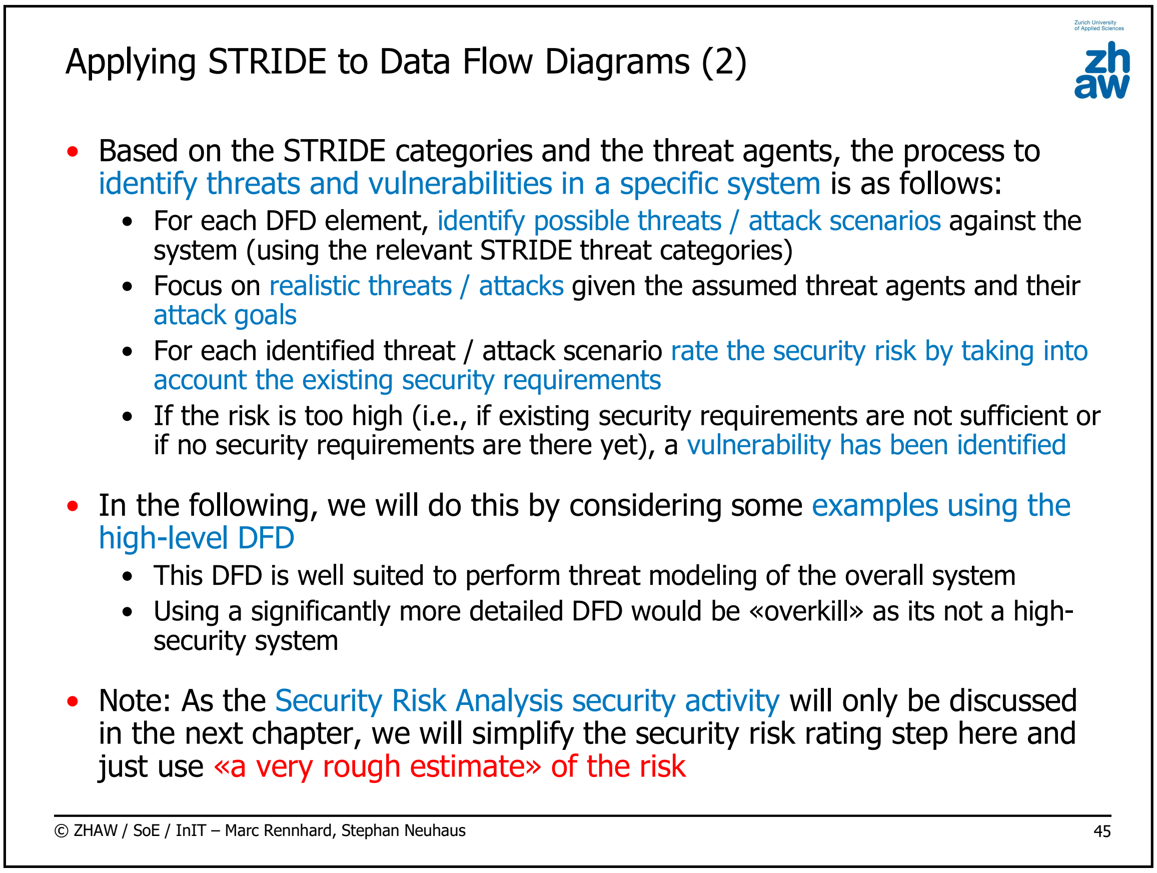
Security Measures for these Examples

Different threat agents definitely lead to different security requirements:

- In the 1st and 3rd example, using, e.g., two-factor authentication would probably be too much, as threats such as phishing attacks are not realistic given the assumed attackers. But in the 2nd example, this is certainly needed.
- Likewise, given the threat agents, security monitoring would certainly be too much in the 1st example, would definitely be needed in the 2nd example (e.g., to ideally detect attacks in their early/preparation stages), and would maybe be reasonable (at least do good logging to maybe learn the root cause of a successful attack) in the 3rd example.
- As another example, encrypting backups would only be needed in then 2nd example, as only there it is likely an attacker (e.g., an insider that is bribed or threatened by cyber criminals) may try to get access to sensitive data my stealing it from backup media.
- With respect to security testing, one should at least use automated tools in the 1st example, do extensive manual pen testing in the 2nd example regularly, and maybe do manual pen testing in the 3rd example after every significant application change.

The University Library Application – Threat Agents Exercise

Consider the university library application: Identify realistic attackers and their corresponding goals

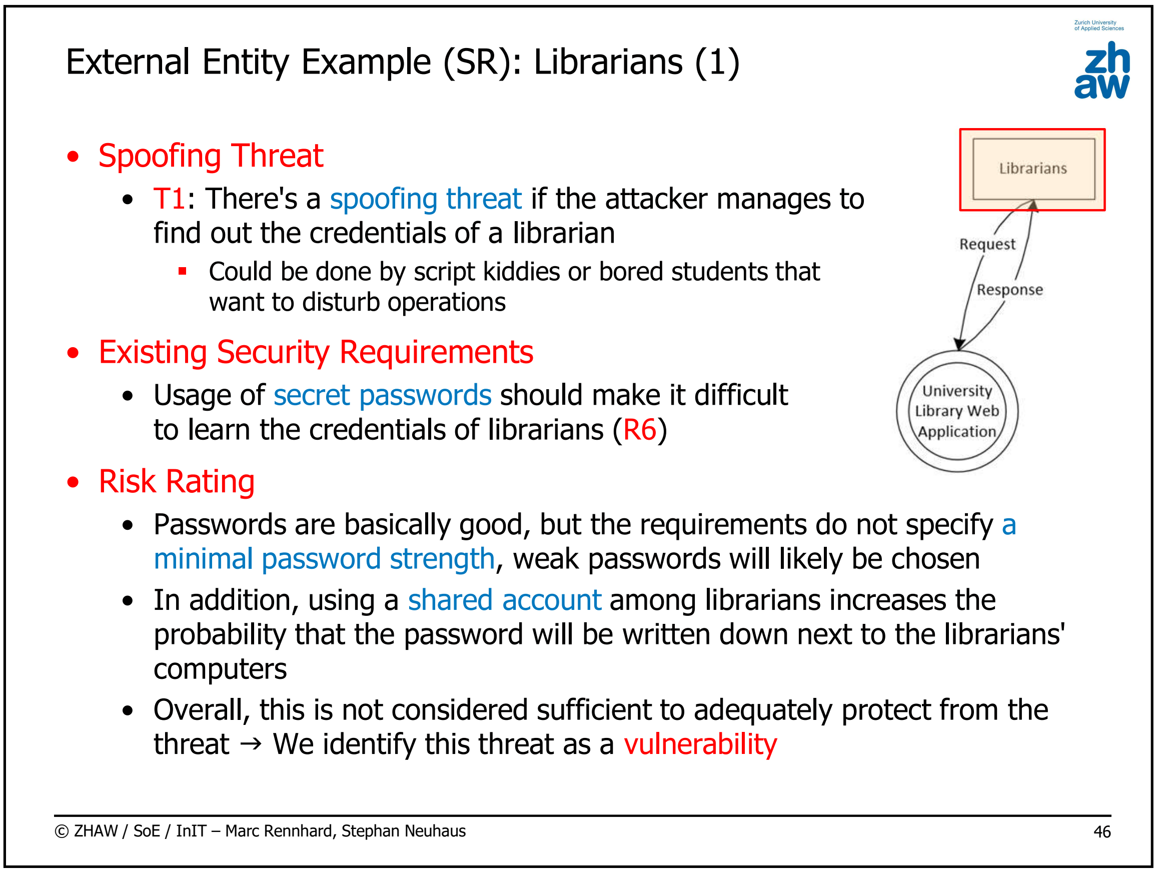


Applying STRIDE to Data Flow Diagrams (2)

- Based on the STRIDE categories and the threat agents, the process to identify threats and vulnerabilities in a specific system is as follows:
  - For each DFD element, identify possible threats / attack scenarios against the system (using the relevant STRIDE threat categories)
  - Focus on realistic threats / attacks given the assumed threat agents and their attack goals
  - For each identified threat / attack scenario rate the security risk by taking into account the existing security requirements
  - If the risk is too high (i.e., if existing security requirements are not sufficient or if no security requirements are there yet), a vulnerability has been identified
- In the following, we will do this by considering some examples using the high-level DFD
  - This DFD is well suited to perform threat modeling of the overall system
  - Using a significantly more detailed DFD would be «overkill» as its not a high-security system
- Note: As the Security Risk Analysis security activity will only be discussed in the next chapter, we will simplify the security risk rating step here and just use «a very rough estimate» of the risk

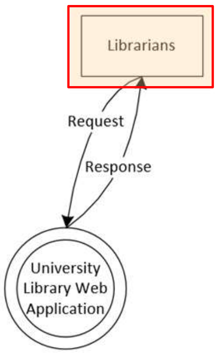
Increase the Efficiency of this Step

To increase the efficiency of this step, put a stronger focus on threats that target the assets of the system and on DFD elements «close to trust boundaries».



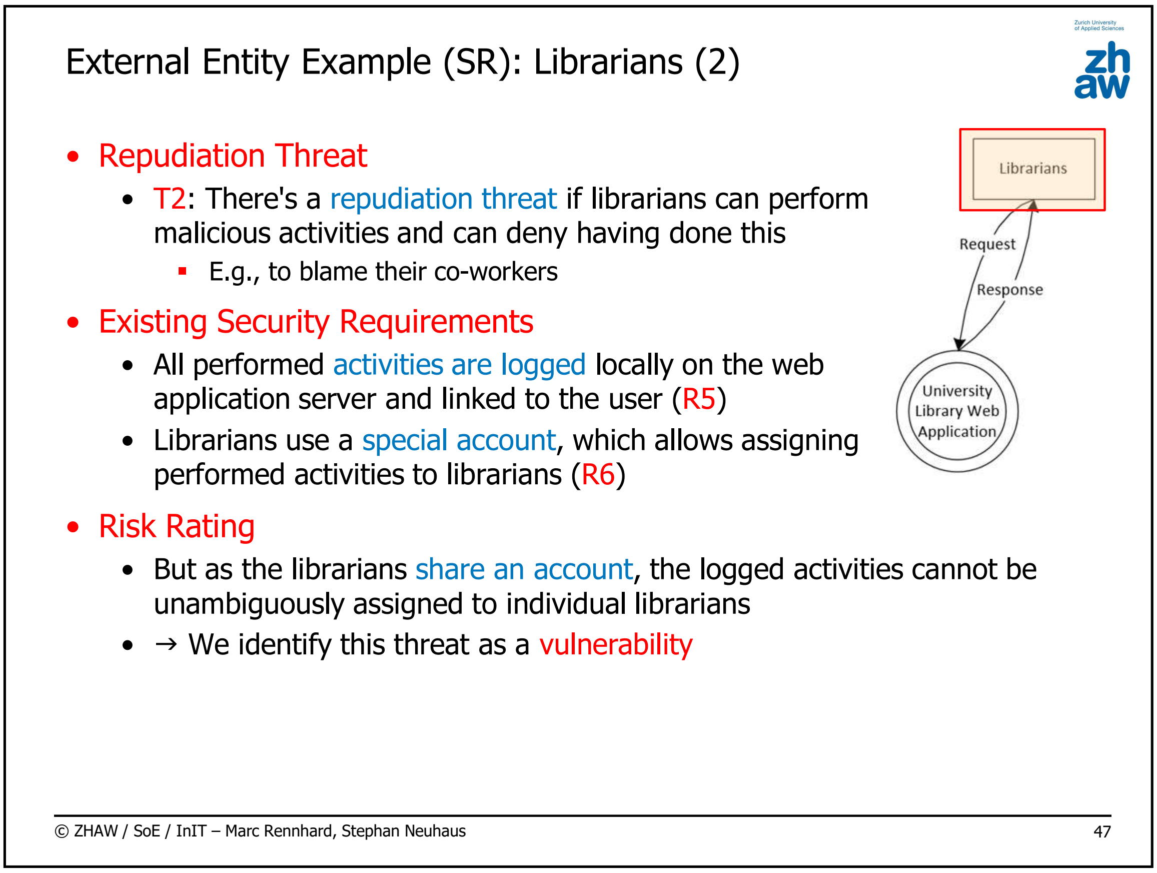
External Entity Example (SR): Librarians (1)

- Spoofing Threat
  - T1: There's a spoofing threat if the attacker manages to find out the credentials of a librarian
    - Could be done by script kiddies or bored students that want to disturb operations
- Existing Security Requirements
  - Usage of secret passwords should make it difficult to learn the credentials of librarians (R6)
- Risk Rating
  - Passwords are basically good, but the requirements do not specify a minimal password strength, weak passwords will likely be chosen
  - In addition, using a shared account among librarians increases the probability that the password will be written down next to the librarians' computers
  - Overall, this is not considered sufficient to adequately protect from the threat → We identify this threat as a vulnerability



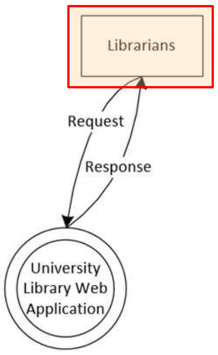
Spoofing Threats

Note that this is just one example of a spoofing threat and in reality, you should try to identify several of them. Another threat could be, e.g., to spoof a librarian by bribing him or by «convincing» a help desk employee (assuming this exists) that you are a librarian who has lost the password so the password should be reset to a new one. Or, as a further example, just omit the password during the login, which may grant access as a librarian in case this is not handled securely by the web application. Or install a physical keylogger at a computer used by a librarian that records all typed data, so the attacker can remove the keylogger later to get access to the information. A further option would be to compromise the computer or browser of a librarian so that it performs malicious requests in the background while the librarian is using the application. This is also a spoofing attack because with this, the attacker also wants to execute actions as a librarian. Of course, that's not really a realistic threat given the realistic attackers and their goals, but in high-security scenarios, such a threat should likely be considered.



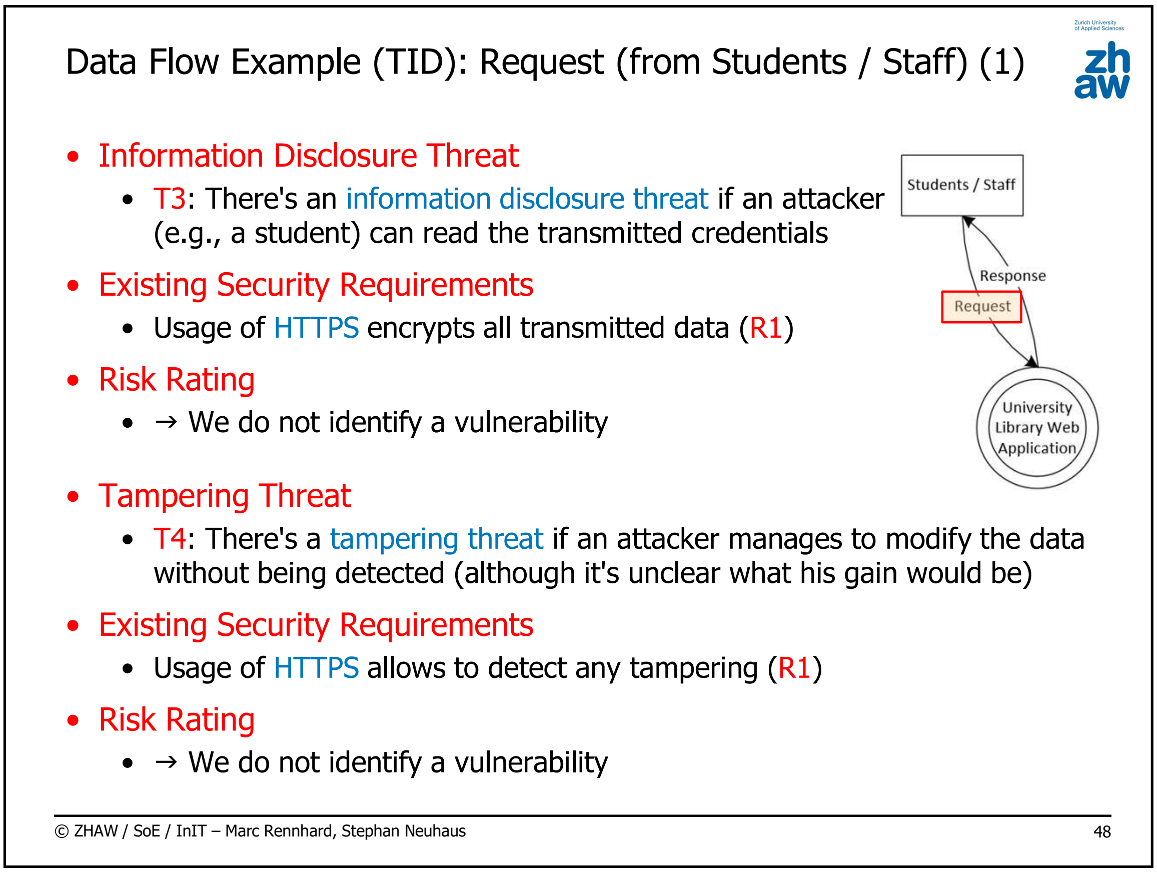
External Entity Example (SR): Librarians (2)

- Repudiation Threat
  - T2: There's a repudiation threat if librarians can perform malicious activities and can deny having done this
    - E.g., to blame their co-workers
- Existing Security Requirements
  - All performed activities are logged locally on the web application server and linked to the user (R5)
  - Librarians use a special account, which allows assigning performed activities to librarians (R6)
- Risk Rating
  - But as the librarians share an account, the logged activities cannot be unambiguously assigned to individual librarians
  - → We identify this threat as a vulnerability



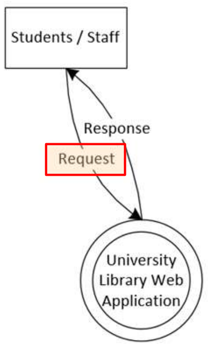
Assets and Vulnerabilities

Note that T1 directly targets an asset we identified earlier (credentials of users and librarians), so it’s certainly an important vulnerability. Similarly, T2 also directly targets an identified asset (the logs), so it’s also an important vulnerability.



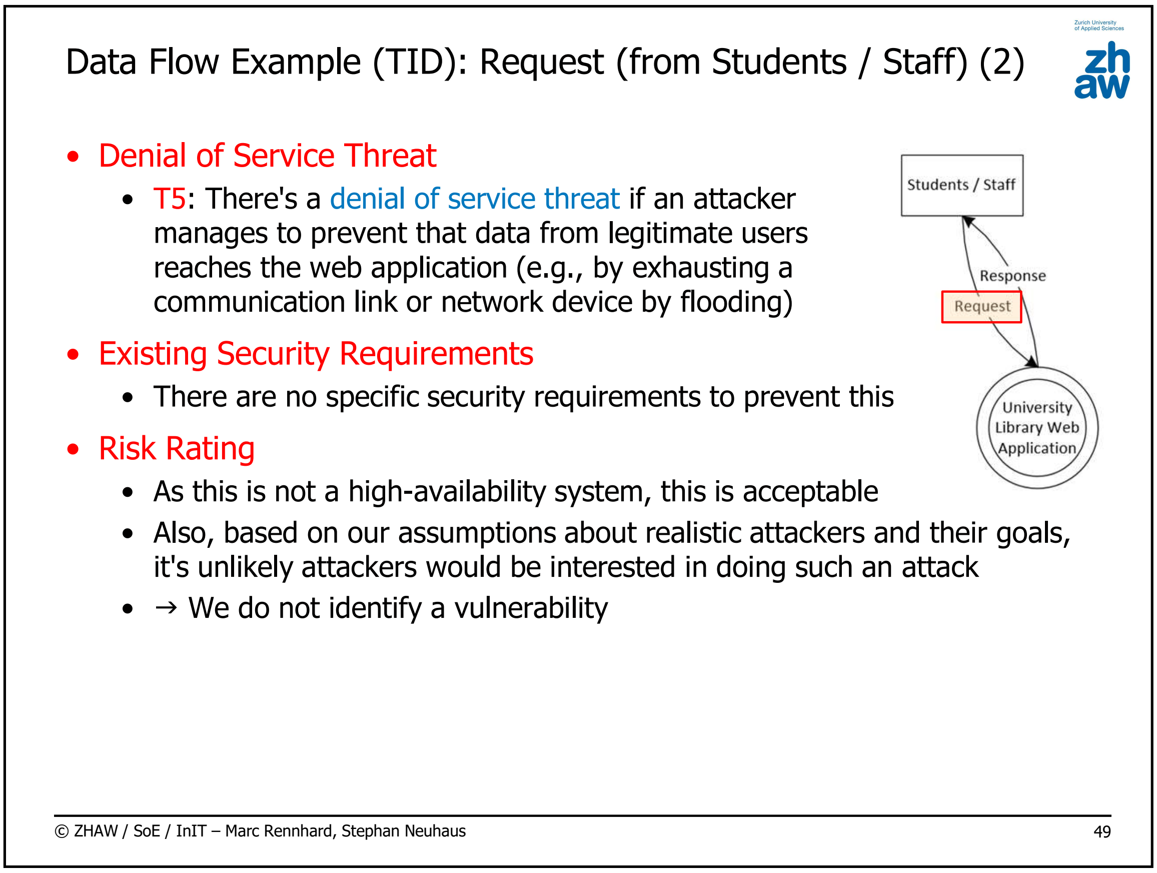
Data Flow Example (TID): Request (from Students / Staff) (1)

- Information Disclosure Threat
  - T3: There's an information disclosure threat if an attacker (e.g., a student) can read the transmitted credentials
- Existing Security Requirements
  - Usage of HTTPS encrypts all transmitted data (R1)
- Risk Rating
  - → We do not identify a vulnerability
- Tampering Threat
  - T4: There's a tampering threat if an attacker manages to modify the data without being detected (although it's unclear what his gain would be)
- Existing Security Requirements
  - Usage of HTTPS allows to detect any tampering (R1)
- Risk Rating
  - → We do not identify a vulnerability



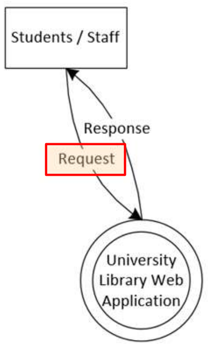
HTTPS

Of course, HTTPS can be broken with certificate spoofing, and doing such an attack in the University network is not very difficult. However, it's also a risky attack (e.g., for students), because there will be an error message in the browser, which means that the University might do a detailed investigation about what happened. Based on various log files, it may then be possible to identify the culprit, which likely would have drastic consequences for the student.



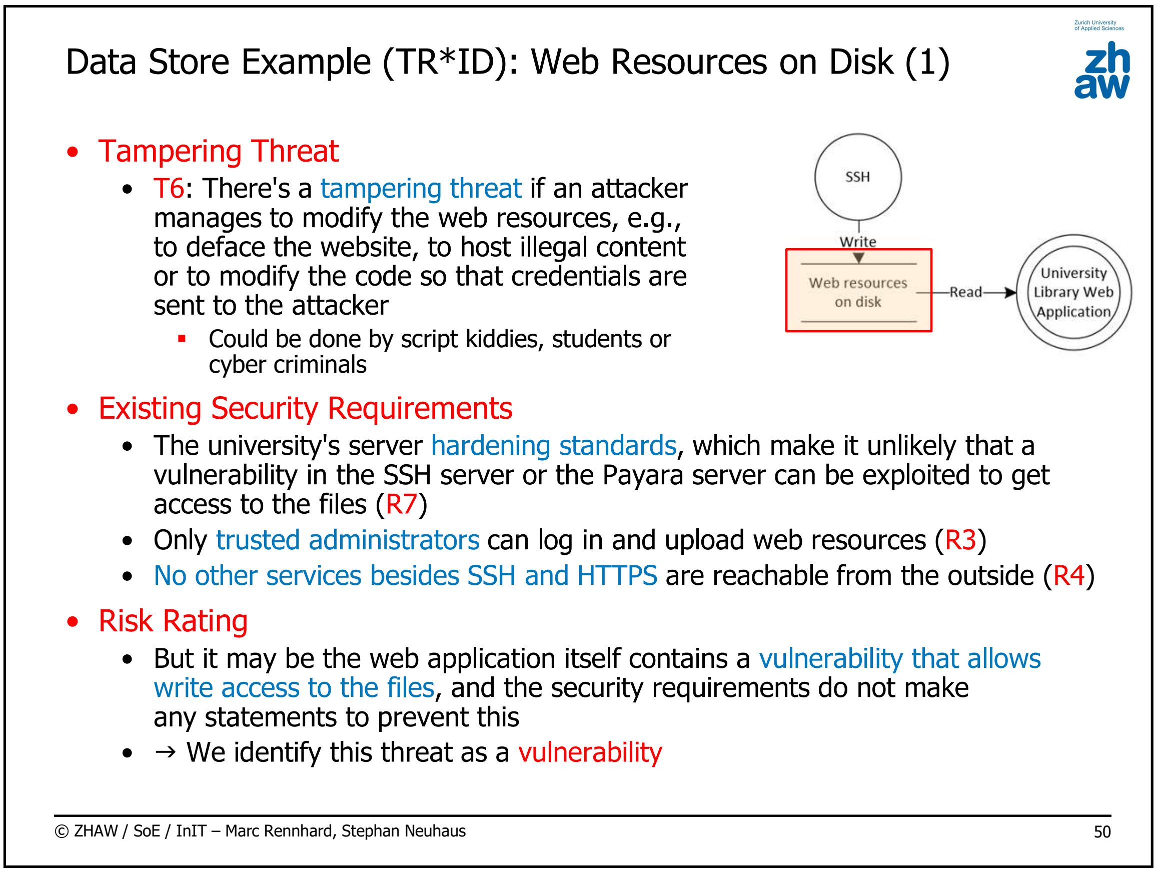
Data Flow Example (TID): Request (from Students / Staff) (2)

- Denial of Service Threat
  - T5: There's a denial of service threat if an attacker manages to prevent that data from legitimate users reaches the web application (e.g., by exhausting a communication link or network device by flooding)
- Existing Security Requirements
  - There are no specific security requirements to prevent this
- Risk Rating
  - As this is not a high-availability system, this is acceptable
  - Also, based on our assumptions about realistic attackers and their goals, it's unlikely attackers would be interested in doing such an attack
  - → We do not identify a vulnerability



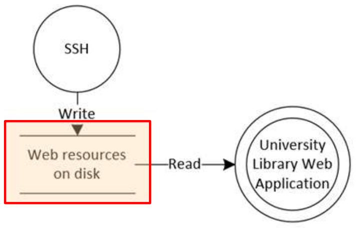
Denial of Service Threat against Data Flows

This threat means that the data flow from users to the web application is no longer possible. When considering DoS threats to data flows, then think about threats that prevent the data of other users to reach the end system (the web application), e.g., by flooding the communication link to exhaust a resource (e.g., a network device such as a router or a link). It is also possible to simply physically disrupt a data flow by cutting a cable.



Data Store Example (TR*ID): Web Resources on Disk (1)

- Tampering Threat
  - T6: There's a tampering threat if an attacker manages to modify the web resources, e.g., to deface the website, to host illegal content or to modify the code so that credentials are sent to the attacker
    - Could be done by script kiddies, students or cyber criminals

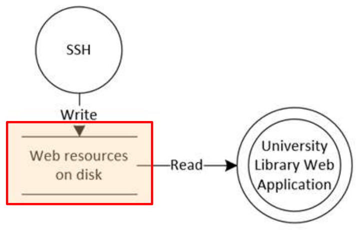


- Existing Security Requirements
  - The university's server hardening standards, which make it unlikely that a vulnerability in the SSH server or the Payara server can be exploited to get access to the files (R7)
  - Only trusted administrators can log in and upload web resources (R3)
  - No other services besides SSH and HTTPS are reachable from the outside (R4)
- Risk Rating
  - But it may be the web application itself contains a vulnerability that allows write access to the files, and the security requirements do not make any statements to prevent this
  - → We identify this threat as a vulnerability

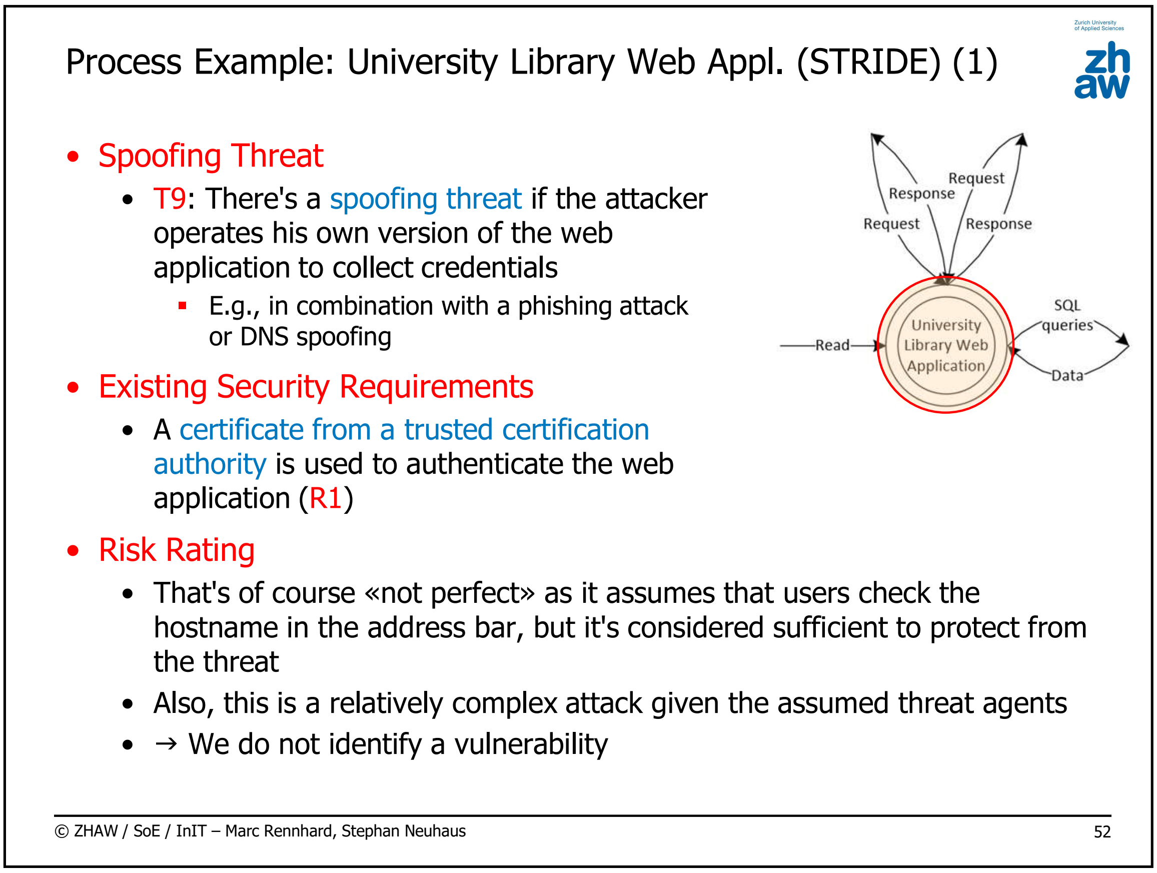
Repudiation Threats

*This data store does not contain logging or audit data, so repudiation threats are no issue.

Data Store Example (TR*ID): Web Resources on Disk (2)

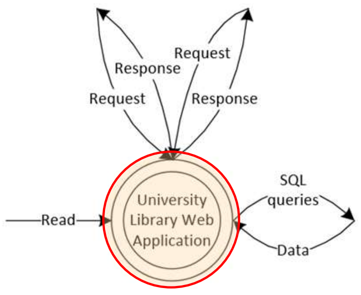


- Information Disclosure Threat
  - T7: There's an information disclosure threat if an attacker manages to read the web resources
  - This could help the attacker to identify vulnera-bilities in the web application code
    - Could be done by script kiddies, students or cyber criminals
- Existing Security Requirements / Risk Rating
  - → For the same reasons as with tampering (T6), this threat is also identified as a vulnerability
- Denial of Service Threat
  - T8: There's a denial of service threat if an attacker can exhaust storage
- Existing Security Requirements
  - None
- Risk Rating
  - → We do not identify a vulnerability, as the application does not write to this data store (unlike, e.g., a log file)



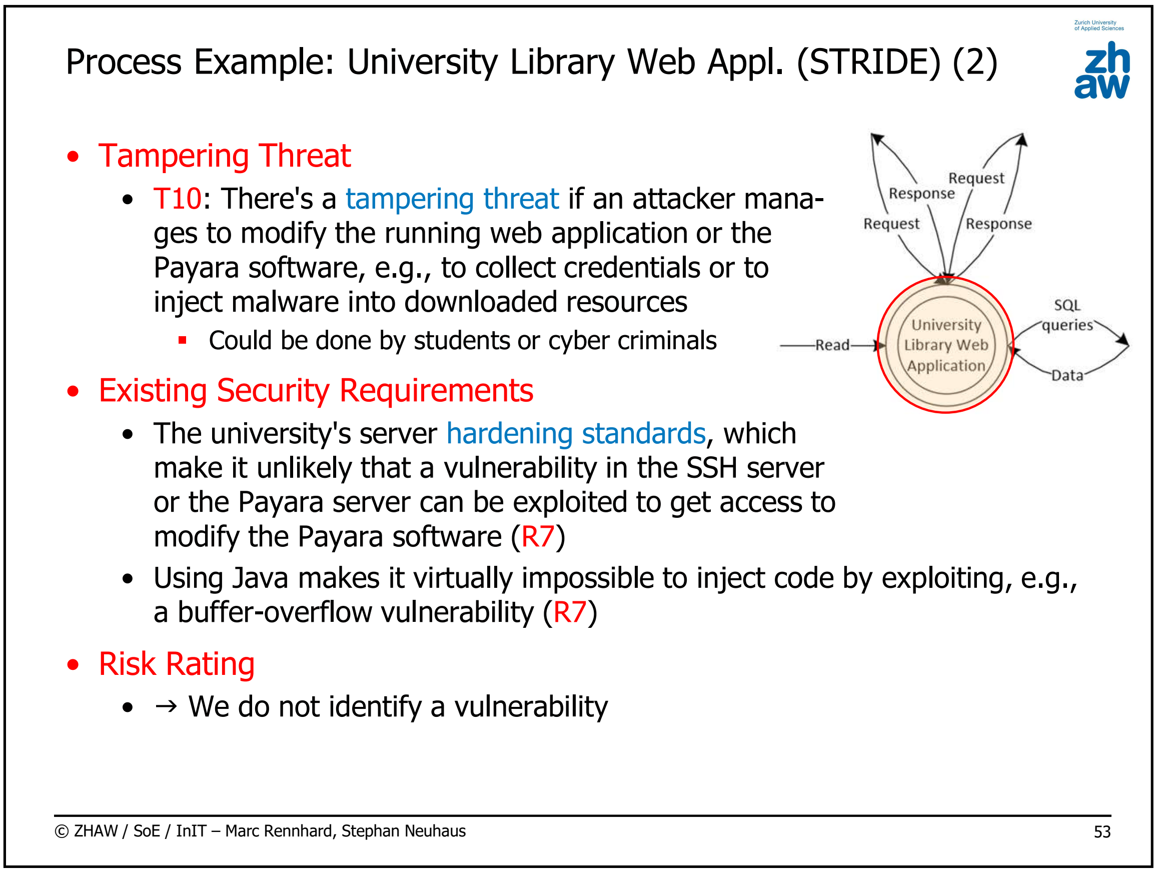
Process Example: University Library Web Appl. (STRIDE) (1)

- Spoofing Threat
  - T9: There's a spoofing threat if the attacker operates his own version of the web application to collect credentials
    - E.g., in combination with a phishing attack or DNS spoofing
- Existing Security Requirements
  - A certificate from a trusted certification authority is used to authenticate the web application (R1)
- Risk Rating
  - That's of course «not perfect» as it assumes that users check the hostname in the address bar, but it's considered sufficient to protect from the threat
  - Also, this is a relatively complex attack given the assumed threat agents
  - → We do not identify a vulnerability



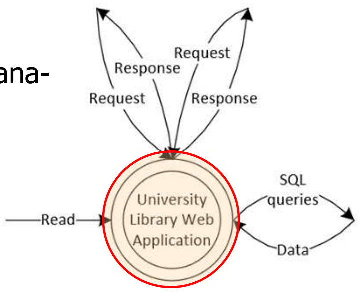
Spoofing the Web Application Server

Of course, a certificate cannot completely prevent this as it depends on whether users check the hostname in the browser's address bar. But unless users are clicking on links in e-mail messages that point to a spoofed server, this provides reasonable protection.



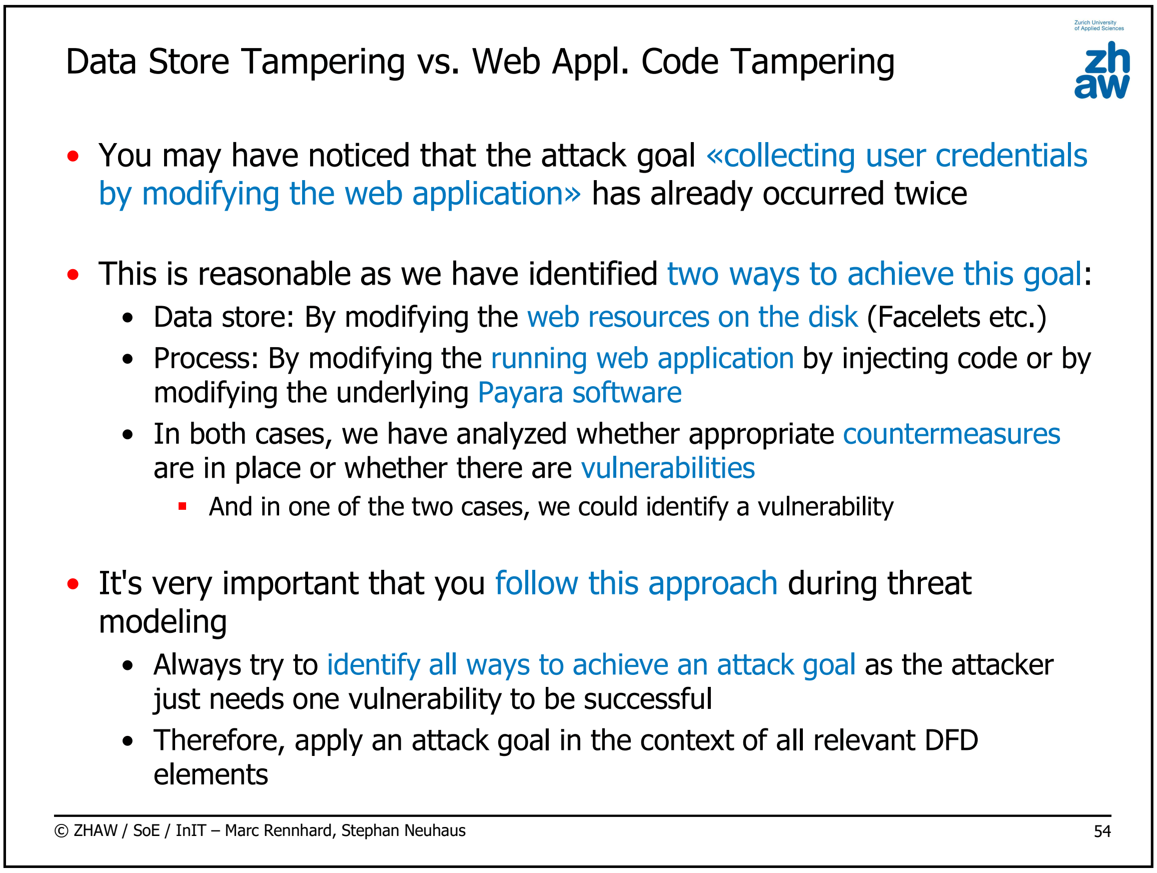
Process Example: University Library Web Appl. (STRIDE) (2)

- Tampering Threat
  - T10: There's a tampering threat if an attacker manages to modify the running web application or the Payara software, e.g., to collect credentials or to inject malware into downloaded resources
    - Could be done by students or cyber criminals
- Existing Security Requirements
  - The university's server hardening standards, which make it unlikely that a vulnerability in the SSH server or the Payara server can be exploited to get access to modify the Payara software (R7)
  - Using Java makes it virtually impossible to inject code by exploiting, e.g., a buffer-overflow vulnerability (R7)
- Risk Rating
  - → We do not identify a vulnerability



Tampering Threat

If the multiple process were further separated in the running web application and the Payara application server, then this tampering threat would also be split into multiple threats, one per DFD.

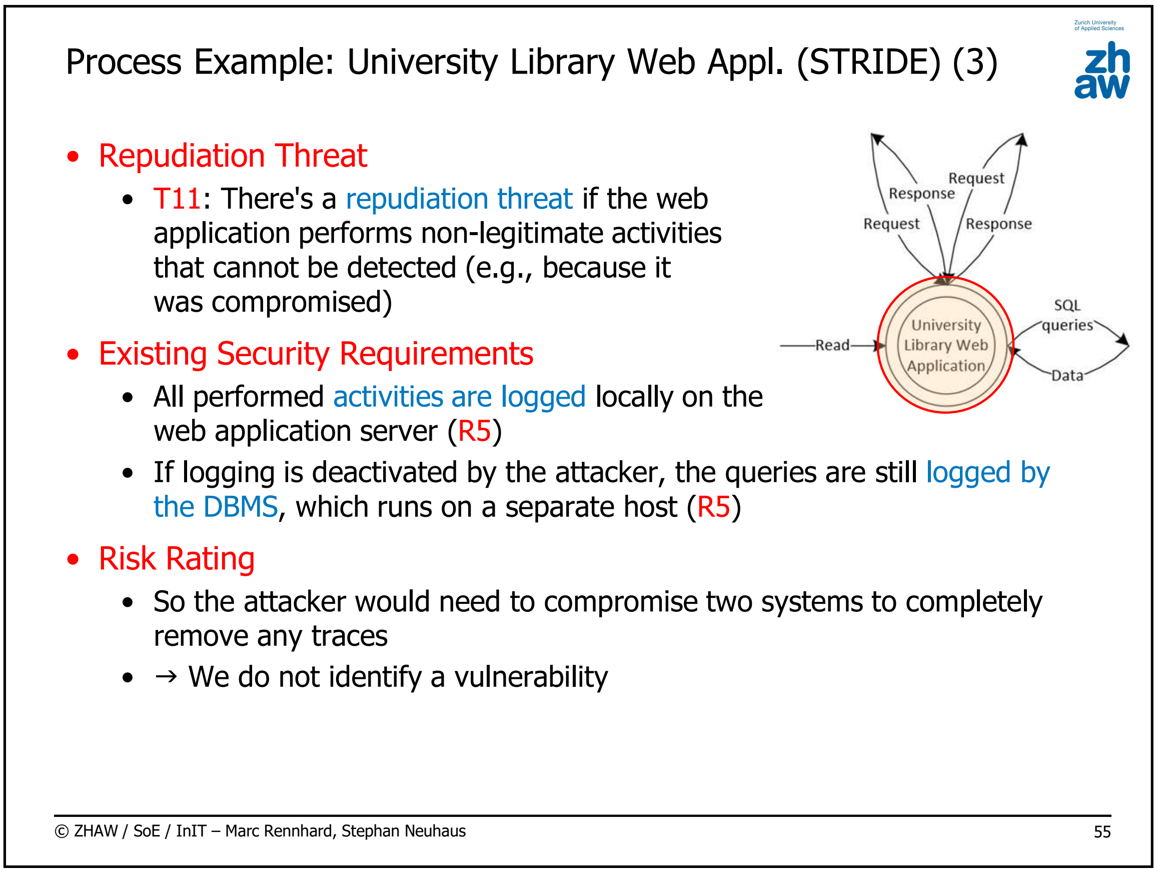


Data Store Tampering vs. Web Appl. Code Tampering

- You may have noticed that the attack goal «collecting user credentials by modifying the web application» has already occurred twice
- This is reasonable as we have identified two ways to achieve this goal:
  - Data store: By modifying the web resources on the disk (Facelets etc.)
  - Process: By modifying the running web application by injecting code or by modifying the underlying Payara software
  - In both cases, we have analyzed whether appropriate countermeasures are in place or whether there are vulnerabilities
    - And in one of the two cases, we could identify a vulnerability
- It's very important that you follow this approach during threat modeling
  - Always try to identify all ways to achieve an attack goal as the attacker just needs one vulnerability to be successful
  - Therefore, apply an attack goal in the context of all relevant DFD elements

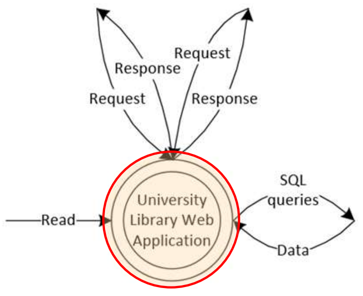
Web Application Code Tampering vs. Data Store Tampering

On the previous slide, we talked about tampering the web resources on disk and here we have again the resources to tamper with the process, so this is obviously an overlap. This sometimes happens with STRIDE when there are dependencies between elements, but its not a problem. As the web resources are in the data store, it makes sense to consider them when analyzing the data store, but as they are of course needed to run the application (the process), we have a dependency between the elements and as a result, the same resources are considered here as well. Note, however, that different vulnerabilities may be uncovered when considering the different dependent elements: With the data store, you focus on possible attacks that may manipulate the data (resources) when they reside on disk. With the process, one thinks about options to manipulate the code during execution, e.g., by exploiting a buffer-overflow vulnerability to inject code.



Process Example: University Library Web Appl. (STRIDE) (3)

- Repudiation Threat
  - T11: There's a repudiation threat if the web application performs non-legitimate activities that cannot be detected (e.g., because it was compromised)
- Existing Security Requirements
  - All performed activities are logged locally on the web application server (R5)
  - If logging is deactivated by the attacker, the queries are still logged by the DBMS, which runs on a separate host (R5)
- Risk Rating
  - So the attacker would need to compromise two systems to completely remove any traces
  - → We do not identify a vulnerability

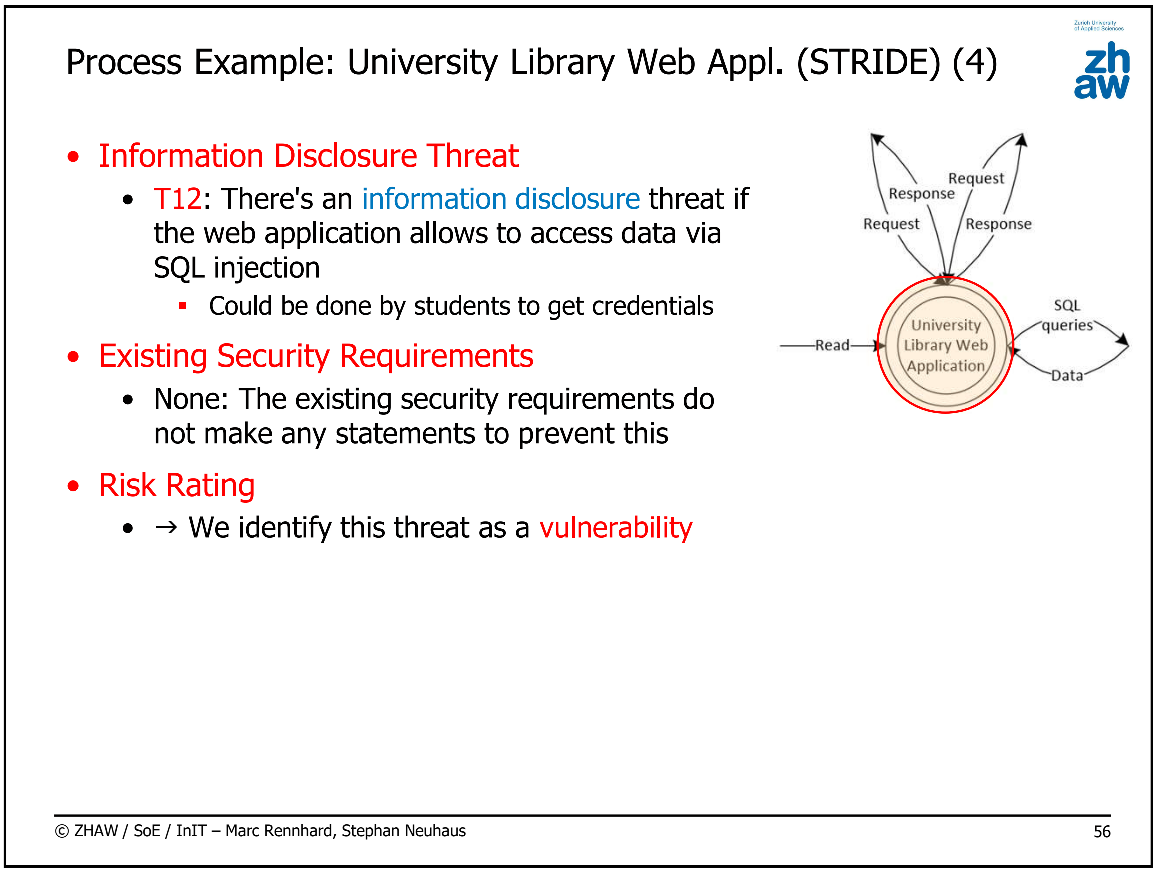


Reputation Threat – Entity, Process or Data Store?

The Reputation threat is a good example that shows that it is sometimes not obvious to which element a specific threat "belongs". According to STRIDE, reputation threats can be associated with entities, processes and data stores. Assuming we have users that use a web application which itself generates logs, three different element types are associated in the logging process, so how should one divide different threats among the three elements? Here are some examples that may help to make a reasonable separation of the element types:

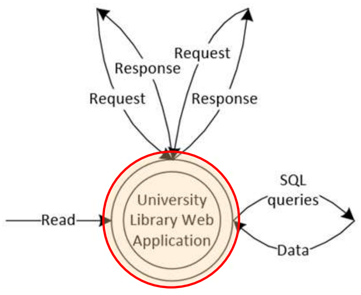
- *Data store*: To determine reputation threats here, you should think about scenarios that directly affect the data store (i.e., the logged data), without considering the web application or the system that produces the logs. Typical threats are a system administrator that often has the necessary (too many?) rights to remove his traces by removing some local log file entries.
- *Process*: To determine reputation threats here, think about potential problems in the process (e.g., the web application) that may result in manipulating the log files. For instance, there may be an input validation problem that allows the user that accesses the web application to write arbitrary system files, which also allows him to modify log files.
- *Entity* (user): Finally, to determine reputation threats here, think about fundamental logging problems, e.g., no logging is done at all, no timestamps are associated with the logged data, multiple user sharing accounts, etc. One could also argue that this "belongs" to the process – as it is an error in the process that no logging is done at all, but in practice, such fundamental logging problems are often associated with the entity

In general, when thinking about STRIDE and analyzing an element, always ask yourself what could go wrong HERE (with this element), which usually results in making the right decision.



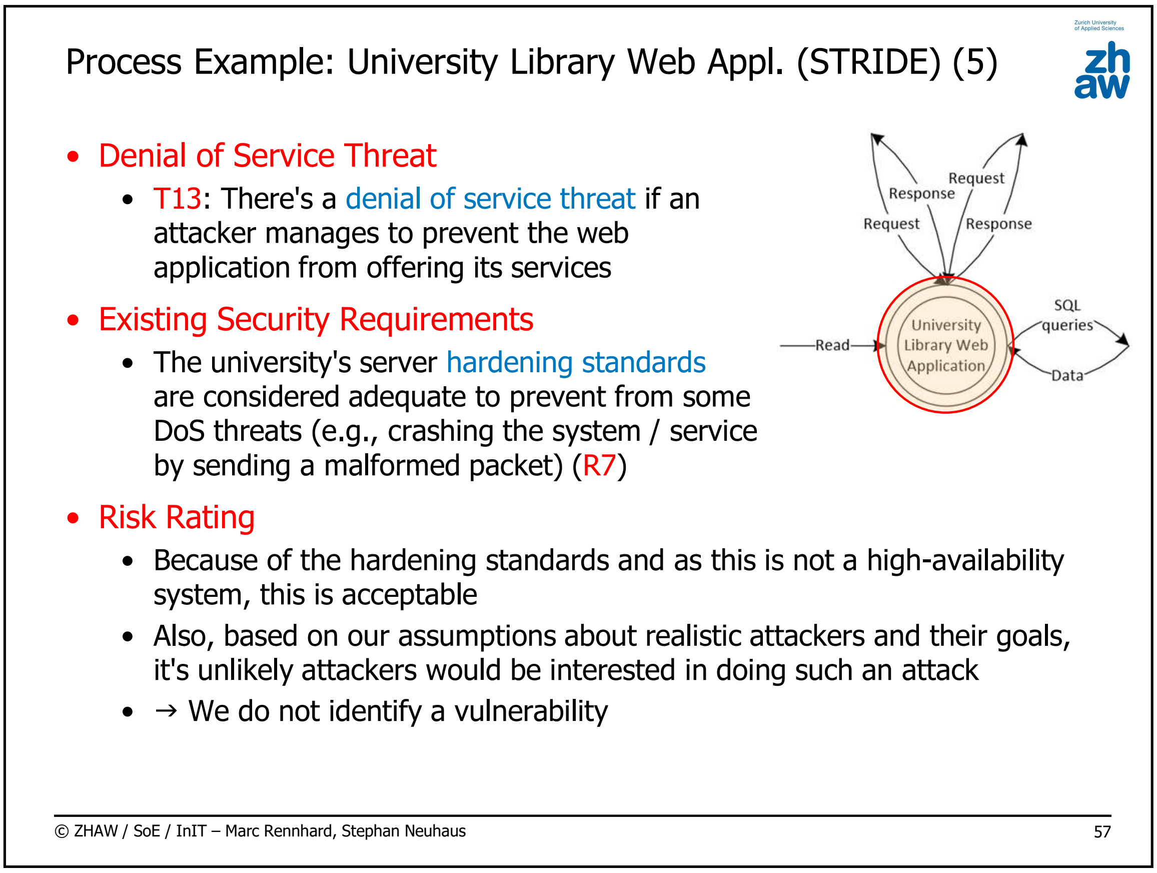
Process Example: University Library Web Appl. (STRIDE) (4)

- Information Disclosure Threat
  - T12: There's an information disclosure threat if the web application allows to access data via SQL injection
    - Could be done by students to get credentials
- Existing Security Requirements
  - None: The existing security requirements do not make any statements to prevent this
- Risk Rating
  - → We identify this threat as a vulnerability



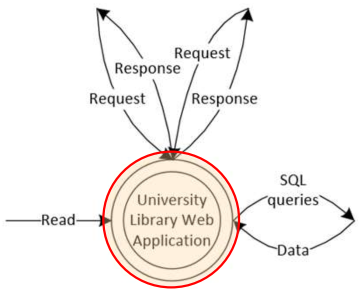
Information Disclosure and Processes

The corresponding threats affect both the web application code itself (e.g., the program that is running) but especially also the data provided by the process. In the case of a web application, this includes all data in the database as it is the task of the web application to provide controlled access to this data, depending on the user rights, and thereby preventing attacks such as SQL injection. Similarly, when thinking about information disclosure threats against web applications, one should also consider attacks to get user credentials or session IDs via Cross-Site Scripting (XSS), as both are sensitive pieces of information maintained by the web application. Although these attacks eventually "happen" in the browser of the user interacting with the web application, the underlying threat/attack targets the web application to insert malicious JavaScripts into web pages.



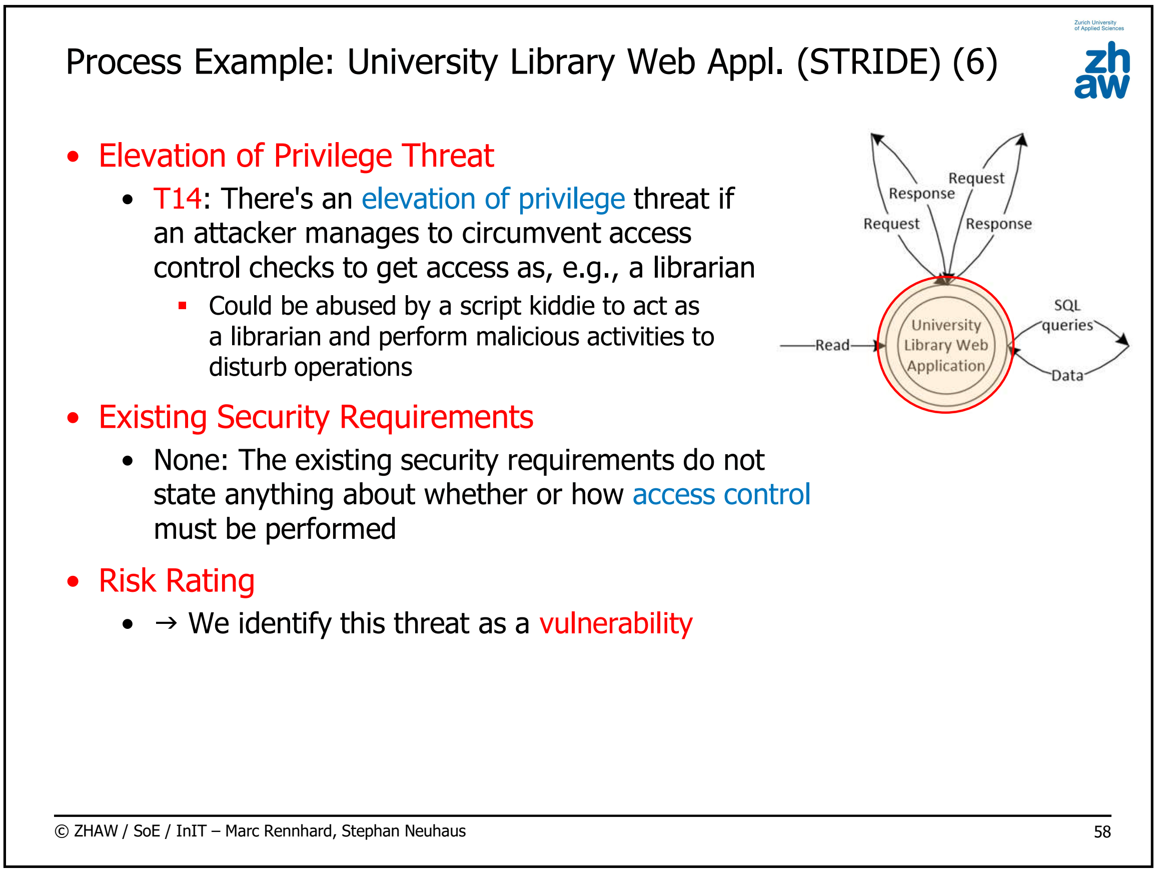
Process Example: University Library Web Appl. (STRIDE) (5)

- Denial of Service Threat
  - T13: There's a denial of service threat if an attacker manages to prevent the web application from offering its services
- Existing Security Requirements
  - The university's server hardening standards are considered adequate to prevent from some DoS threats (e.g., crashing the system / service by sending a malformed packet) (R7)
- Risk Rating
  - Because of the hardening standards and as this is not a high-availability system, this is acceptable
  - Also, based on our assumptions about realistic attackers and their goals, it's unlikely attackers would be interested in doing such an attack
  - → We do not identify a vulnerability



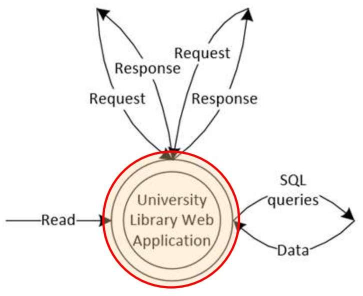
Denial of Service Threat against Processes

Here, think about threats that take the process “offline”, e.g., by flooding the web application so it can no longer server legitimate user or by sending the server a malformed packet, so it crashes.



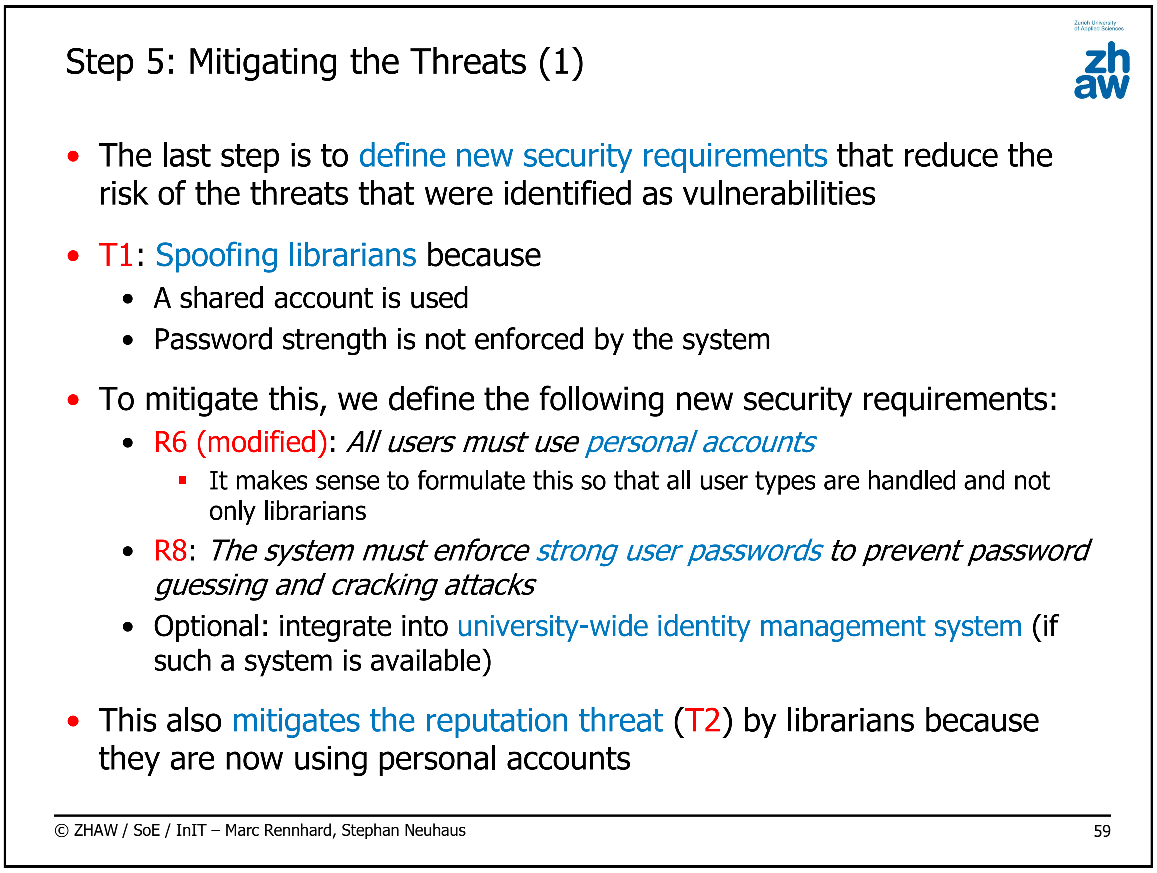
Process Example: University Library Web Appl. (STRIDE) (6)

- Elevation of Privilege Threat
  - T14: There's an elevation of privilege threat if an attacker manages to circumvent access control checks to get access as, e.g., a librarian
    - Could be abused by a script kiddie to act as a librarian and perform malicious activities to disturb operations
- Existing Security Requirements
  - None: The existing security requirements do not state anything whether or how access control must be performed
- Risk Rating
  - → We identify this threat as a vulnerability



Assets and Vulnerabilities

Just like T1 and T2, threats T5 and T6 also result in important vulnerabilities, as both directly target assets we identified earlier. T5 may be exploited to access personal user data and maybe also credentials (assuming they are stored in the database in plaintext) and T6 may also be used to access personal user data

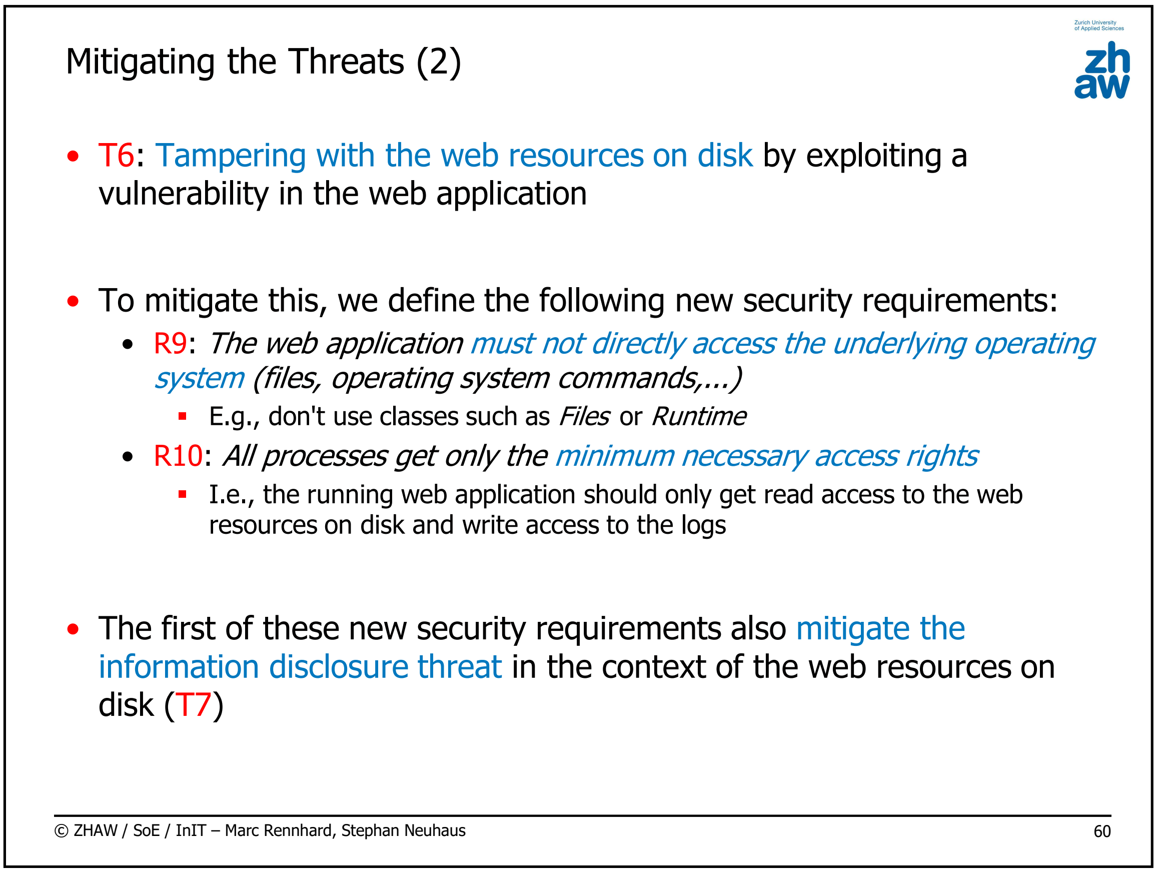


Step 5: Mitigating the Threats (1)

- The last step is to define new security requirements that reduce the risk of the threats that were identified as vulnerabilities
- T1: Spoofing librarians because
  - A shared account is used
  - Password strength is not enforced by the system
- To mitigate this, we define the following new security requirements:
  - R6 (modified): *All users must use personal accounts*
    - It makes sense to formulate this so that all user types are handled and not only librarians
  - R8: *The system must enforce strong user passwords to prevent password guessing and cracking attacks*
  - Optional: integrate into university-wide identity management system (if such a system is available)
- This also mitigates the reputation threat (T2) by librarians because they are now using personal accounts

Generalizing Security Requirements

Note that the security requirements are often formulated in a more general way than needed to mitigate the identified threat. For instance, it would be enough to write *Librarians must use personal accounts* to mitigate the threat. However, this requirements seems to be a very reasonable requirement for all users in the system and therefore, it is formulated accordingly.

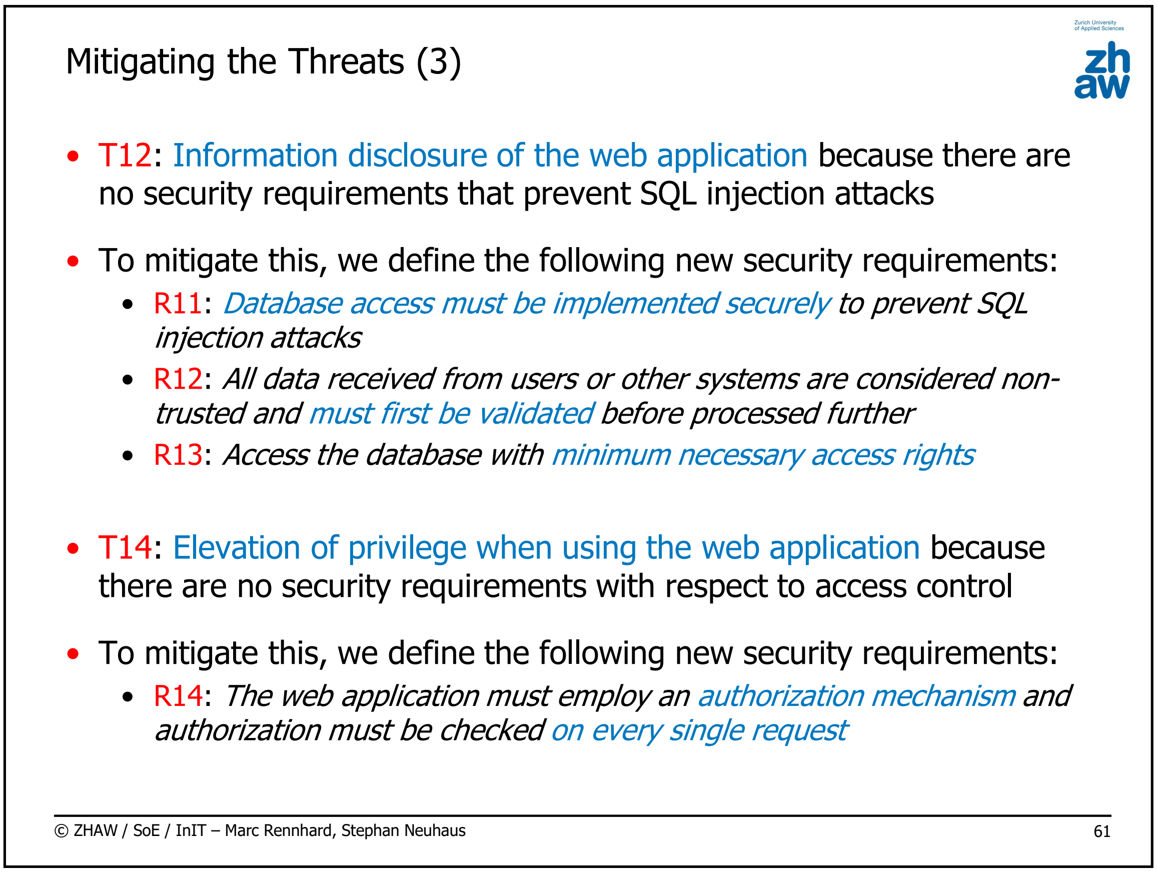


Mitigating the Threats (2)

- T6: Tampering with the web resources on disk by exploiting a vulnerability in the web application
- To mitigate this, we define the following new security requirements:
  - R9: *The web application must not directly access the underlying operating system (files, operating system commands,...)*
    - E.g., don't use classes such as *Files* or *Runtime*
  - R10: *All processes get only the minimum necessary access rights*
    - I.e., the running web application should only get read access to the web resources on disk and write access to the logs
- The first of these new security requirements also mitigate the information disclosure threat in the context of the web resources on disk (T7)

Accesses to the Operating System

- Directly accessing files from the web application code is risky, as this may allow an attacker to find a way to access the web resources in the file system. And note that this is also a reasonable requirement as usually, such a direct access is not required in the web application code.
- Likewise, accessing operating systems is risky as this may also be exploited by an attacker by an attacker to get access to the operating system and therefore the web resources in the file system.



Mitigating the Threats (3)

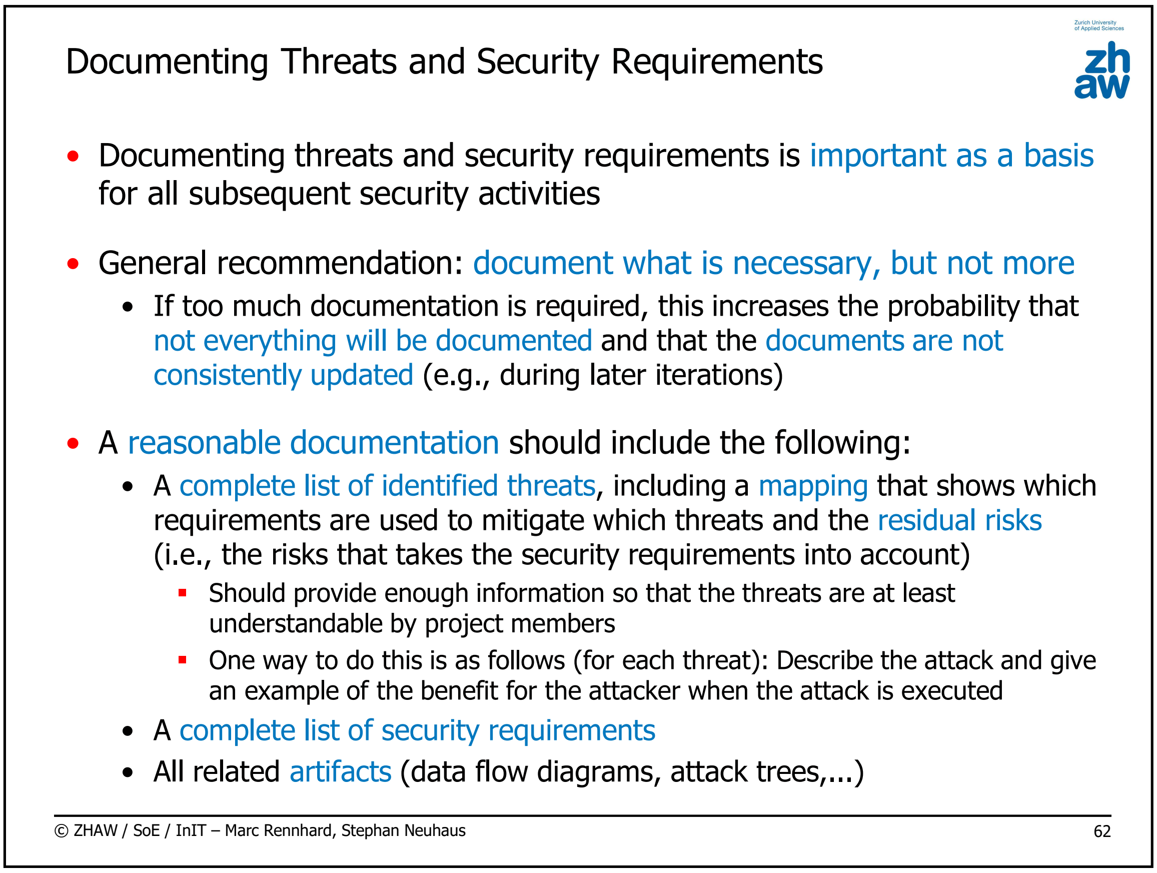
- T12: Information disclosure of the web application because there are no security requirements that prevent SQL injection attacks
- To mitigate this, we define the following new security requirements:
  - R11: *Database access must be implemented securely to prevent SQL injection attacks*
  - R12: *All data received from users or other systems are considered non-trusted and must first be validated before processed further*
  - R13: *Access the database with minimum necessary access rights*
- T14: Elevation of privilege when using the web application because there are no security requirements with respect to access control
- To mitigate this, we define the following new security requirements:
  - R14: *The web application must employ an authorization mechanism and authorization must be checked on every single request*

STRIDE may lead to Defense in Depth

A nice side effect of STRIDE is that it often leads to defense in depth if you use It correctly:

- Assume we identify a threat «spoofing the admin by guessing credentials»
  - This could lead to a security requirement «strong admin passwords»
- Assume we identify another threat «attacker who knows admin credentials manipulates the web application log file to hide the traces of an attack»
  - We could argue that this is already covered with the security requirement defined above
- But if we are using STRIDE correctly, we should think about how to prevent the second threat without relying on the first security requirement
  - In this case, we could formulate another requirement which states that «administrators should not get root access but only the necessary rights»
  - With this, we have achieved defense in depth, because if an attacker wants to manipulate the web log files to hide the traces, he first has to get the admin credentials to get access to the system and then, in addition, find a way to elevate his privileges on the system to get write access to the logs

To summarize, thinking about threats in the context of each DFD element individually and defining appropriate security requirements will likely result in multiple security requirements that ultimately protect from the same attack scenario. That's great from the point of view of a defense in depth approach because defense in depth is usually a desired property. When reviewing the complete list of security requirements, it is of course always possible to reduce the list of security requirements if you think that you have «too much security», e.g., by reducing the number of security requirements that help to protect from a specific attack from three to two.



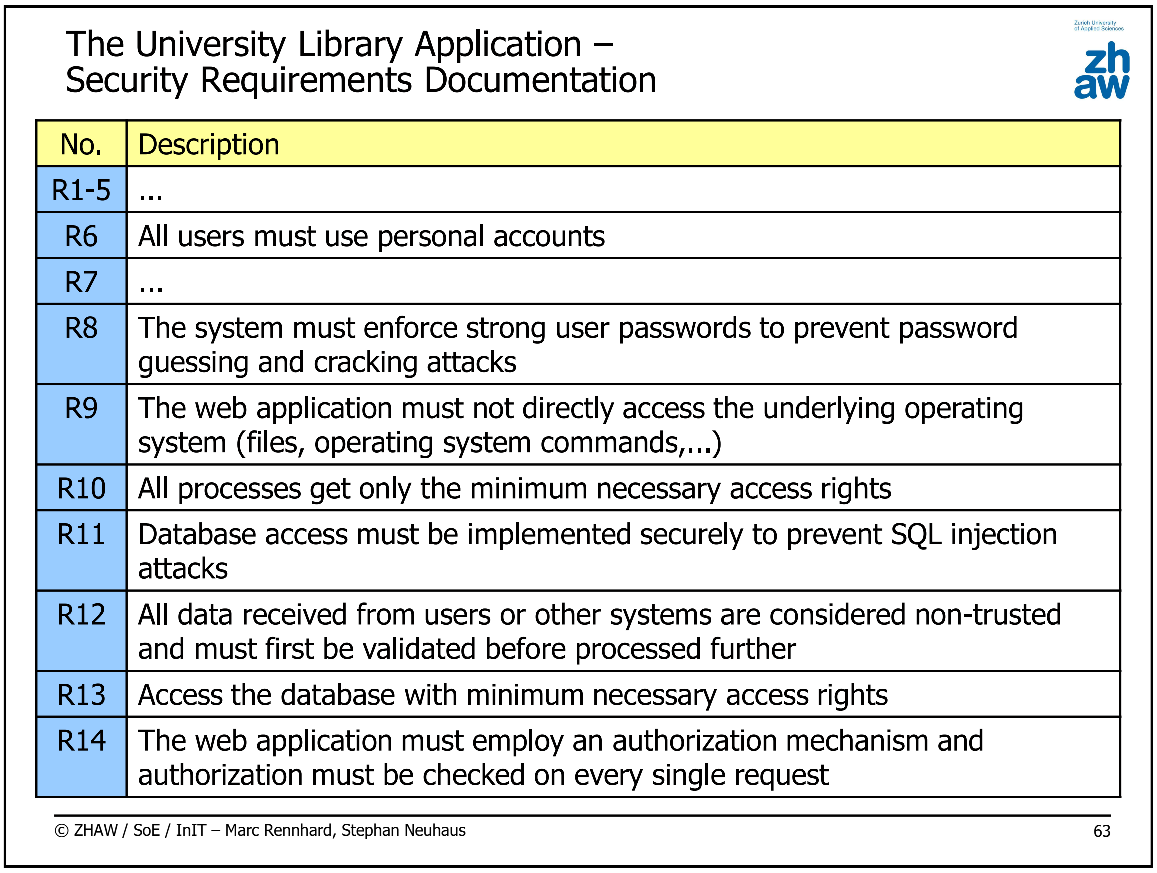
Documenting Threats and Security Requirements

- Documenting threats and security requirements is important as a basis for all subsequent security activities
- General recommendation: document what is necessary, but not more
  - If too much documentation is required, this increases the probability that not everything will be documented and that the documents are not consistently updated (e.g., during later iterations)
- A reasonable documentation should include the following:
  - A complete list of identified threats, including a mapping that shows which requirements are used to mitigate which threats and the residual risks (i.e., the risks that takes the security requirements into account)
    - Should provide enough information so that the threats are at least understandable by project members
    - One way to do this is as follows (for each threat): Describe the attack and give an example of the benefit for the attacker when the attack is executed
  - A complete list of security requirements
  - All related artifacts (data flow diagrams, attack trees,...)

Documenting Threats and Security Requirements

There is no widely used or accepted standard about how security requirements should be documented. In general, you can use virtually any method / template (e.g., wikis, spread sheets, custom developed tools etc.) you like, but what's important is that you make sure that this is then used consistently during the entire project and that security requirements are truly documented. Because if some of them are not documented, they will likely be forgotten, and no countermeasures will be implemented.

Remember that security requirements engineering is usually an iterative process, so documentation must typically be updated during each iteration. With too much documentation, it can easily happen that not everything will be maintained and updated consistently. So, it's important to focus on the most important details when documenting security requirements and leave out what's not needed.

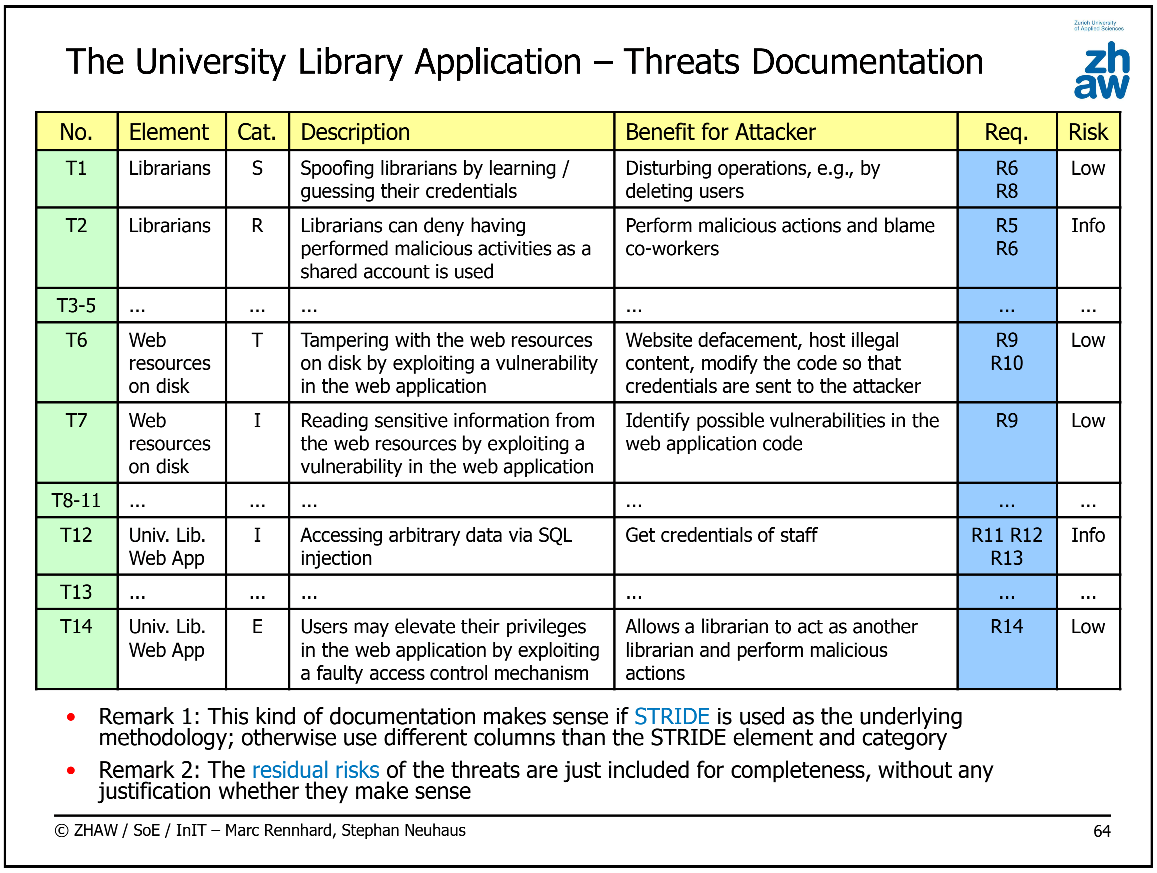


The University Library Application – Security Requirements Documentation

No.	Description
R1-5	...
R6	All users must use personal accounts
R7	...
R8	The system must enforce strong user passwords to prevent password guessing and cracking attacks
R9	The web application must not directly access the underlying operating system (files, operating system commands,...)
R10	All processes get only the minimum necessary access rights
R11	Database access must be implemented securely to prevent SQL injection attacks
R12	All data received from users or other systems are considered non-trusted and must first be validated before processed further
R13	Access the database with minimum necessary access rights
R14	The web application must employ an authorization mechanism and authorization must be checked on every single request

Include all Security Requirements

Of course, the list should include all 14 requirements. For space restrictions, only the newly defined (or adapted in the case of R6) security requirements are shown.



The University Library Application – Threats Documentation

No.	Element	Cat.	Description	Benefit for Attacker	Req.	Risk
T1	Librarians	S	Spoofing librarians by learning / guessing their credentials	Disturbing operations, e.g., by deleting users	R6 R8	Low
T2	Librarians	R	Librarians can deny having performed malicious activities as a shared account is used	Perform malicious actions and blame co-workers	R5 R6	Info
T3-5	…	…	…	…	…	…
T6	Web resources on disk	T	Tampering with the web resources on disk by exploiting a vulnerability in the web application	Website defacement, host illegal content, modify the code so that credentials are sent to the attacker	R9 R10	Low
T7	Web resources on disk	I	Reading sensitive information from the web resources by exploiting a vulnerability in the web application	Identify possible vulnerabilities in the web application code	R9	Low
T8-11	…	…	…	…	…	…
T12	Univ. Lib. Web App	I	Accessing arbitrary data via SQL injection	Get credentials of staff	R11 R12 R13	Info
T13	…	…	…	…	…	…
T14	Univ. Lib. Web App	E	Users may elevate their privileges in the web application by exploiting a faulty access control mechanism	Allows a librarian to act as another librarian and perform malicious actions	R14	Low

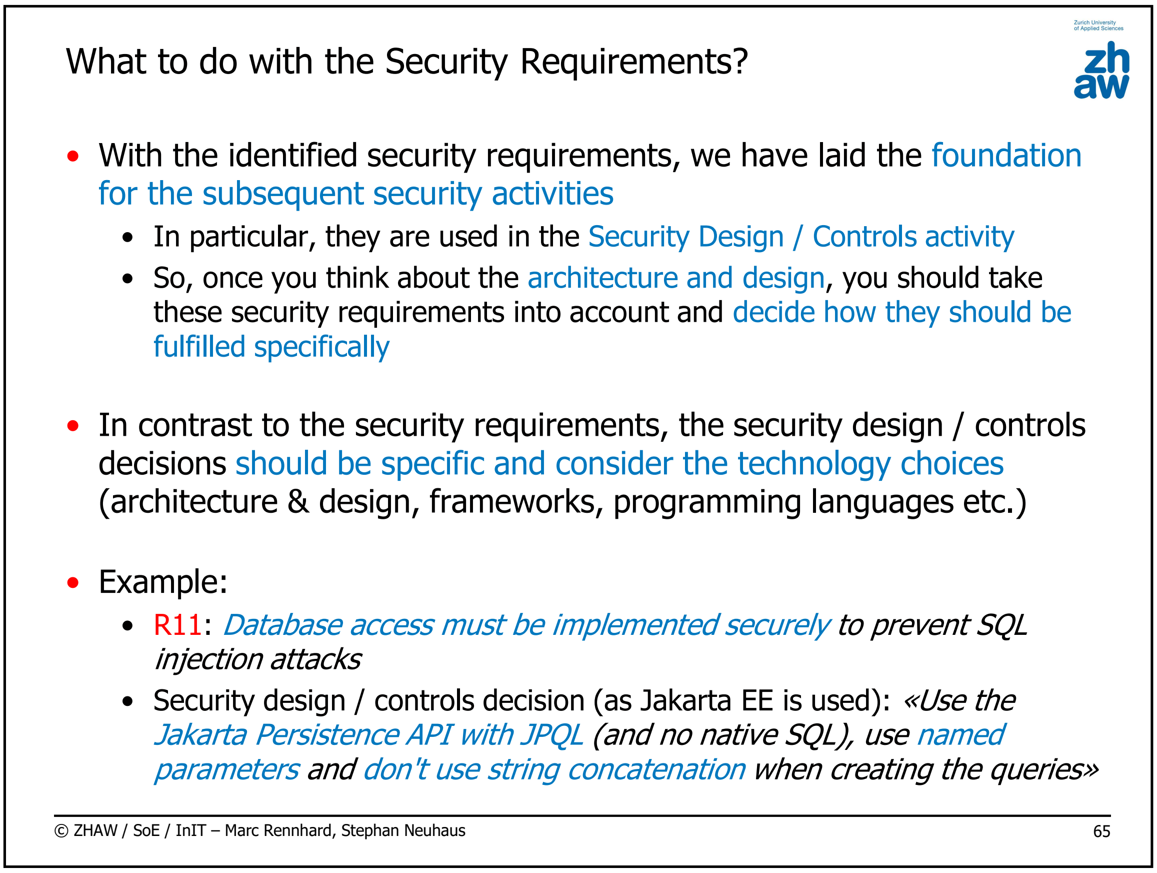
- Remark 1: This kind of documentation makes sense if STRIDE is used as the underlying methodology; otherwise use different columns than the STRIDE element and category
- Remark 2: The residual risks of the threats are just included for completeness, without any justification whether they make sense

Documentation of Threats

The documentation of the threats can be different from the one above. But when using STRIDE as the basis, it's reasonable to include the DFD elements and the STRIDE categories as well. On the other hand, a minimal threat documentation could also just consist of threat numbers, descriptions and corresponding security requirements.

Include all Threats

Of course, the list should include all 14 threats. For space restrictions, only the threats where we identified vulnerabilities are shown.



What to do with the Security Requirements?

- With the identified security requirements, we have laid the foundation for the subsequent security activities
  - In particular, they are used in the Security Design / Controls activity
  - So, once you think about the architecture and design, you should take these security requirements into account and decide how they should be fulfilled specifically
- In contrast to the security requirements, the security design / controls decisions should be specific and consider the technology choices (architecture & design, frameworks, programming languages etc.)
- Example:
  - R11: *Database access must be implemented securely to prevent SQL injection attacks*
  - Security design / controls decision (as Jakarta EE is used): *«Use the Jakarta Persistence API with JPQL (and no native SQL), use named parameters and don't use string concatenation when creating the queries»*

Specific Security Design / Control Decisions

As stated above, the specific security measures should take the technology decisions into account. But of course, the requirements can also influence the architecture & design of the system.

For instance, if a security requirement states that

- «Administrative areas in the web shop must isolated from public and customer areas as much as possible to minimize the risk that normal users can abuse powerful admin functionality»

Then a resulting security design decision could be

- «Use a completely separate web application for administrative functions (that only shares the database with the actual web shop) 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»

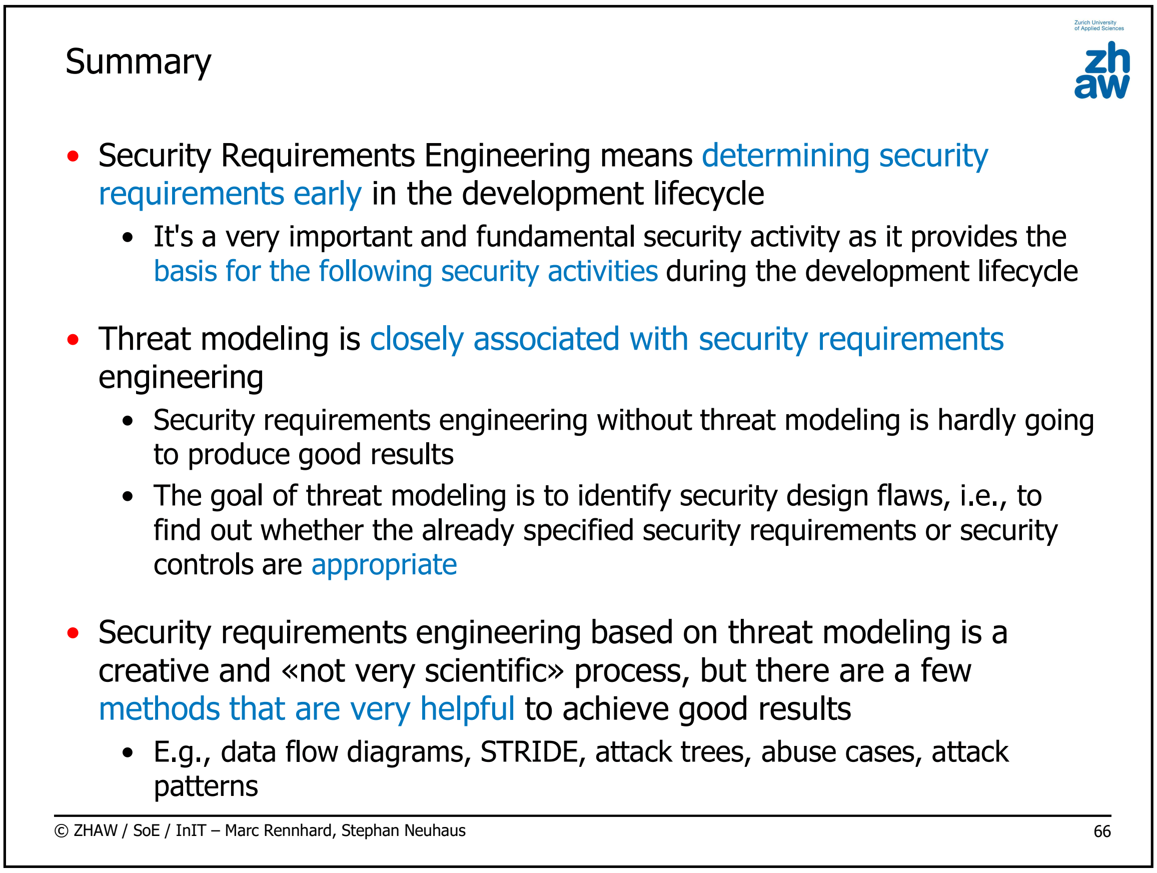
So, in this case, the specific security mechanism has a major impact on the overall architecture and design and also on networking (firewalls, VPNs).

Security Patterns

A possible option to choose specific security controls or make specific security design decisions is using so-called security patterns. These are standard approaches (therefore the term «pattern») to solve recurring security problems. Although well described in the literature, they are not often used in practice. One reason for this is that they solve typical, well-known security problems, but experienced security engineers typically know how to solve them anyway, so the patterns are of limited value to them. And when non-typical security problems have to be solved, the patterns usually do not provide any help because there's no corresponding pattern, so tailored solutions have to be found anyway. However, security patterns still have some value, e.g., for security engineers that are not so experienced yet. The patterns may help them to get a good overview over reasonable solutions to typical security problems and when they are unsure how a specific security solutions (that exists as a pattern) should be solved in practice.

To learn more about security patterns or security control catalogues, refer, e.g., to the following:

- Open Group Security Design Patterns: Security: *https://pubs.opengroup.org/onlinepubs/9299969899/toc.pdf*
- NIST SP 800-53: *https://nvd.nist.gov/800-53*



Summary

- Security Requirements Engineering means determining security requirements early in the development lifecycle
  - It's a very important and fundamental security activity as it provides the basis for the following security activities during the development lifecycle
- Threat modeling is closely associated with security requirements engineering
  - Security requirements engineering without threat modeling is hardly going to produce good results
  - The goal of threat modeling is to identify security design flaws, i.e., to find out whether the already specified security requirements or security controls are appropriate
- Security requirements engineering based on threat modeling is a creative and «not very scientific» process, but there are a few methods that are very helpful to achieve good results
  - E.g., data flow diagrams, STRIDE, attack trees, abuse cases, attack patterns

Security Requirements Engineering / Threat Modeling – Additional Remarks

As said before, this is not a scientific method but rather a creative process

- As a result, this is best done in a team, where different views and knowledge areas of the participants can be considered

What's the appropriate level of detail to use for the DFD?

- A high-level DFD as we used it here is usually well suited to analyze a system with «typical security requirements»
- In some cases, a more detailed view is appropriate
- To analyze very security-critical processes (e.g., the entire payment process in an e-banking system), it may be reasonable to identify all involved components in detail

Don't make security assumptions!

- You analyze what you see and not what «you would like to see»
- If a security requirements or control is not in the documentation and no one has told you about it in an interview, assume it's not there
- It's good to be on the conservative side, if someone tells you afterwards that «this requirement has been defined or this control is in place», no great harm was done

Don't forget insiders

- Often, security analyses focus on external attackers, but in reality, internal attacks also happen from time to time
- This includes malicious insiders (disgruntled employees)...
- ...but also accidents by insiders (e.g., an administrator who connects an infected laptop to the network in the server room)

Document the threats and the security requirements and adapt the documentation during each iteration (if needed)

- Using any drawing tool for DFDs together with a spreadsheet is fine
- Microsoft has made available a Wondows-only Threat Modeling Tool, which allows drawing DFDs and assign threats directly to the elements: *https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling*

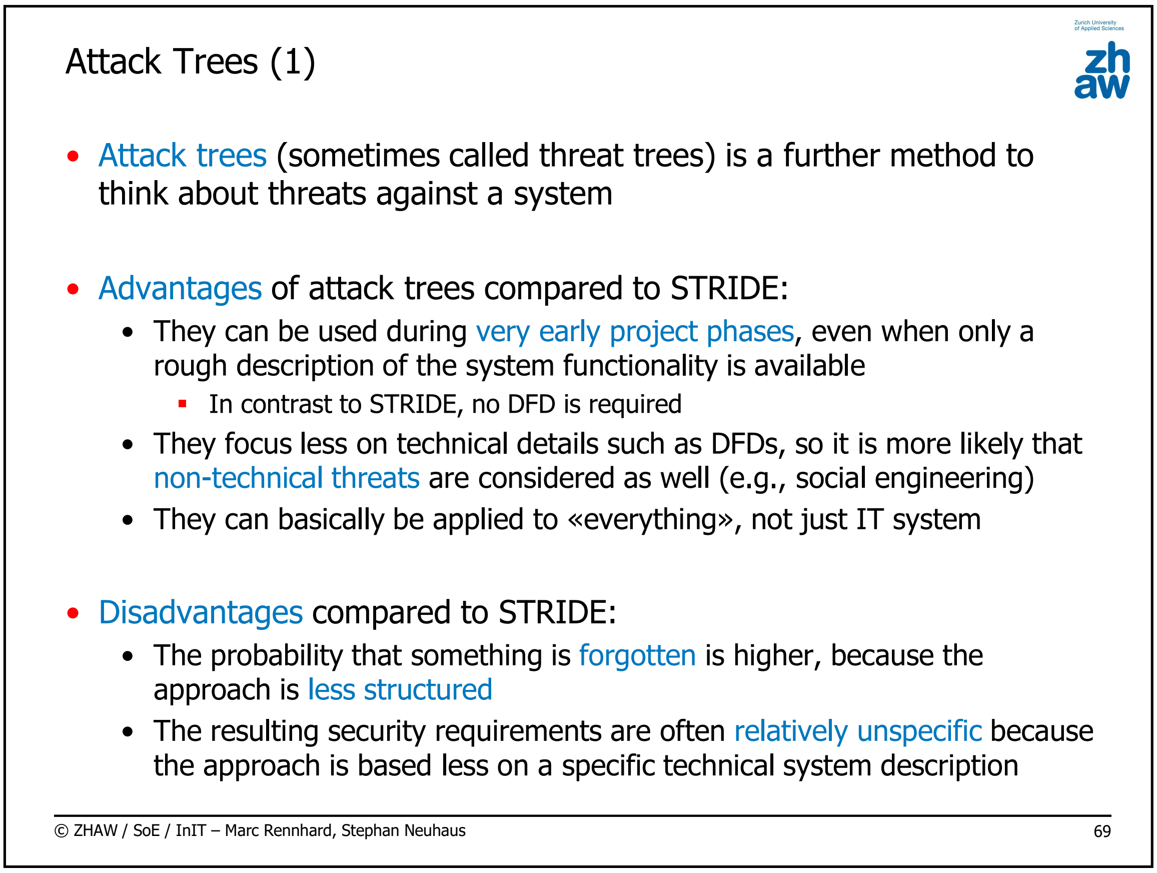
Beyond security requirements that protect from attacks, there may also be security requirements that are needed because of external regulations, e.g., to be compliant with the PCI-DSS standard if credit cards are stored, to be compliant with standards such as HIPAA in case of medical systems that involve patient data, etc.

Even the best security requirements engineering / threat modeling process can never prove a system is secure

- But you can significantly increase the likelihood that the system does not contain major security design flaws

Appendix

Attack Trees



Attack Trees (1)

- Attack trees (sometimes called threat trees) is a further method to think about threats against a system

- Advantages of attack trees compared to STRIDE:
  - They can be used during very early project phases, even when only a rough description of the system functionality is available
    - In contrast to STRIDE, no DFD is required
  - They focus less on technical details such as DFDs, so it is more likely that non-technical threats are considered as well (e.g., social engineering)
  - They can basically be applied to «everything», not just IT system

- Disadvantages compared to STRIDE:
  - The probability that something is forgotten is higher, because the approach is less structured
  - The resulting security requirements are often relatively unspecific because the approach is based less on a specific technical system description

Attack Trees

The idea of attack trees was published in *Bruce Schneier, Dr. Dobb's Journal, December 1999*

The online version is available here: *https://www.schneier.com/academic/archives/1999/12/attack_trees.html*

Some of the examples we present here were directly taken from this paper.

Sometimes, attack trees are also called “threat trees”.

Attack Trees (2)

- So what should be used, STRIDE or Attack Trees?
  - It depends...

- When performing security requirements engineering / threat modeling as an integrated part of software development, STRIDE should be the primary choice
  - STRIDE it is optimized to be used in the context of software development
  - It allows specifying security requirements that are truly targeted at the system under development

- But as general method to identify threats / attacks against any (IT) system, attack trees are well suited
  - Attack trees are also useful at the beginning of software development to define a list of threats / attacks that can be applied to DFDs later

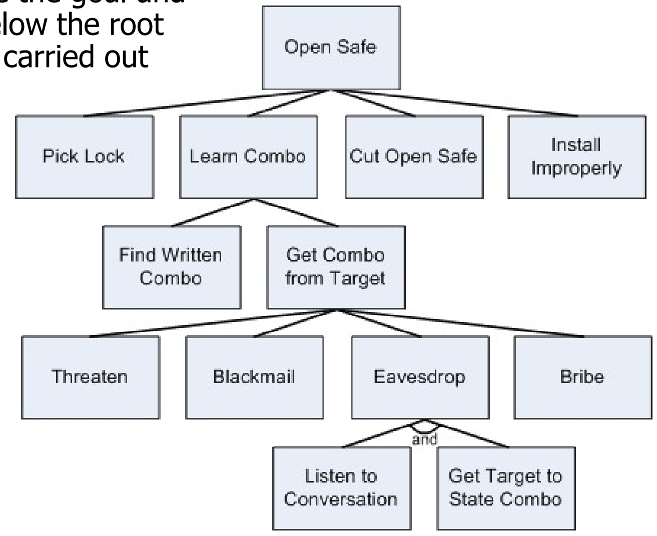
Attack Trees (3)

- To use attack trees, start by defining high-level attack goals
  - Steal credit cards, make system unavailable, open safe,...
  - For each attack goal, an attack tree is constructed
- To construct the tree, think about different ways to achieve this goal
  - E.g., «stealing credit cards» can be achieved by «reading credit cards from database», by «reading credit cards when transmitted over the network», by «getting credit card from user via social engineering»,...
  - The different ways are illustrated in a tree where the goal is used as the root node
- Once a tree has been completed, it can be used in various ways
  - Simply to get an overview of different attacks to achieve a specific goal → can be used to identify vulnerabilities and define security requirements
  - By analyzing the nodes and paths, one can determine, e.g., the cheapest or the easiest way to achieve the attack goal

Attack Trees – Opening a Safe

As an example, we construct an attack tree to open a safe
→ *Open Safe* is the overall attack goal and is used as the root node

- Think about ways to achieve the goal and write them down directly below the root node → attacks that can be carried out to achieve the overall goal
- Each attack can then be split further to describe variants to carry out this attack until a reasonable level of detail is reached
- In the end, only the leaf nodes correspond to attacks that are «directly» executed by the attacker, as they include all attacks up to the root



Basic Idea of Attack Trees

The basic idea of an attack tree is to think about different ways to achieve an attack goal and write down these ways in a tree until you have reached a reasonable level of detail.

Attack Tree to Open a Safe

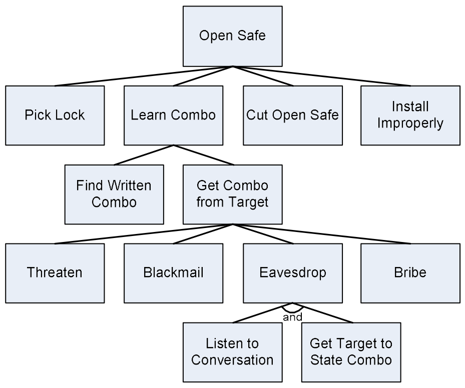
The goal is opening the safe. To open the safe, attackers can pick the lock, learn the combination, cut open the safe, or install the safe improperly so that they can easily open it later. To learn the combination, they either have to find the combination written down or get the combination from the safe owner. And so on. Each node becomes an attack variant, and children of that node are different ways to execute that attack. Of course, this is just a sample attack tree, and an incomplete one. How many other attacks can you think of that would achieve the goal?

Only Leaf Nodes correspond to Attacks that must be executed

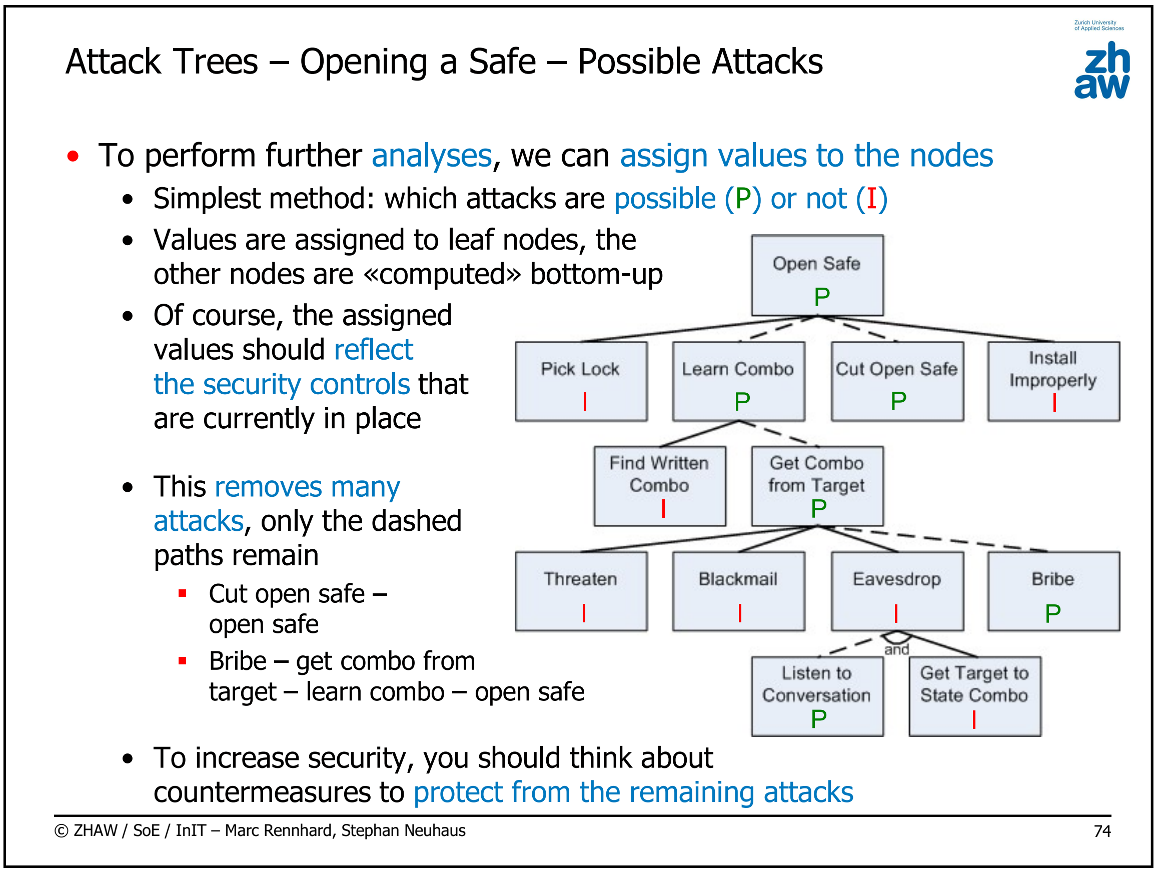
At the end, only the leaf nodes are attacks that are «directly» executed by the attacker as they include all attacks up to the root. So, for example, if the attacker manages to successfully bribe someone who knows the combination (so the attacker «solves» leaf node «Bribe», then node «Get Combo from Target» is also solved, which solves node «Learn Combo», which solves the overall goal «Open Safe».

Attack Trees – AND / OR Nodes

- Note that there are AND nodes and OR nodes
  - Usually, only AND nodes are marked and everything that isn't an AND node is an OR node
- OR nodes are alternatives
  - There are four ways to open a safe (pick lock, learn combo, cut open safe and install improperly...) but only one is necessary to achieve the goal

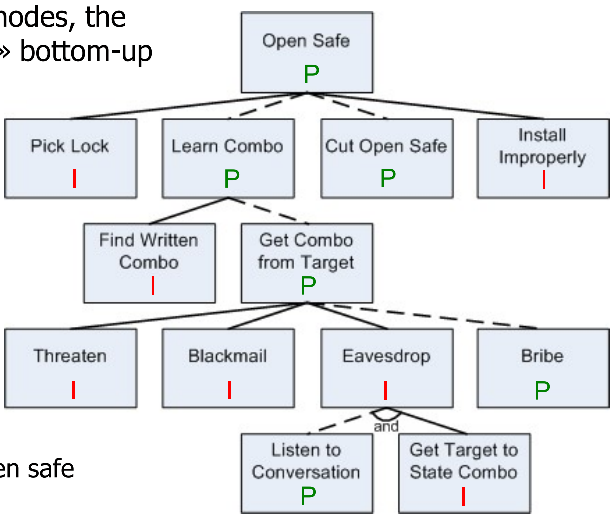


- AND nodes mean that multiple problems must be solved
  - To eavesdrop on someone speaking out the safe combination, attackers have to listen to the conversation AND to get that person to speak out the combination



Attack Trees – Opening a Safe – Possible Attacks

- To perform further analyses, we can assign values to the nodes
  - Simplest method: which attacks are possible (P) or not (I)
  - Values are assigned to leaf nodes, the other nodes are «computed» bottom-up
  - Of course, the assigned values should reflect the security controls that are currently in place
  - This removes many attacks, only the dashed paths remain
    - Cut open safe – open safe
    - Bribe – get combo from target – learn combo – open safe
  - To increase security, you should think about countermeasures to protect from the remaining attacks



Possible and Impossible Attacks

Of course, you must decide for yourself what is possible and what not. In the case above, the values could be determined as follows:

- We have instructed our personnel never to state he combo, so «Get Target to State Combo» gets an I
- We have instructed our personnel to never write down the combo, so «Find Written Combo» gets an I
- It may be well possible our personnel can be bribed (if the amount of money is large enough), so «Bribe» gets a P
- And so on...

Values are Computed Bottom-Up

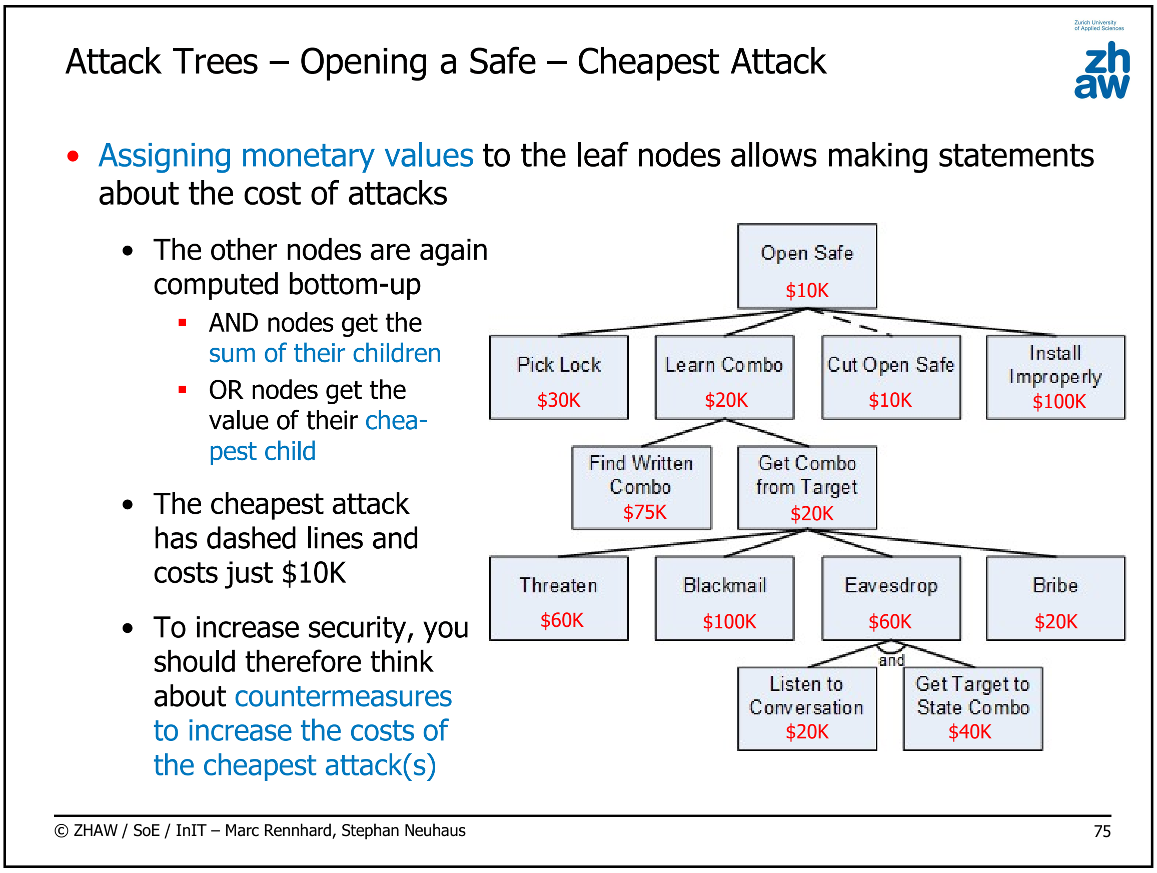
For instance:

- «Listen to Conversation» gets assigned a P and «Get Target to State Combo» an I. As a result, «Eavesdrop» becomes I, as it would only be possible (P) if both subnodes were rated P (because it's an AND node).
- «Threaten», «Blackmail» and «Eavesdrop» are rated I and «Bribe» is rated P. As a result, «Get Combo from Target» is rated P as at least one of the subnodes is rated P (because it's an OR-node).

Other Boolean Values

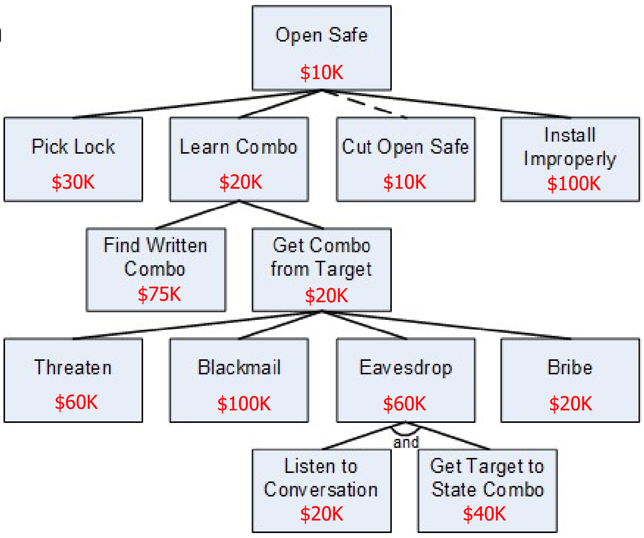
Assigning other boolean values is possible

- easy versus difficult
- expensive versus inexpensive
- intrusive versus nonintrusive
- special equipment required versus no special equipment
- ...



Attack Trees – Opening a Safe – Cheapest Attack

- Assigning monetary values to the leaf nodes allows making statements about the cost of attacks
  - The other nodes are again computed bottom-up
    - AND nodes get the sum of their children
    - OR nodes get the value of their chea-pest child
  - The cheapest attack has dashed lines and costs just $10K
  - To increase security, you should therefore think about countermeasures to increase the costs of the cheapest attack(s)



Monetary Values

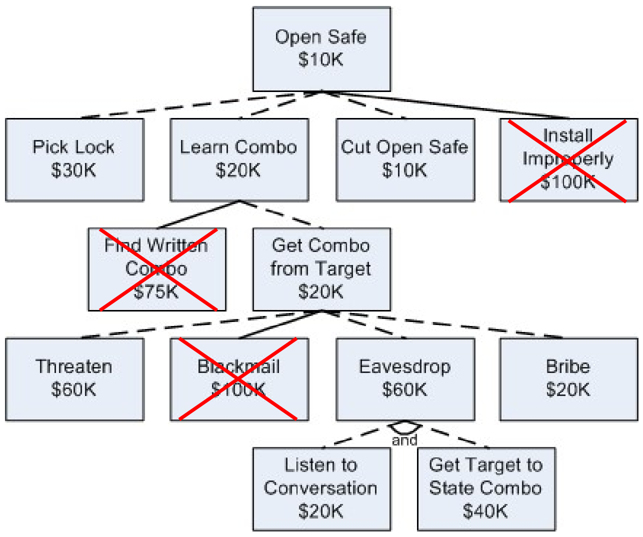
Note that the actual numbers are completely fictional as this is just an example.

Computing Bottom-Up

- «Listen to Conversation» costs 20K and «Get Target to State Combo» costs 40K. Therefore, «Eavesdrop» costs 60K because it's an AND-node (gets the sum of all children).
- «Threaten» costs 60K, «Blackmail» costs 100K, «Eavesdrop» costs 60K and «Bribe» costs 20K. Therefore, «Get Combo from Target» costs 20K because it's an OR-node (gets the value of the «cheapest» child).

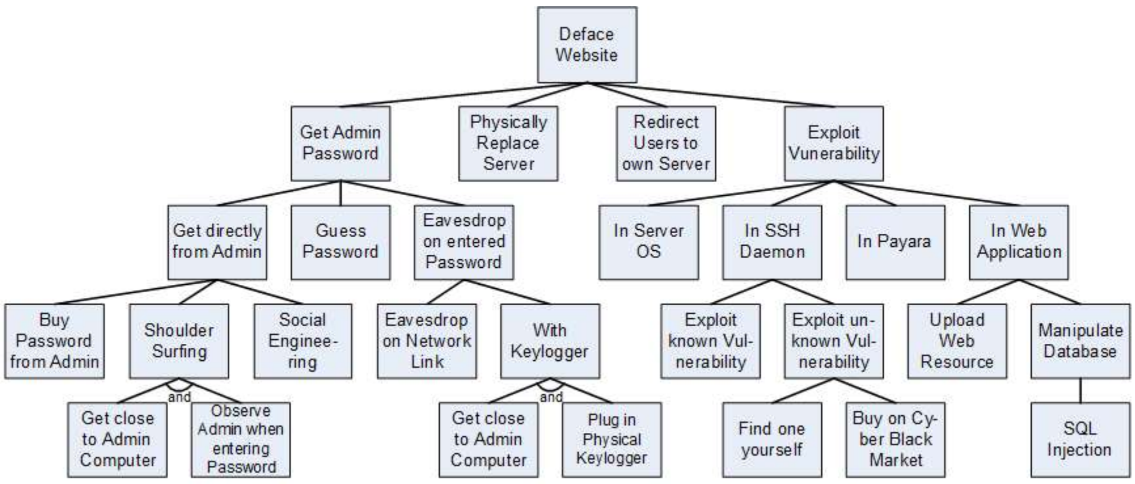
Attack Trees – Opening a Safe – Attacks cheaper than $70K

- Monetary values can also be used to determine all attacks that are, e.g., cheaper than $70K
  - If we assume that there $70K in the safe, the attacker probably won't invest more in an attack
  - One simply removes the nodes with costs of $70K or higher, which eliminates the corresponding attacks
  - Again, further counter-measures should help making the attacks that are cheaper than $70K more expensive to improve overall security



The University Library Application – Attack Tree (1)

- In the following, we consider an example (incomplete) Attack Tree for the University Library Application for the attack goal *Deface Website*



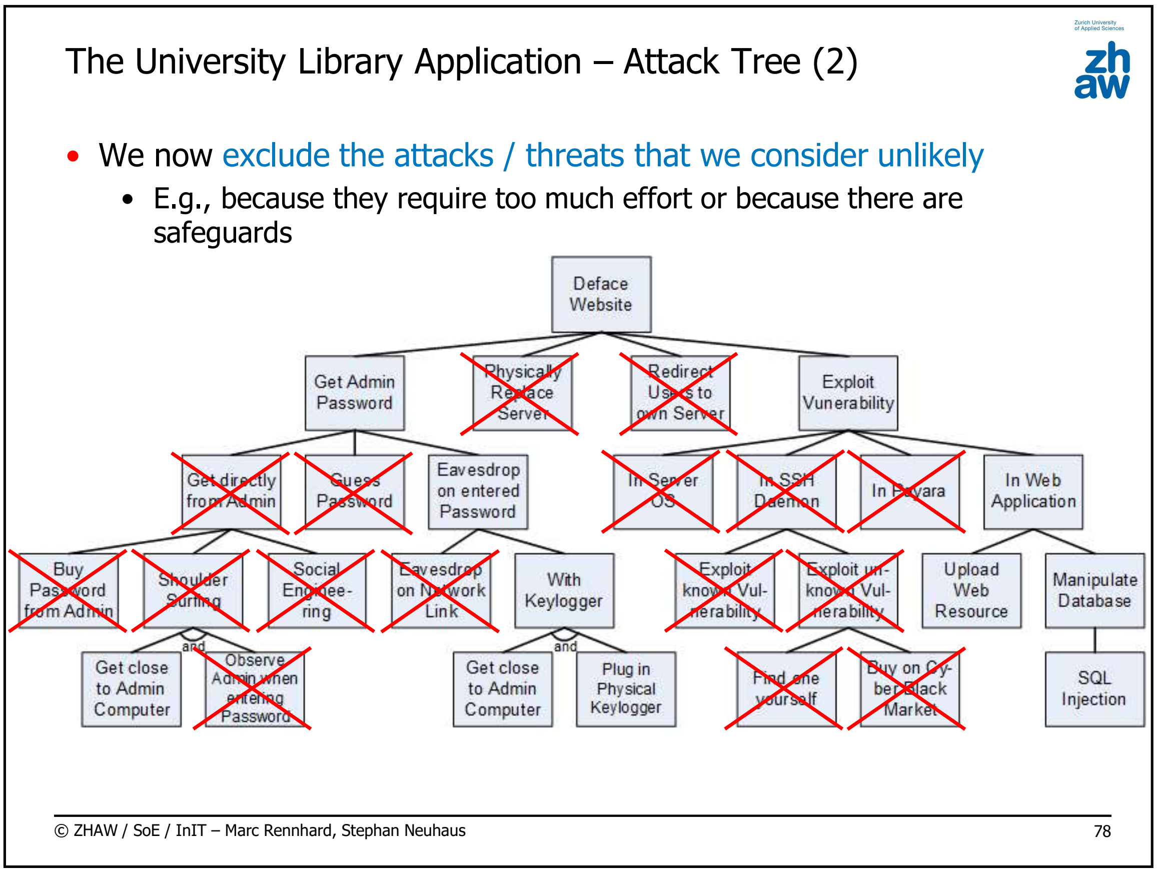
Deface Website
Get Admin Password
Physically Replace Server
Redirect Users to own Server
Exploit Vunerability
Get directly from Admin
Guess Password
Eavesdrop on entered Password
In Server OS
In SSH Daemon
In Payara
In Web Application
Buy Password from Admin
Shoulder Surfing
Social Enginee-ring
Eavesdrop on Network Link
With Keylogger
Exploit known Vul-nerability
Exploit un-known Vul-nerability
Upload Web Resource
Manipulate Database
and
Get close to Admin Computer
Observe Admin when entering Password
and
Get close to Admin Computer
Plug in Physical Keylogger
Find one yourself
Buy on Cy-ber Black Market
SQL Injection

Constructing the Attack Tree

This is done as explained in the example with the Safe:

- Identify the attack goal and use it as the root node: «Deface Website».
- Identify different ways to achieve this goal and write them below the root node: «Get Admin Password», «Physically Replace Server», «Redirect Users to own Server» and «Exploit Vulnerability». There are certainly more ways, but these for ways are considered «good enough» in this example.
- For each of the four nodes, think about ways to achieve them. For instance, «Get Admin Password» can be achieved by getting the password directly from the admin, by guessing the password or by eavesdropping when the admin enters the password (and again, there are certainly more ways).
- This can be refined further until the attack tree has achieved a reasonable level of detail that is suitable for subsequent analyses.

Note that for space restrictions, nodes «Physically Replace Server» and «Redirect Users to own Server» do not have any child nodes. But in reality, they would of course have some child nodes.



The University Library Application – Attack Tree (2)

- We now exclude the attacks / threats that we consider unlikely
  - E.g., because they require too much effort or because there are safeguards

Deface Website
Get Admin Password
~~Physically Replace Server~~
~~Redirect Users to own Server~~
Exploit Vunerability
~~Get directly from Admin~~
~~Guess Password~~
Eavesdrop on entered Password
~~In Server OS~~
~~In SSH Daemon~~
~~In Payara~~
In Web Application
~~Buy Password from Admin~~
~~Shoulder Surfing~~
~~Social Enginee-ring~~
~~Eavesdrop on Network Link~~
With Keylogger
~~Exploit known Vul-nerability~~
~~Exploit un-known Vul-nerability~~
Upload Web Resource
Manipulate Database
and
Get close to Admin Computer
~~Observe Admin when entering Password~~
and
Get close to Admin Computer
Plug in Physical Keylogger
~~Find one yourself~~
~~Buy on Cy-ber Black Market~~
SQL Injection

Excluding Attacks

Which attacks / threats you can exclude is of course highly dependent of the security controls that are planned/installed in the system. In this example, we removed the attacks for the following reasons:

- The admins have security clearance “medium”, so we trust them not to “sell a password”.
- Observing the admin when entering the password is very difficult, as the passwords never show up on the screen and watching the keyboard itself (to get the pressed keys) is very difficult and would require the attacker to be very close to the computer. We therefore consider this attack unlikely to be executed successfully.
- The admins are well-trained and we doubt social engineering attacks will succeed.
- Password guessing is not an option as we assume that admin passwords require a certain password strength.
- Eavesdropping on the communication links will unlikely succeed, as passwords are never transmitted in the clear.
- Physically replacing a server is an unlikely threat as the server is in a well-protected room and all access to it is logged by guards.
- Redirecting users to other server could be done via DNS spoofing, but to reach many users (that access the wrong server), the DNS entries of the university server would have to be spoofed, and the servers are well protected, so we do not identify this as a threat.
- Exploiting a known vulnerability is unlikely, as our administrators continuously monitor security bulletins and security patches are applied regularly.
- Finding an unknown vulnerability requires too much effort and is therefore an unlikely threat.
- Buying an unknown vulnerability/exploit on the cyber black market costs too much considering the value of the attack.
- For the same reasons, exploiting vulnerabilities in the server OS or Payara is an unlikely threat.

The University Library Application – Attack Tree (3)

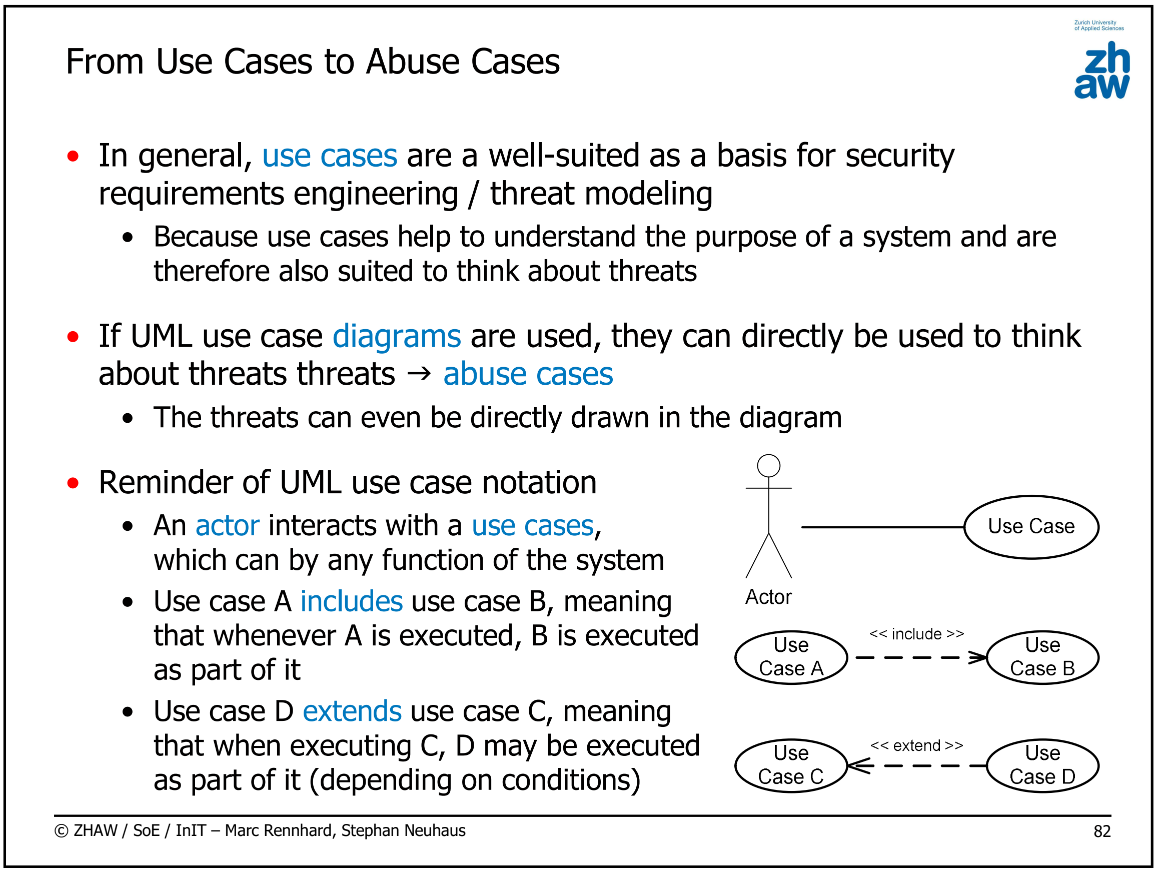
One can now think about mitigating the remaining threats:

- Getting admin password via installing a physical keylogger is considered a threat
  - Because workplaces of admins are also accessible by other employees and students and plugging in a keylogger is easily possible
  - Can be prevented by preventing «Get close to Admin Computer» or «Plug in Physical Keylogger»
  - Prevention options (→ new security requirements):
    - Control physical access to admin computers (may not be practical)
    - Attach keyboard such that keyloggers cannot easily be plugged in
- Exploit vulnerability in the web application (SQL Injection, Upload Web Resource,...)
  - Because no corresponding security requirements have been defined
  - Can be prevented by defining appropriate security requirements (prepared statements, input validation, no write access rights to web resources,...)

Attack Tree Summary: Construction and Usage

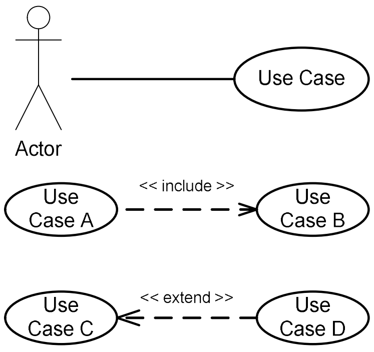
- Identify the overall attack goals
  - Each goal forms a separate tree, but they might share subtrees and nodes
  - E.g., a subtree «compromise user password» is likely part of many trees
- To construct the tree, identify attacks or attack steps that can be used to achieve the overall goal and add them to the tree
  - For each attack, this can be repeated until you have reached your desired level of detail
  - Ideally, construct the tree in a team or let it review by somebody else to identify a broad spectrum of attacks
- Once constructed, the attack trees can be used to perform analyses
  - E.g., remove the attacks for which you have adequate security requirements defined / or security controls in place
  - The remaining attacks should be analyzed in detail and appropriate countermeasures (security requirements) should be selected
  - Attack trees also allow to compare attacks to calculate the most likely (cheapest, easiest,…) attacks against your system

Abuse Cases



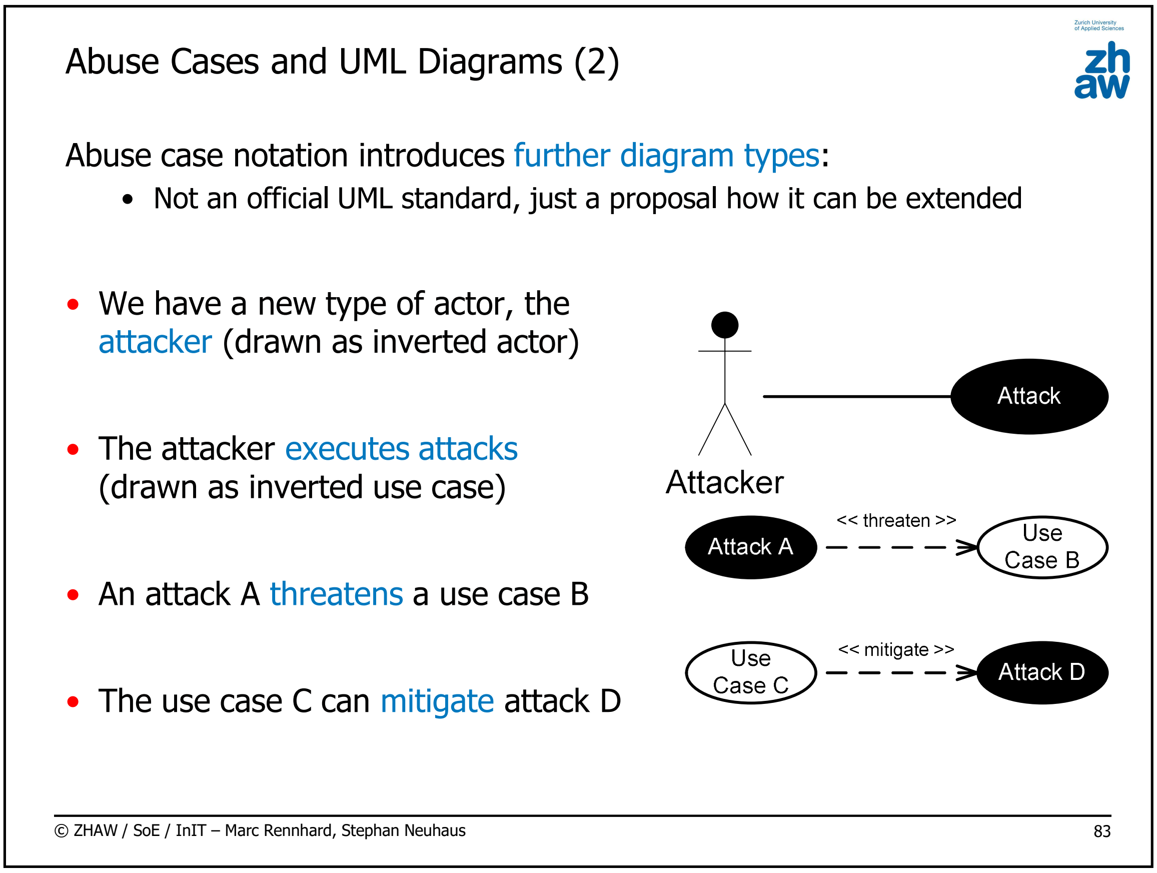
From Use Cases to Abuse Cases

- In general, use cases are a well-suited as a basis for security requirements engineering / threat modeling
  - Because use cases help to understand the purpose of a system and are therefore also suited to think about threats
- If UML use case diagrams are used, they can directly be used to think about threats threats → abuse cases
  - The threats can even be directly drawn in the diagram
- Reminder of UML use case notation
  - An actor interacts with a use cases, which can by any function of the system
  - Use case A includes use case B, meaning that whenever A is executed, B is executed as part of it
  - Use case D extends use case C, meaning that when executing C, D may be executed as part of it (depending on conditions)



UML

A very good basic introduction to UML can be found here (in German): *http://www.highscore.de/uml*

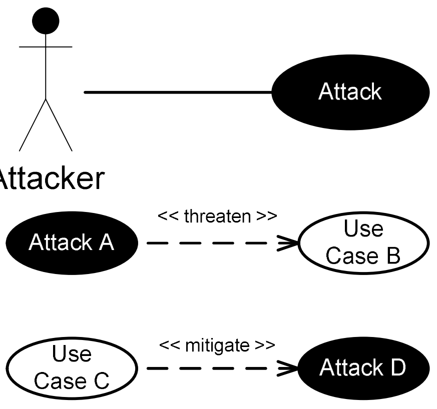


Abuse Cases and UML Diagrams (2)

Abuse case notation introduces further diagram types:

- Not an official UML standard, just a proposal how it can be extended

- We have a new type of actor, the attacker (drawn as inverted actor)
- The attacker executes attacks (drawn as inverted use case)
- An attack A threatens a use case B
- The use case C can mitigate attack D



Abuse Cases and UML Diagrams

This has been discussed in several papers, including the following:

Guttorm Sindre and Andreas Opdahl, Eliciting Security Requirements by Misuse Cases. In Proceedings of 37th International Conference on Technology of Object-Oriented Languages and Systems, 2000, TOOLS-Pacific 2000.

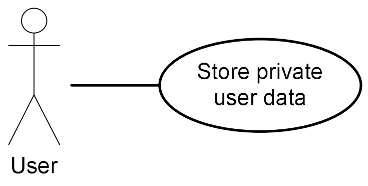
Joshua Pauli and Dianxiang Xu, Threat-Driven Architectural Design of Secure Information Systems. Avaliable at http://cs.ndsu.edu/~dxu/publications/pauli-xu-ICEIS05.pdf

Lilian Rostad, An extended misuse case notation: Including vulnerabilities and the insider threat. Available at http://www.di.unipi.it/REFSQ06/Papers/01%20Rostad.pdf

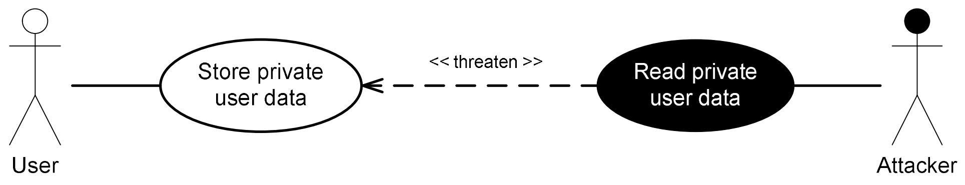
The notation we are using here is closely related to the one in the paper by Lilian Rostad.

Abuse Cases – Simple Example (1)

- Consider a system that stores private user data, e.g., a health care application
- To keep it very simply, we consider one use case: store private user data

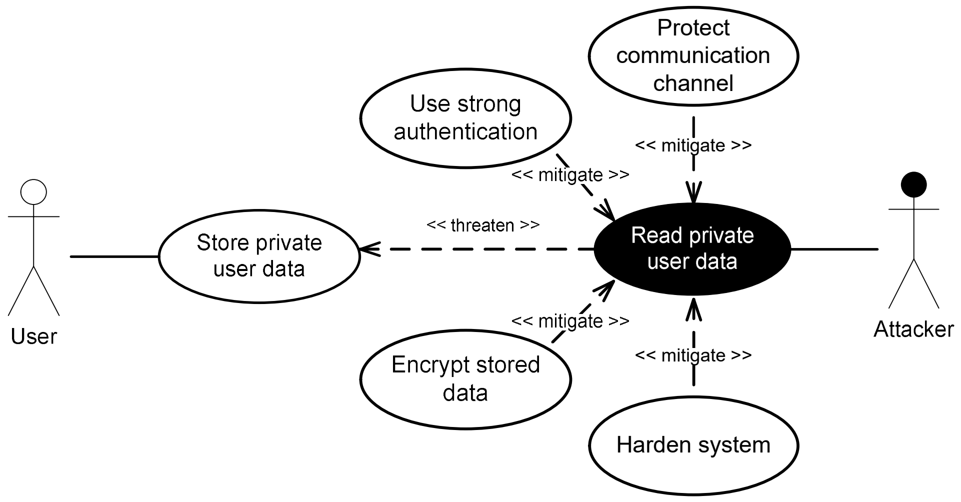


- If this use case is used to think about attacks, then we can identify an attacker that wants to read the private user data
  - So we have an attack (abuse case) «read private user data» that threatens the stored private user data



Abuse Cases – A Simple Example (2)

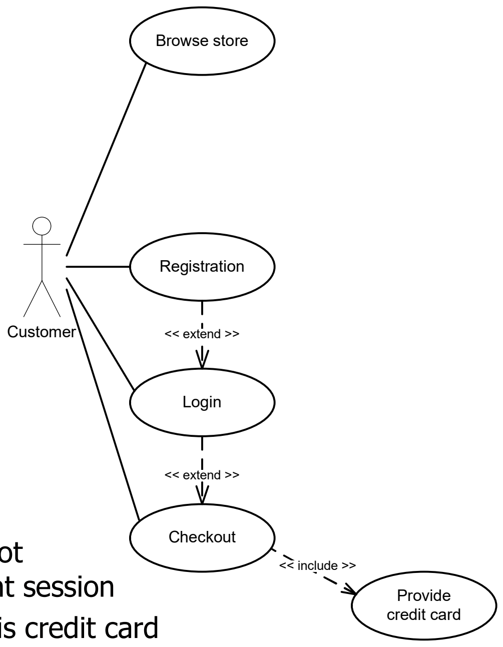
- Based on this scenario, we can now think about security requirements (drawn as use cases) that can mitigate the attack



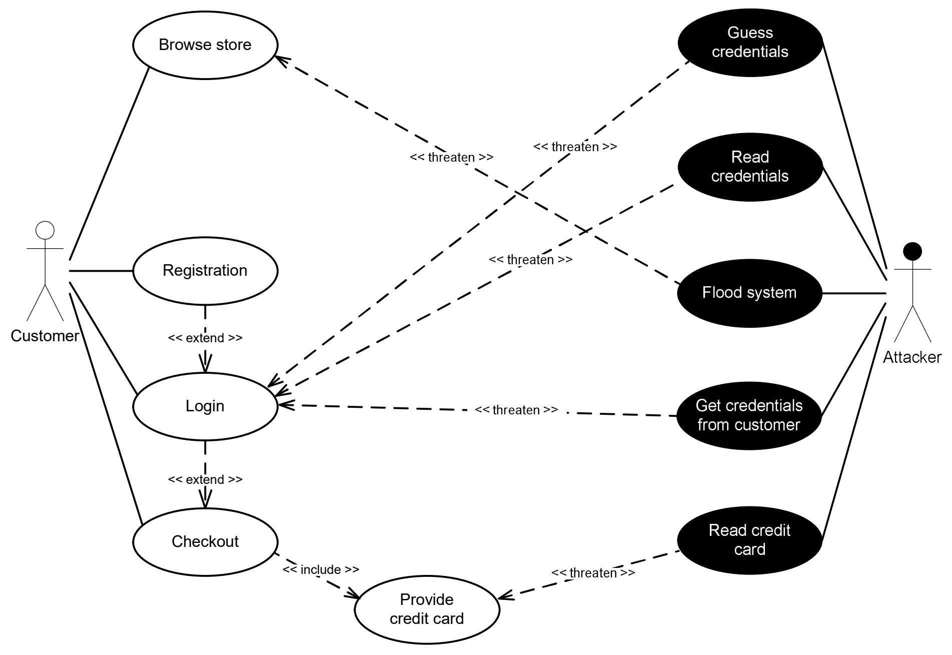
A more Elaborate Example – Basic Use Cases

Consider an e-shop with some basic functions (use cases)

- Browsing the store
- Customer registration
- Login
  - May require registration if customer has not registered before
- Checkout
  - May require login if customer has not performed a login during the current session
  - Requires the customer to provide his credit card



A more Elaborate Example – Abuse Cases



© ZHAW / SoE / InIT – Marc Rennhard, Stephan Neuhaus 87

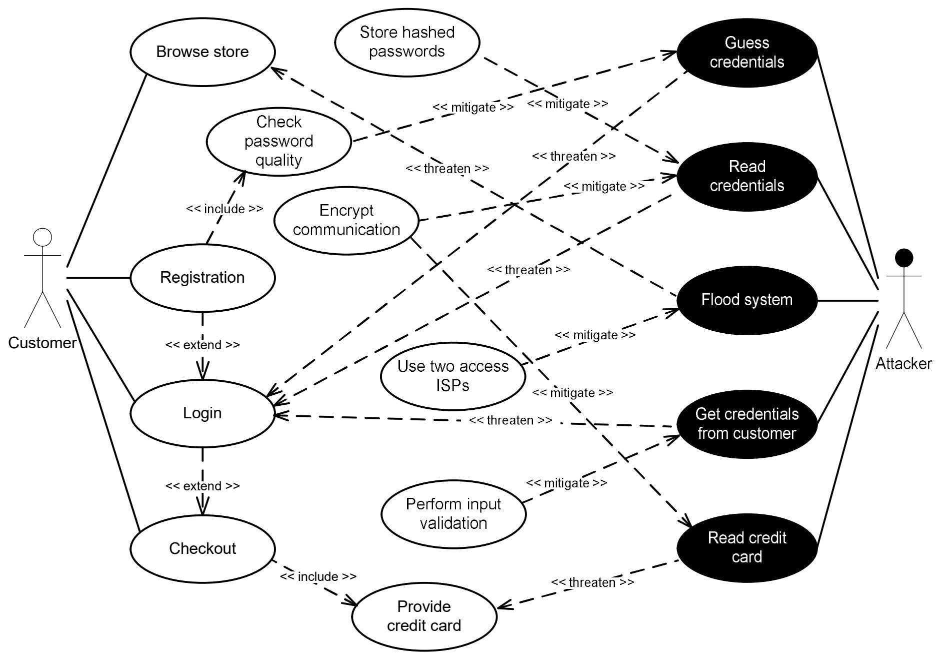
Abuse Cases

- Read credentials: E.g., via SQL injection, reading unencrypted network data, or by stealing the server disk
- Get credentials from customer: E.g., via XSS or phishing

Completeness

Note that not all associations have been drawn into the diagram above for clarity. For instance, “flood system” does not only target the use case “browse store”, but will most likely affect the availability of the entire store and should therefore affect virtually all use cases.

A more Elaborate Example – Security Requirements / Mitigation Use Cases



Mitigation Use Cases

• Perform input validation: To mitigate "get credentials from customer" via XSS

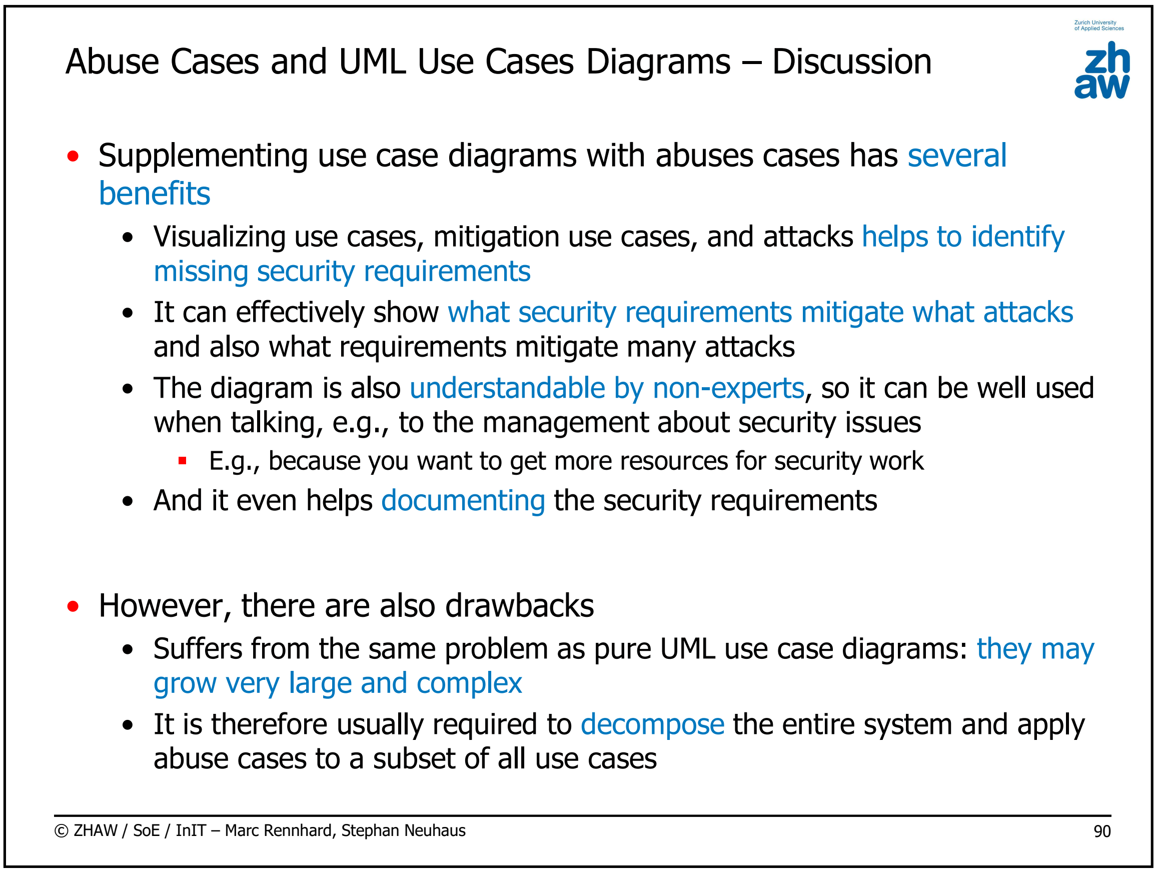
Of course, many more mitigation use cases are possible and also reasonable, e.g., data sanitation against XSS, prepared statements against SQLi, 2-factor authentication against phishing etc.

What about Malicious Insiders?

Normal use case actors (e.g., customer, administrator) can also be attackers. In this case, they exist on the actor side and on the attacker side (e.g., malicious administrator).

A more Elaborate Example – Summary

- A UML use case diagram is used as the basis
  - This helps to think about attacks against use cases, which are included in the diagram as abuse cases
  - Which then helps to identify appropriate security requirements (mitigation use cases), which are also included in the diagram
- Not all mitigation use case are noticeable as functions by the users
  - «Password quality check» and «encrypt communication» will be visible
  - «Store hashed passwords», «use two access ISPs», and «perform input validation» typically won't be noticed
- Often, the non-visible mitigation use cases are also less obvious and are more likely to be forgotten during software development
  - Even developers that are not security-aware will likely realize that «Password quality check» and «encrypt communication» should be incorporated into the application
  - But the others are more likely to be forgotten, especially if the developer is not aware of the corresponding threats



Abuse Cases and UML Use Cases Diagrams – Discussion

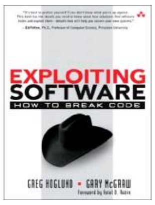
- Supplementing use case diagrams with abuses cases has several benefits
  - Visualizing use cases, mitigation use cases, and attacks helps to identify missing security requirements
  - It can effectively show what security requirements mitigate what attacks and also what requirements mitigate many attacks
  - The diagram is also understandable by non-experts, so it can be well used when talking, e.g., to the management about security issues
    - E.g., because you want to get more resources for security work
  - And it even helps documenting the security requirements
- However, there are also drawbacks
  - Suffers from the same problem as pure UML use case diagrams: they may grow very large and complex
  - It is therefore usually required to decompose the entire system and apply abuse cases to a subset of all use cases

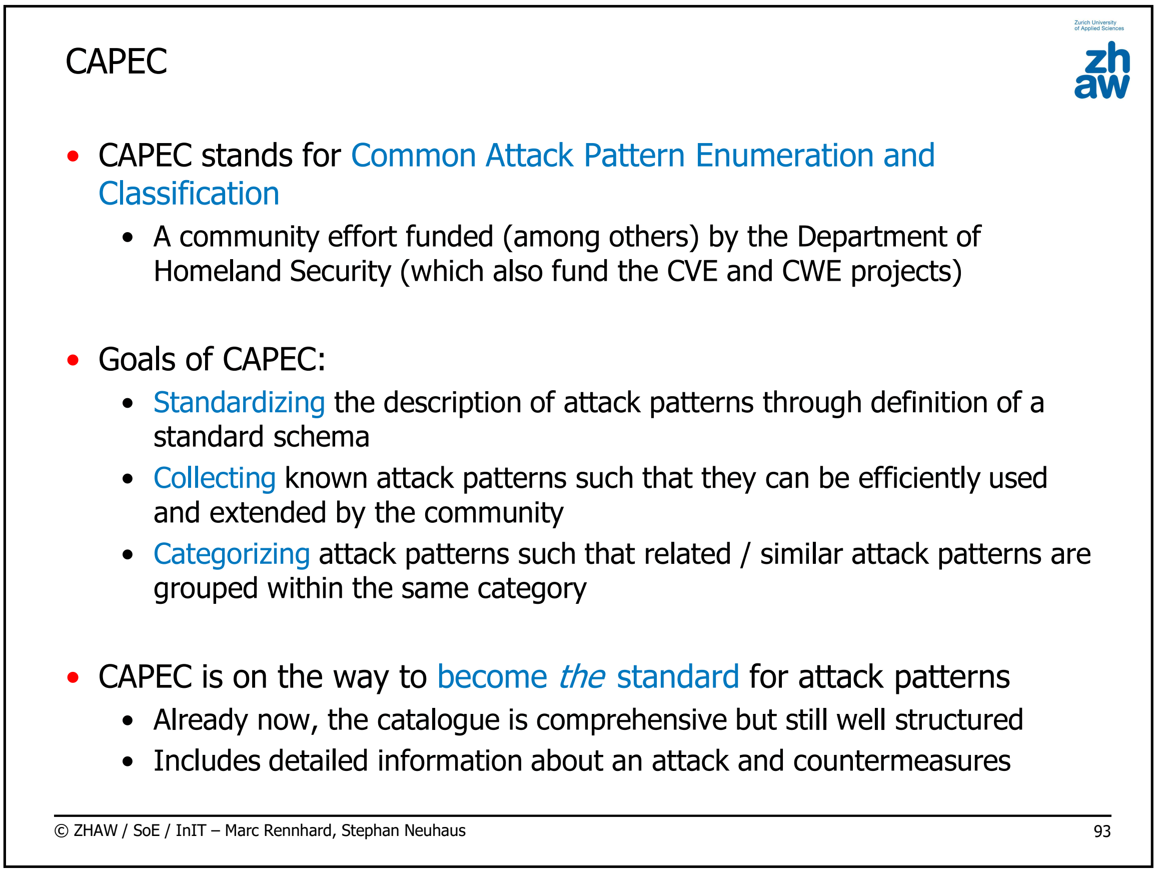
A diagram with use cases and abuse cases is most likely much better understandable by the management than a rather technical Data Flow Diagram.

Attack Patterns

Attack Patterns

- Attack patterns are basically collections of different attacks
- Using such a collection is helpful when thinking about possible attacks
  - They can also be used to support virtually any threat modeling method you employ (e.g., STRIDE, attack trees etc.)
- Different sources of attack patterns exist, e.g.:
  - 48 attack patterns as discussed in the book Exploiting Software
    - One of the first attempts to provide a collection of attacks
    - Very detailed description of various attacks
    - Problem: Defined once and not maintained
  - Common Attack Pattern Enumeration and Classification (CAPEC)
    - Also a collection of attacks, available online
    - Advantage: The collection is continuously maintained and extended





CAPEC

- CAPEC stands for Common Attack Pattern Enumeration and Classification
  - A community effort funded (among others) by the Department of Homeland Security (which also fund the CVE and CWE projects)
- Goals of CAPEC:
  - Standardizing the description of attack patterns through definition of a standard schema
  - Collecting known attack patterns such that they can be efficiently used and extended by the community
  - Categorizing attack patterns such that related / similar attack patterns are grouped within the same category
- CAPEC is on the way to become *the* standard for attack patterns
  - Already now, the catalogue is comprehensive but still well structured
  - Includes detailed information about an attack and countermeasures

CAPEC

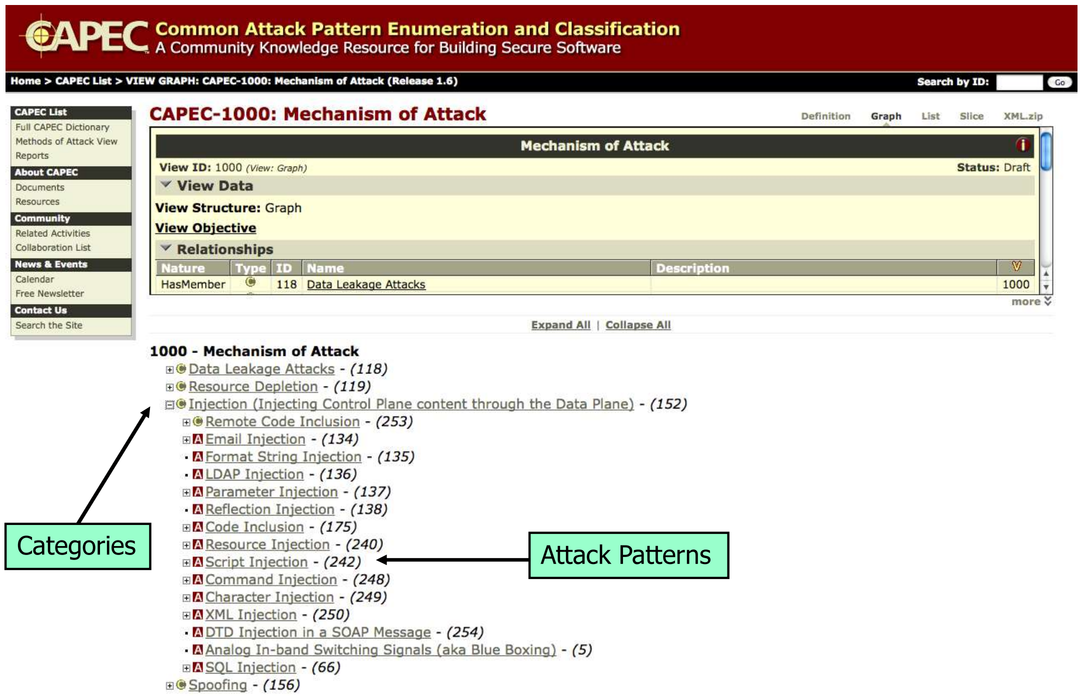
The main web page is *http://capec.mitre.org*

Similar Projects

CVE: Common Vulnerabilities and Exposure*: http://cve.mitre.org*

CWE: Common Weakness Enumeration: *http://cwe.mitre.org*

CAPEC – Overview of Attacks



CAPEC Common Attack Pattern Enumeration and Classification
A Community Knowledge Resource for Building Secure Software

Home > CAPEC List > VIEW GRAPH: CAPEC-1000: Mechanism of Attack (Release 1.6) Search by ID: Go

CAPEC List
Full CAPEC Dictionary
Methods of Attack View
Reports
About CAPEC
Documents
Resources
Community
Related Activities
Collaboration List
News & Events
Calendar
Free Newsletter
Contact Us
Search the Site

CAPEC-1000: Mechanism of Attack

Definition Graph List Slice XML.zip

Mechanism of Attack

View ID: 1000 *(View: Graph)* **Status:** Draft

View Data

View Structure: Graph

View Objective

Relationships

Nature	Type	ID	Name	Description	V
HasMember	●	118	Data Leakage Attacks		1000

more

Expand All | Collapse All

1000 - Mechanism of Attack

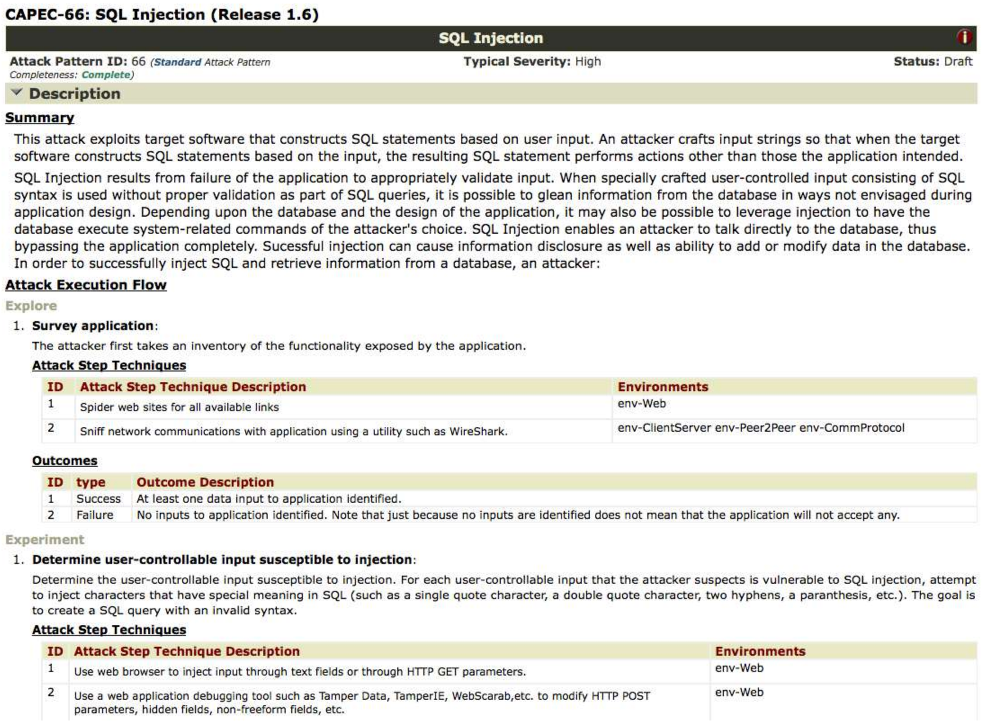
- Data Leakage Attacks - *(118)*
- Resource Depletion - *(119)*
- Injection (Injecting Control Plane content through the Data Plane) - *(152)*
  - Remote Code Inclusion - *(253)*
  - Email Injection - *(134)*
  - Format String Injection - *(135)*
  - LDAP Injection - *(136)*
  - Parameter Injection - *(137)*
  - Reflection Injection - *(138)*
  - Code Inclusion - *(175)*
  - Resource Injection - *(240)*
  - Script Injection - *(242)*
  - Command Injection - *(248)*
  - Character Injection - *(249)*
  - XML Injection - *(250)*
  - DTD Injection in a SOAP Message - *(254)*
  - Analog In-band Switching Signals (aka Blue Boxing) - *(5)*
  - SQL Injection - *(66)*
- Spoofing - *(156)*

CAPEC

The main web page is *http://capec.mitre.org*

The above view is received by clicking *Methods of Attack View* in the left menu.

CAPEC – SQL Injection



CAPEC-66: SQL Injection (Release 1.6)

SQL Injection

Attack Pattern ID: 66 (*Standard Attack Pattern Completeness: Complete*) **Typical Severity:** High **Status:** Draft

Description

Summary

This attack exploits target software that constructs SQL statements based on user input. An attacker crafts input strings so that when the target software constructs SQL statements based on the input, the resulting SQL statement performs actions other than those the application intended.

SQL Injection results from failure of the application to appropriately validate input. When specially crafted user-controlled input consisting of SQL syntax is used without proper validation as part of SQL queries, it is possible to glean information from the database in ways not envisaged during application design. Depending upon the database and the design of the application, it may also be possible to leverage injection to have the database execute system-related commands of the attacker's choice. SQL Injection enables an attacker to talk directly to the database, thus bypassing the application completely. Sucessful injection can cause information disclosure as well as ability to add or modify data in the database. In order to successfully inject SQL and retrieve information from a database, an attacker:

Attack Execution Flow

Explore

1. **Survey application**:

   The attacker first takes an inventory of the functionality exposed by the application.

   Attack Step Techniques

ID	Attack Step Technique Description	Environments
1	Spider web sites for all available links	env-Web
2	Sniff network communications with application using a utility such as WireShark.	env-ClientServer env-Peer2Peer env-CommProtocol

   Outcomes

ID	type	Outcome Description
1	Success	At least one data input to application identified.
2	Failure	No inputs to application identified. Note that just because no inputs are identified does not mean that the application will not accept any.

Experiment

1. **Determine user-controllable input susceptible to injection**:

   Determine the user-controllable input susceptible to injection. For each user-controllable input that the attacker suspects is vulnerable to SQL injection, attempt to inject characters that have special meaning in SQL (such as a single quote character, a double quote character, two hyphens, a paranthesis, etc.). The goal is to create a SQL query with an invalid syntax.

   Attack Step Techniques

ID	Attack Step Technique Description	Environments
1	Use web browser to inject input through text fields or through HTTP GET parameters.	env-Web
2	Use a web application debugging tool such as Tamper Data, TamperIE, WebScarab,etc. to modify HTTP POST parameters, hidden fields, non-freeform fields, etc.	env-Web

CAPEC – Value

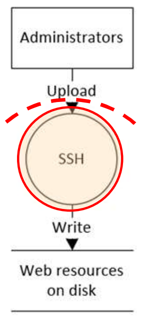
- CAPEC is a valuable resource because it provides detailed information about typical attacks, including
  - A general description of the attack
  - How to execute the attack (attack execution flow)
  - Related vulnerabilities (by CVE numbers)
  - Probing techniques
  - Attacker skill required
  - Consequences of successful exploitation
  - Solutions and mitigations
- This information is valuable during different phases of the software development lifecycle, e.g.
  - During threat modeling to identify attacks, to understand how they work and their potential impact, and to propose appropriate countermeasures
  - During penetration testing to search for vulnerabilities

Zurich University of Applied Sciences

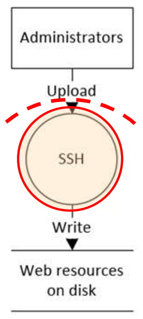
STRIDE Exercise

Exercise based on University Library Web Application (1)

- In the context of the SSH process, do the following:
  - Identify 1 or 2 threats (pick any STRIDE category you want)
  - Determine whether appropriate countermeasures are in place to protect from the threat(s)
  - If this is not the case, you have found a vulnerability and you should propose new reasonable security requirements



Exercise based on University Library Web Application (2)


Administrators
Upload
SSH
Write
Web resources on disk