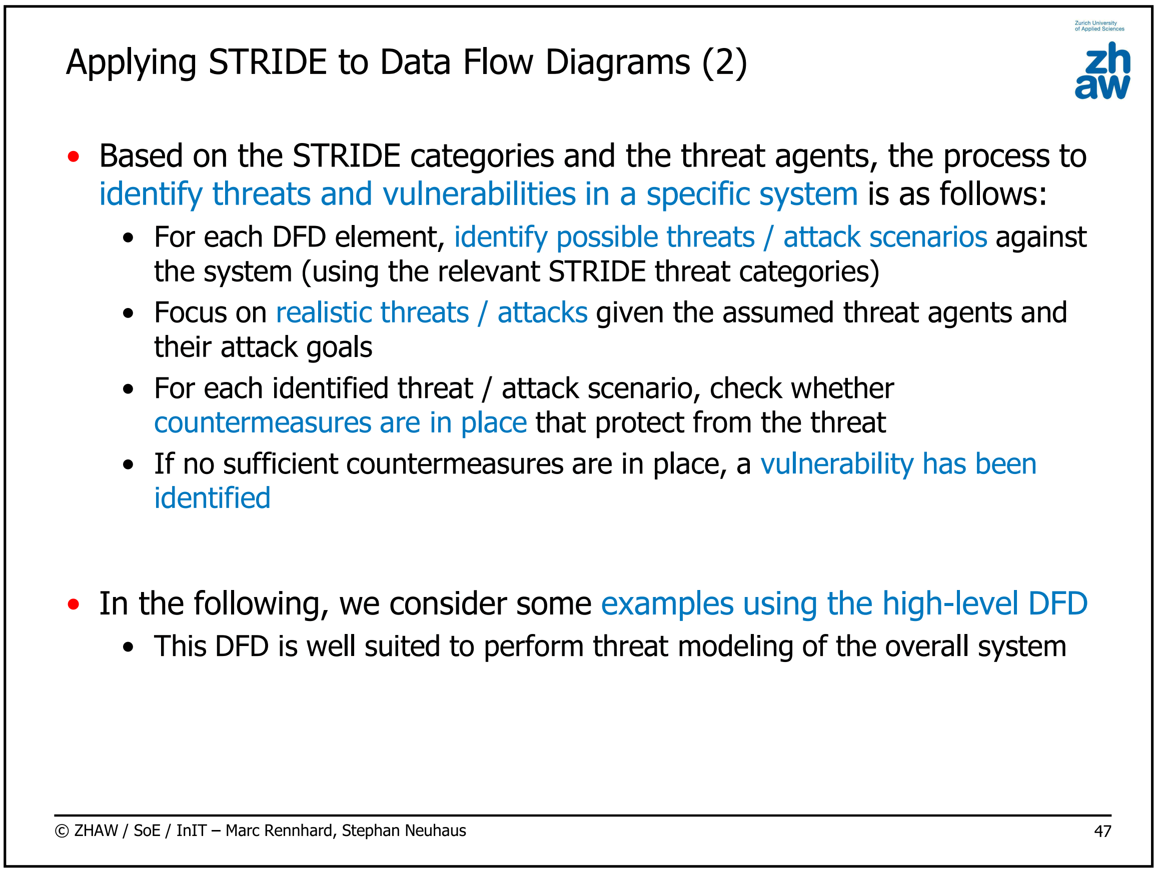
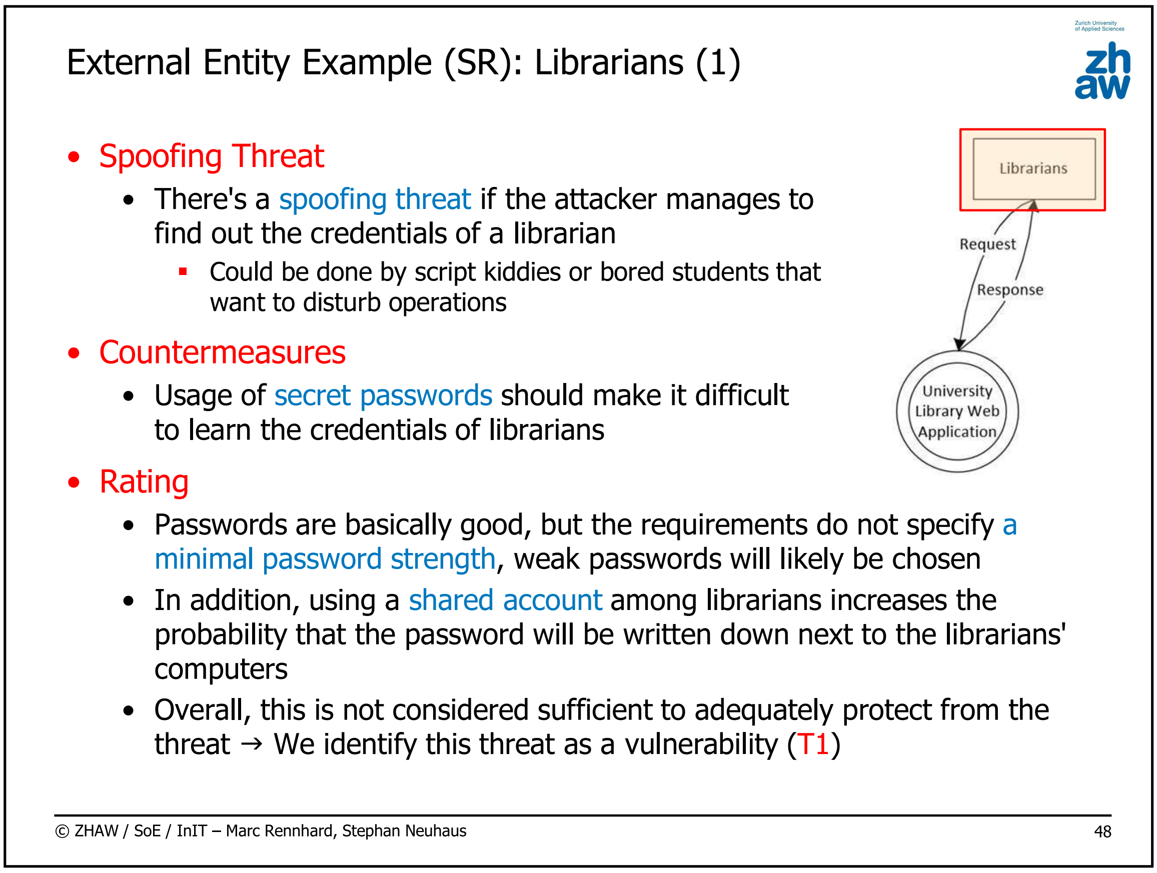

Applying STRIDE to Data Flow Diagrams (2)

- 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, check whether countermeasures are in place that protect from the threat
  - If no sufficient countermeasures are in place, a vulnerability has been identified
- In the following, we consider some examples using the high-level DFD
  - This DFD is well suited to perform threat modeling of the overall system

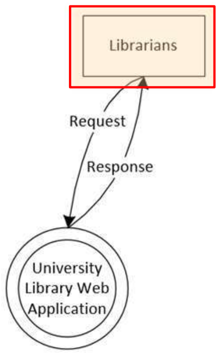
Increase the Efficiency of this Step

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».



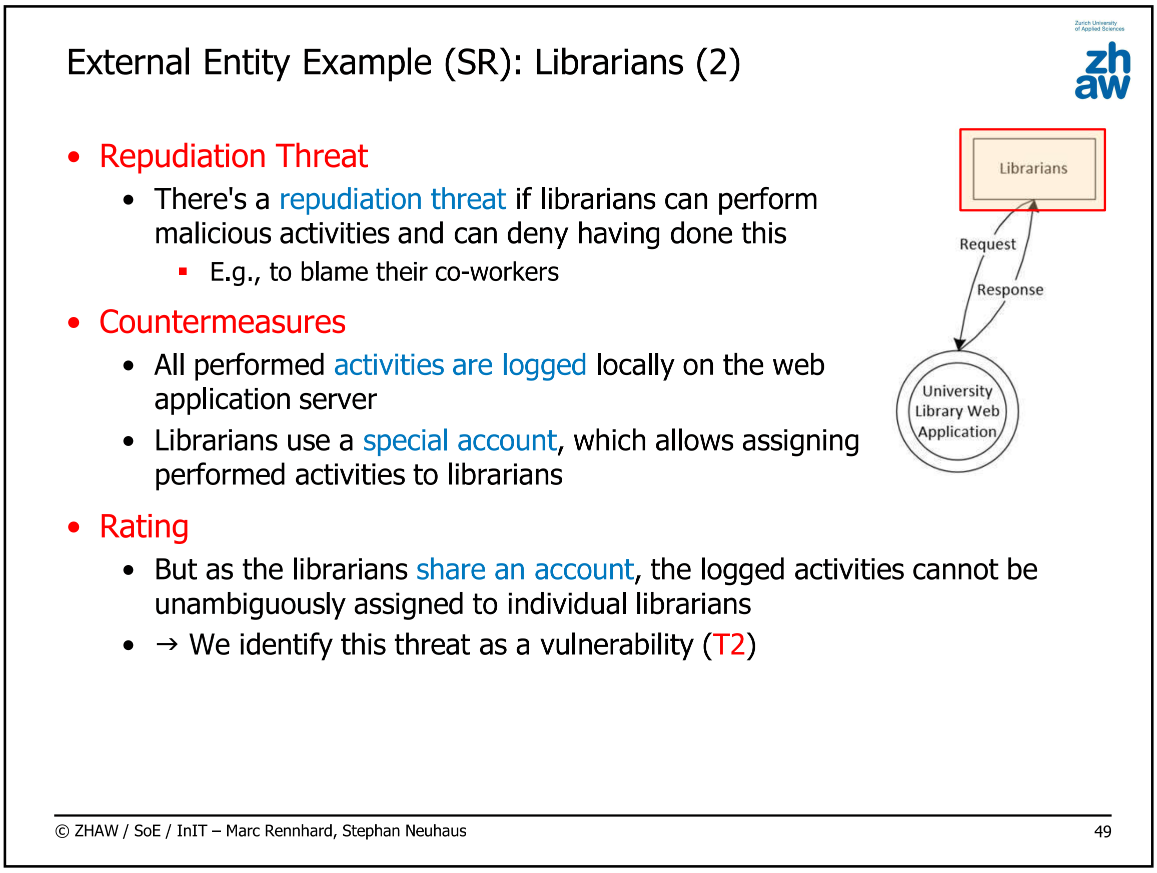
External Entity Example (SR): Librarians (1)

- Spoofing Threat
  - 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
- Countermeasures
  - Usage of secret passwords should make it difficult to learn the credentials of librarians
- 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 (T1)



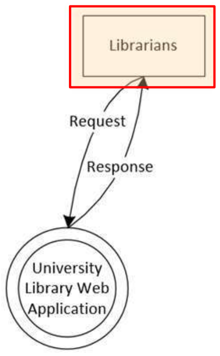
Spoofing Threats

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.



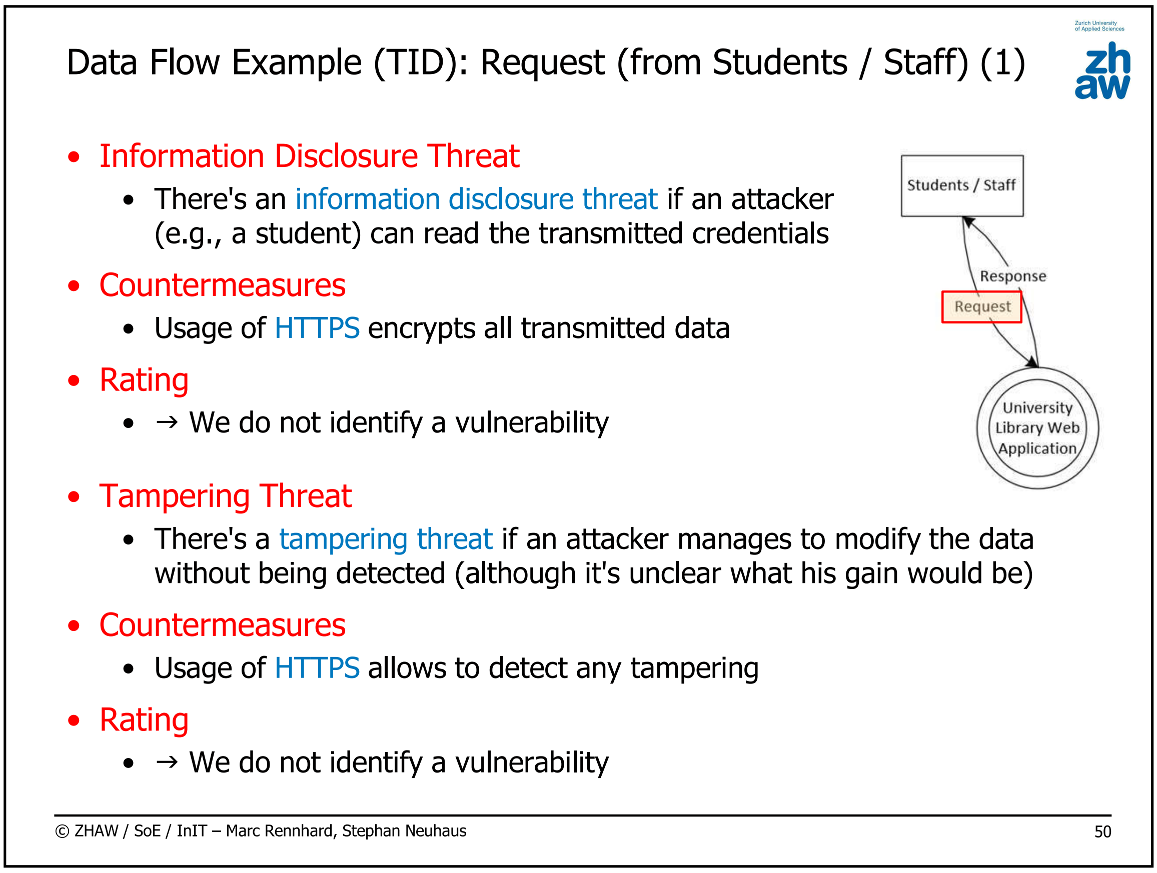
External Entity Example (SR): Librarians (2)

- Repudiation Threat
  - There's a repudiation threat if librarians can perform malicious activities and can deny having done this
    - E.g., to blame their co-workers
- Countermeasures
  - All performed activities are logged locally on the web application server
  - Librarians use a special account, which allows assigning performed activities to librarians
- 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 (T2)



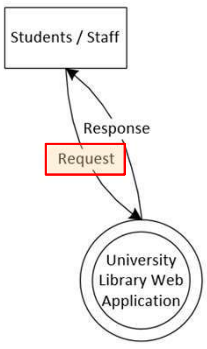
Assets and Vulnerabilities

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)

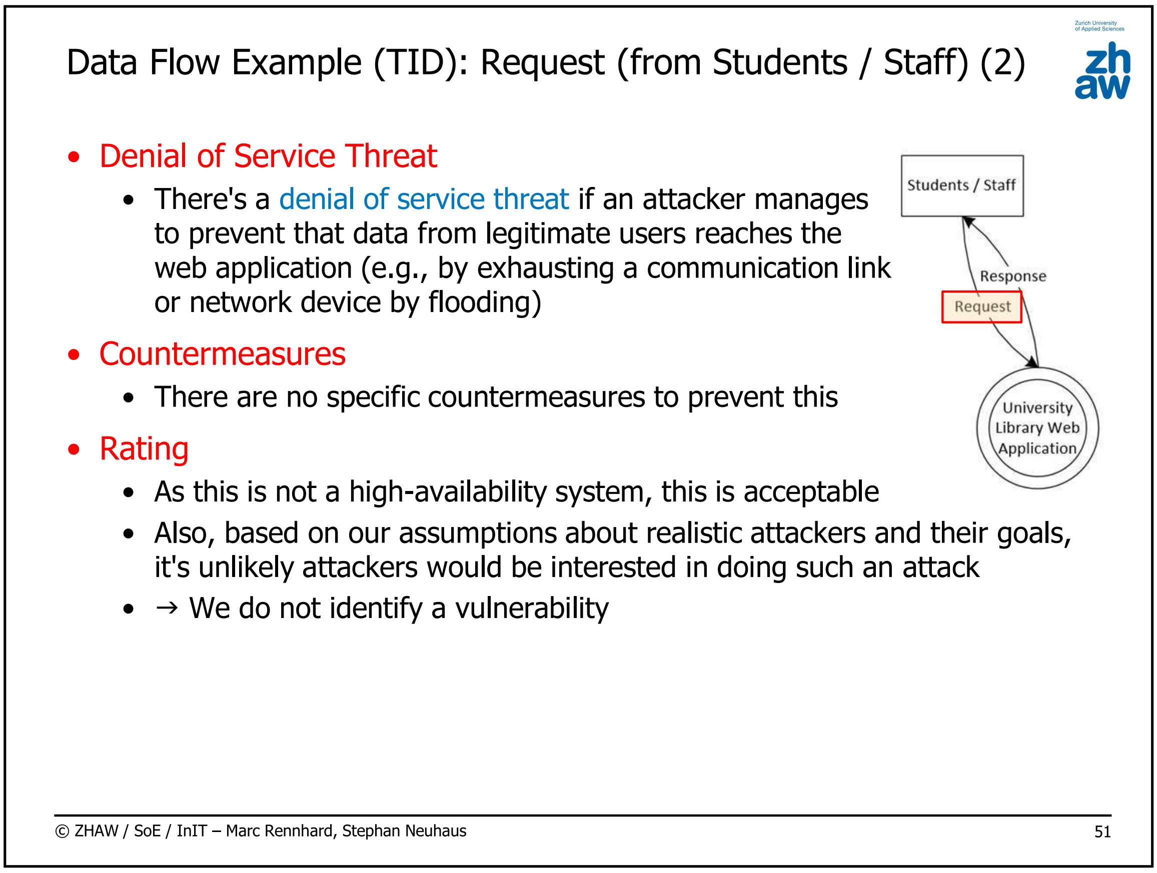
- Information Disclosure Threat
  - There's an information disclosure threat if an attacker (e.g., a student) can read the transmitted credentials
- Countermeasures
  - Usage of HTTPS encrypts all transmitted data
- Rating
  - → We do not identify a vulnerability



- Tampering Threat
  - 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)
- Countermeasures
  - Usage of HTTPS allows to detect any tampering
- Rating
  - → We do not identify a vulnerability

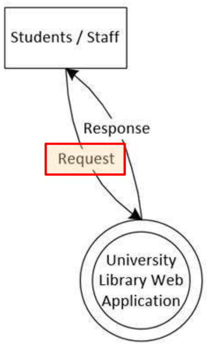
HTTPS

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.



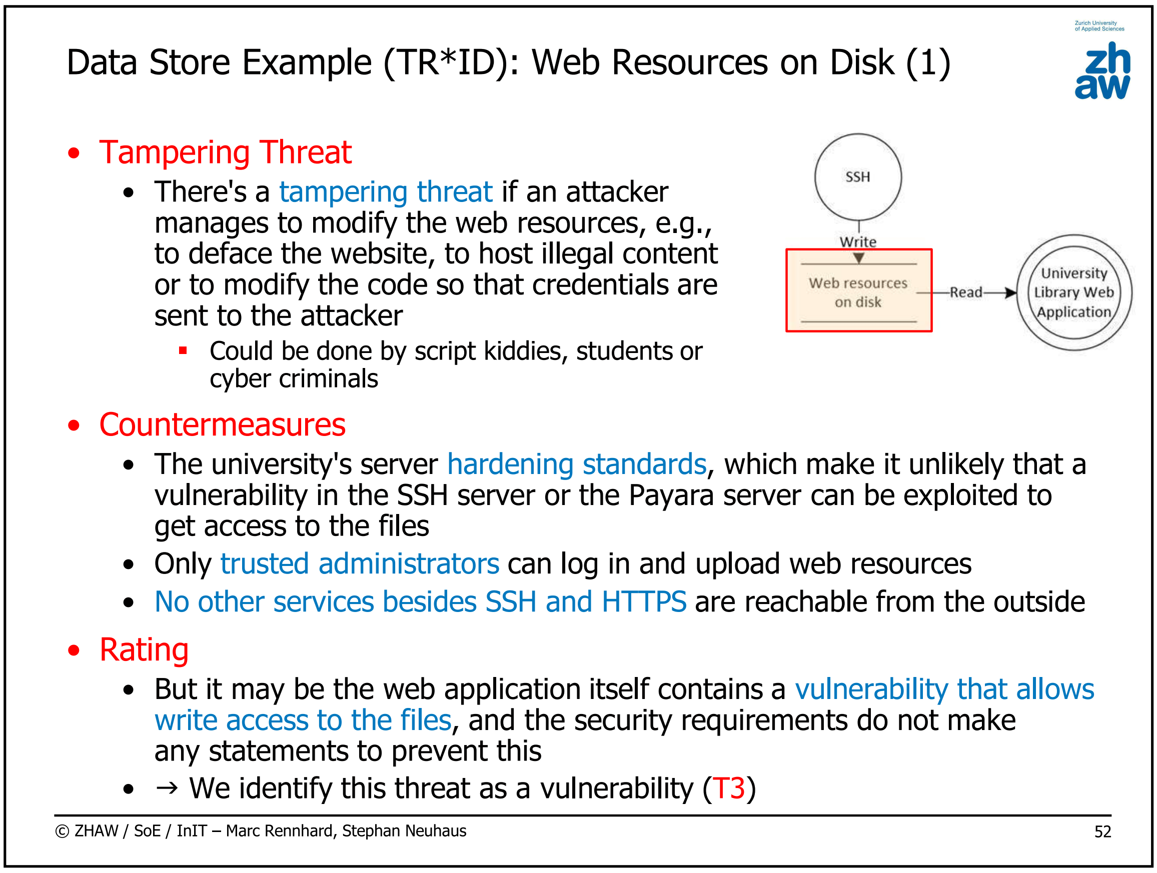
Data Flow Example (TID): Request (from Students / Staff) (2)

- Denial of Service Threat
  - 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)
- Countermeasures
  - There are no specific countermeasures to prevent this
- 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



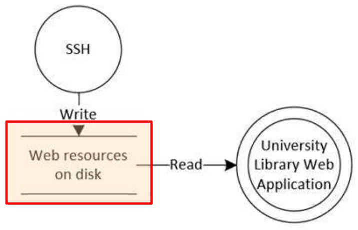
Denial of Service Threat against Data Flows

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
  - 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



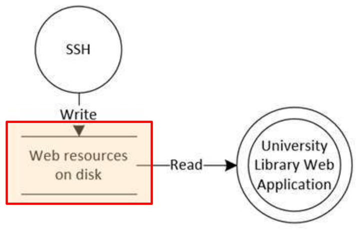
- Countermeasures
  - The university's server hardening standards, which make it unlikely that a vulnerability in the SSH server or the Payara server can be exploited to get access to the files
  - Only trusted administrators can log in and upload web resources
  - No other services besides SSH and HTTPS are reachable from the outside
- 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 (T3)

Repudiation Threats

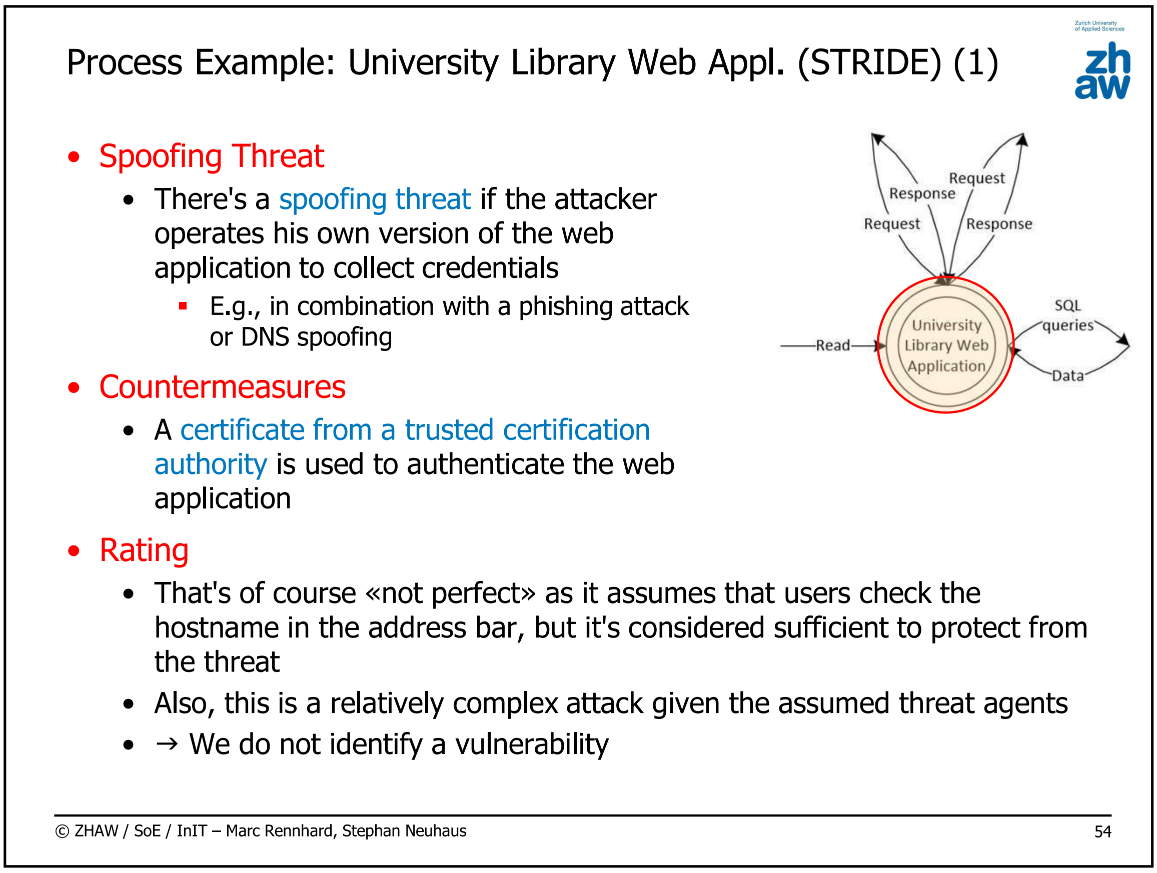
*This data store does not contain logging or audit data, so repudiation threats are no issue.

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

- Information Disclosure Threat
  - 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

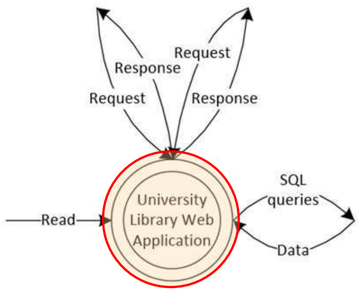


- Countermeasures / Rating
  - → For the same reasons as with tampering, this threat is also identified as a vulnerability (T4)
- Denial of Service Threat
  - There's a denial of service threat if an attacker can exhaust storage
- Countermeasures
  - None
- Rating
  - → We do not identify a vulnerability, as the application does not write to this data store (unlike, e.g., a log file)



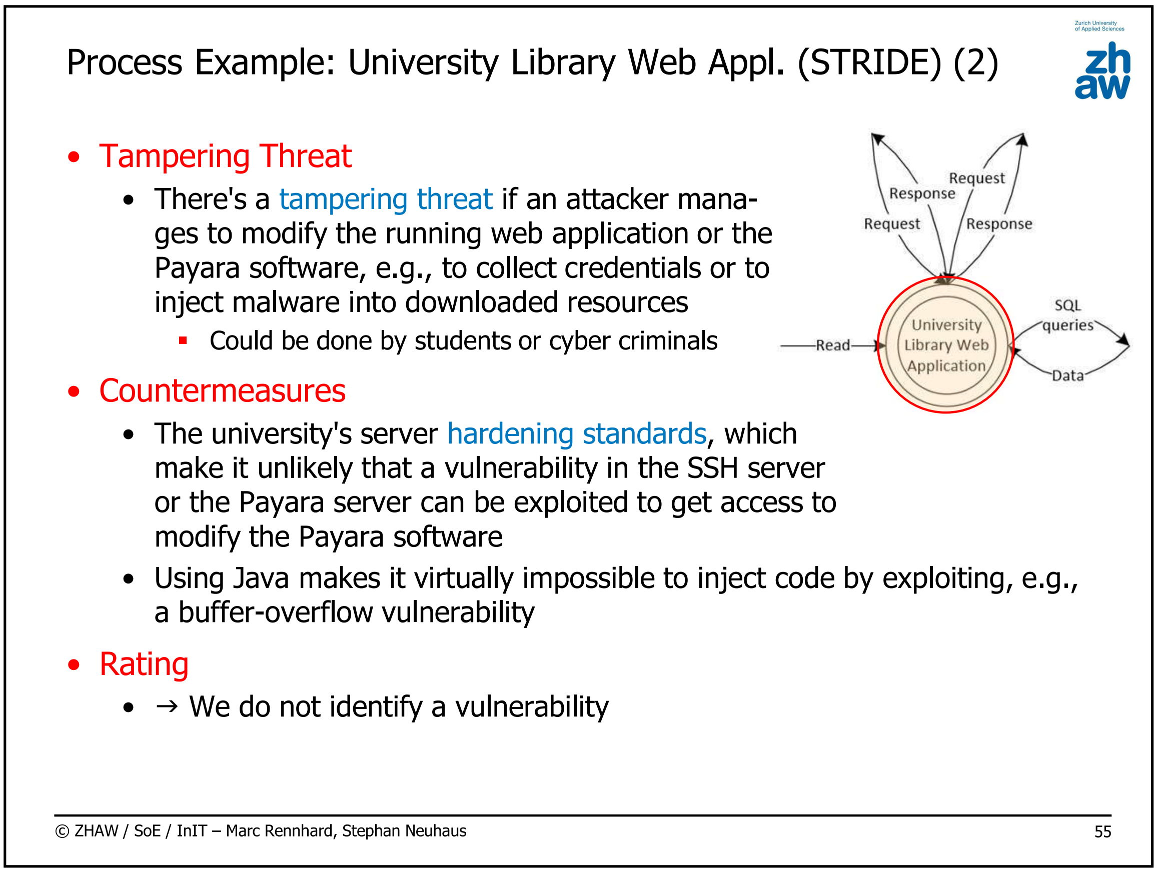
Process Example: University Library Web Appl. (STRIDE) (1)

- Spoofing Threat
  - 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
- Countermeasures
  - A certificate from a trusted certification authority is used to authenticate the web application
- 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



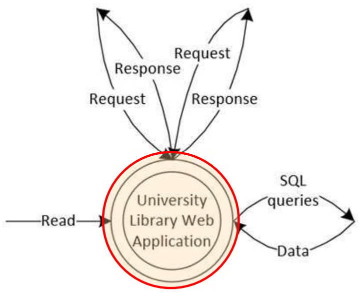
Spoofing the Web Application Server

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.



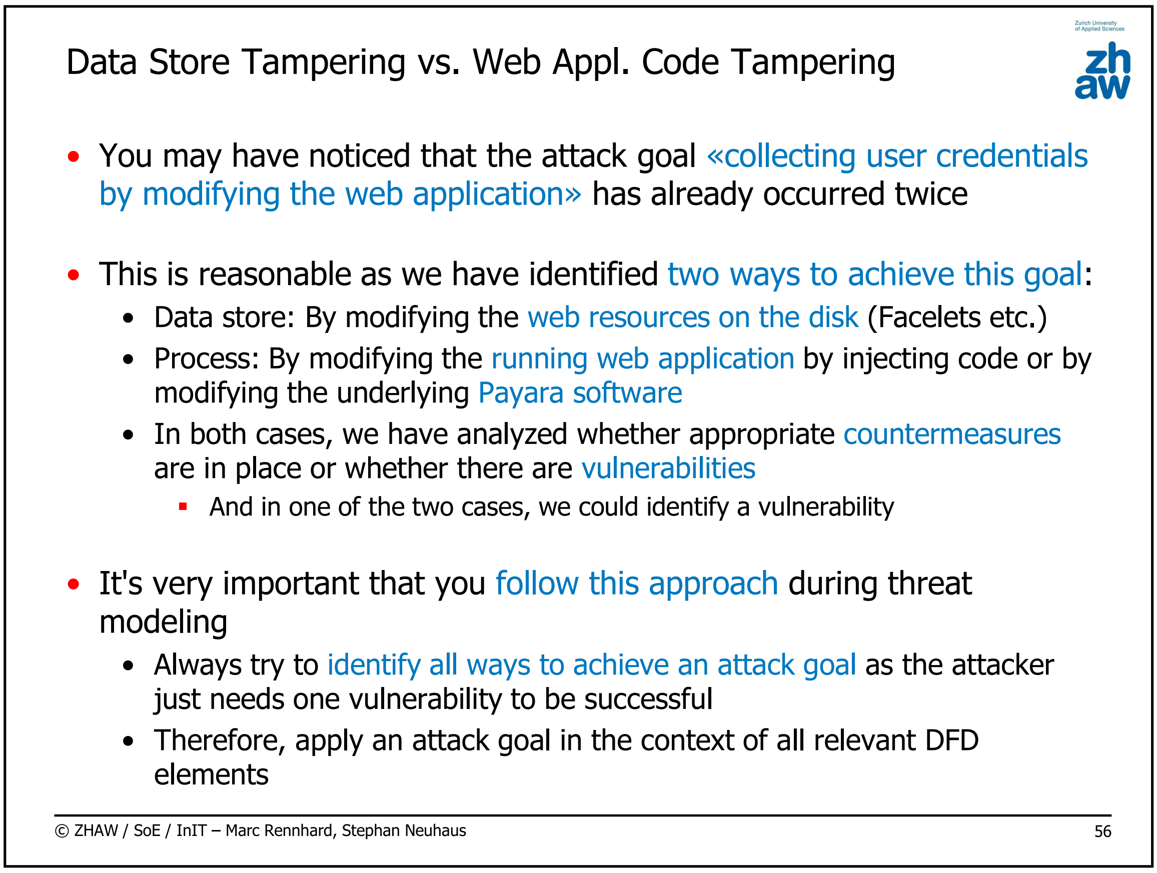
Process Example: University Library Web Appl. (STRIDE) (2)

- Tampering Threat
  - 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
- Countermeasures
  - 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
  - Using Java makes it virtually impossible to inject code by exploiting, e.g., a buffer-overflow vulnerability
- 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.

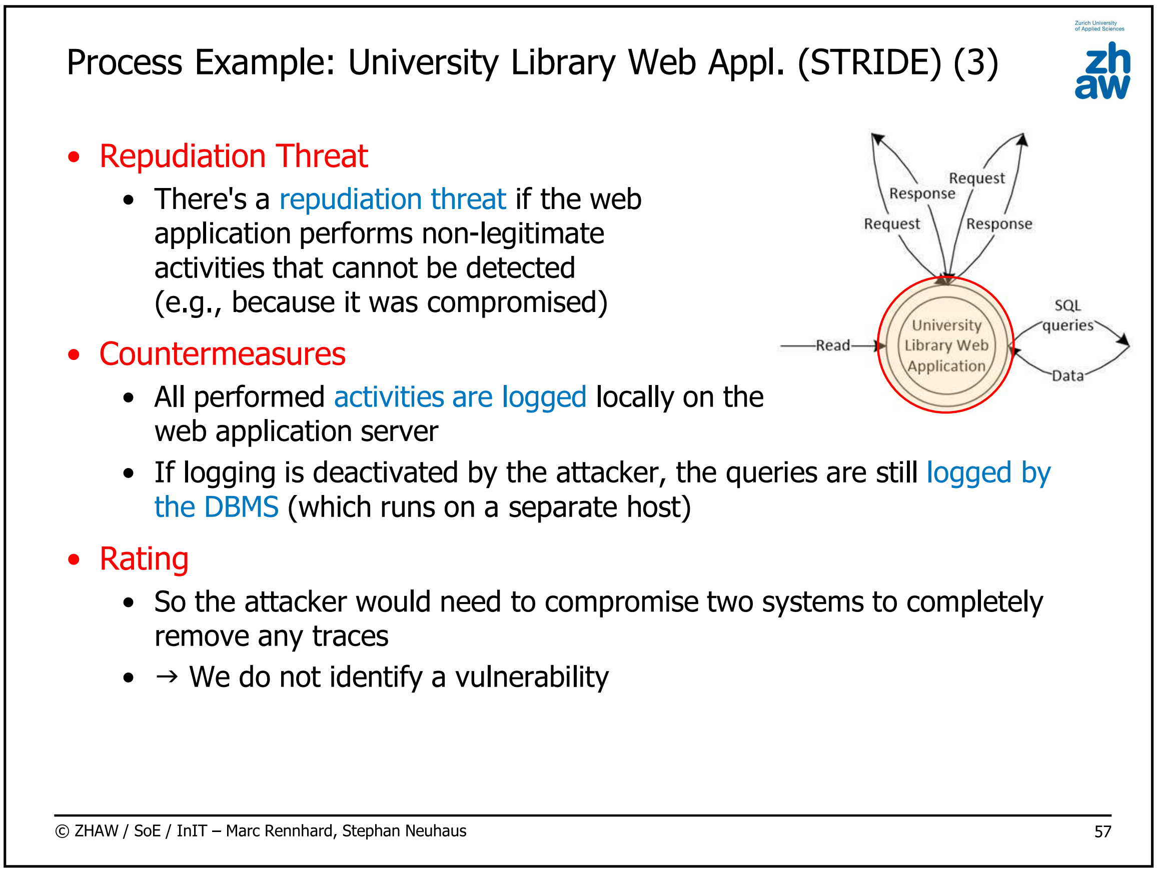


Data Store Tampering vs. Web Appl. Code Tampering

- 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

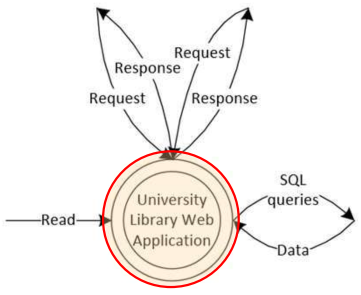
Web Application Code Tampering vs. Data Store Tampering

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

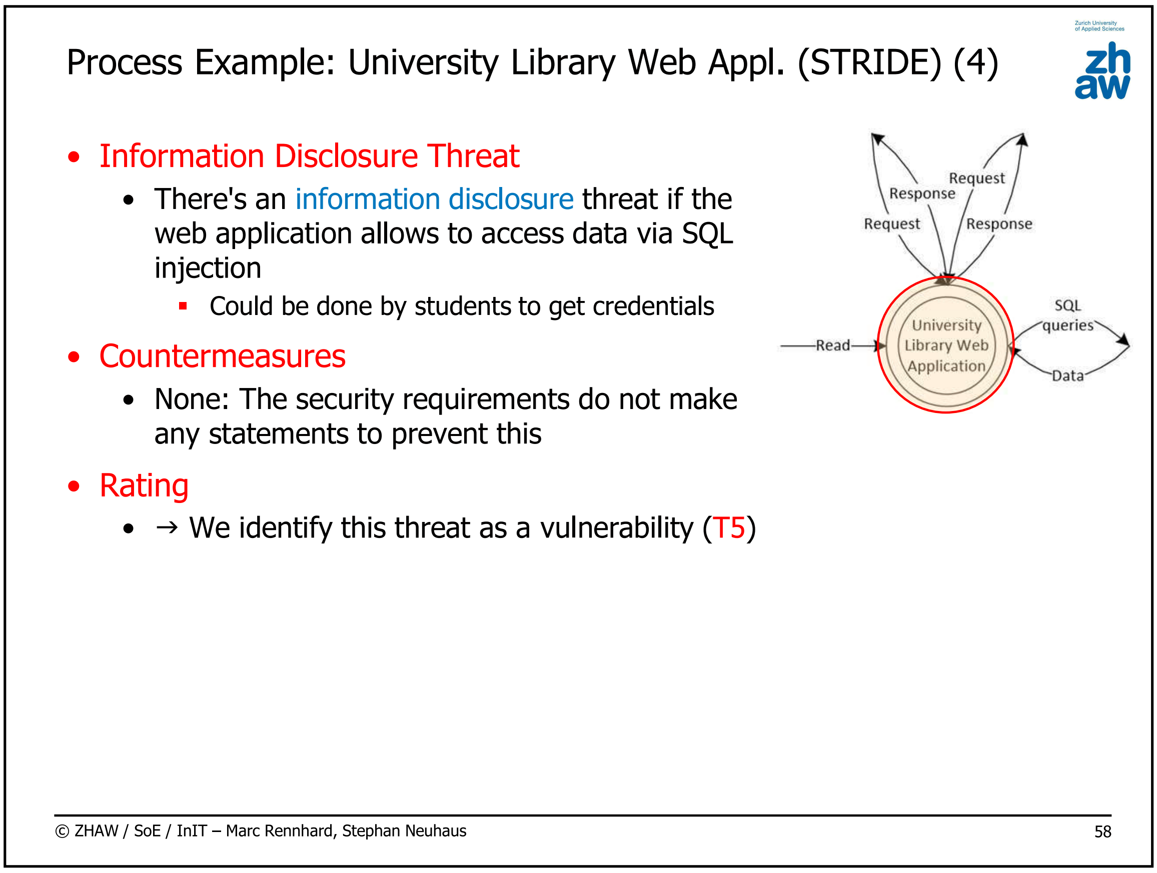


Reputation Threat – Entity, Process or Data Store?

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 an 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:

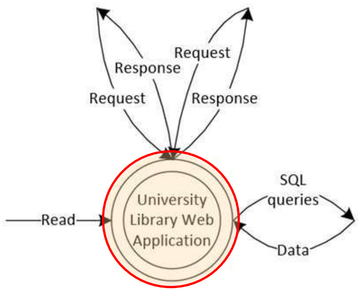
- *Data store*: To determine reputation threats here, you should think about scenarios that directly affect the data store (i.e., the logged data), without considering the application or the system that produces the logs. Typical threats are a system administrator that often has the necessary (too many?) 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 application) that may result in manipulating the log files. For instance, there may be an input validation problem that allows the user accessing the 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 a 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)

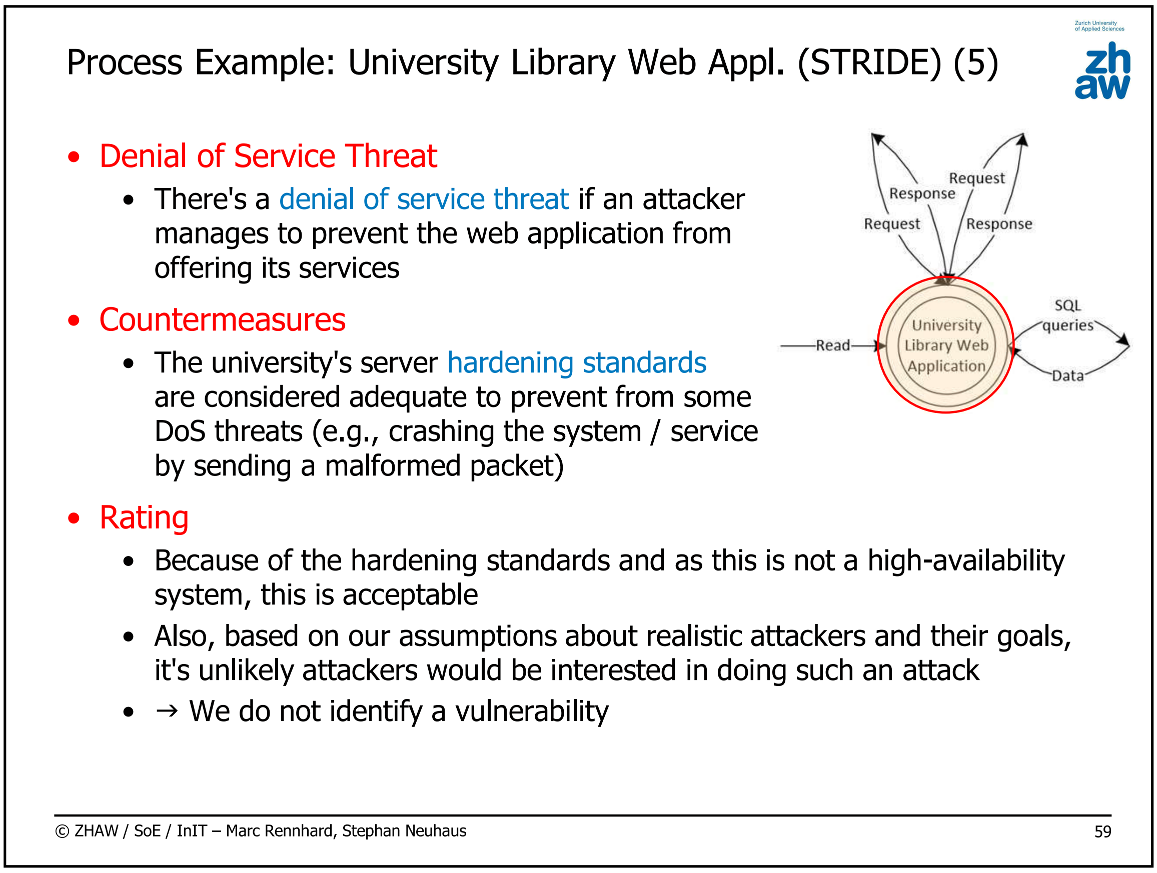
- Information Disclosure Threat
  - There's an information disclosure threat if the web application allows to access data via SQL injection
    - Could be done by students to get credentials
- Countermeasures
  - None: The security requirements do not make any statements to prevent this
- Rating
  - → We identify this threat as a vulnerability (T5)



© ZHAW / SoE / InIT – Marc Rennhard, Stephan Neuhaus 58

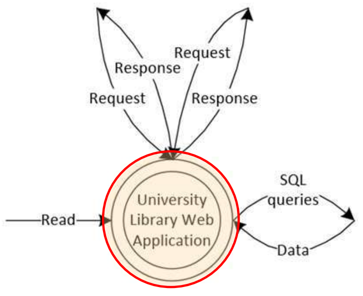
Information Disclosure and Processes

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.



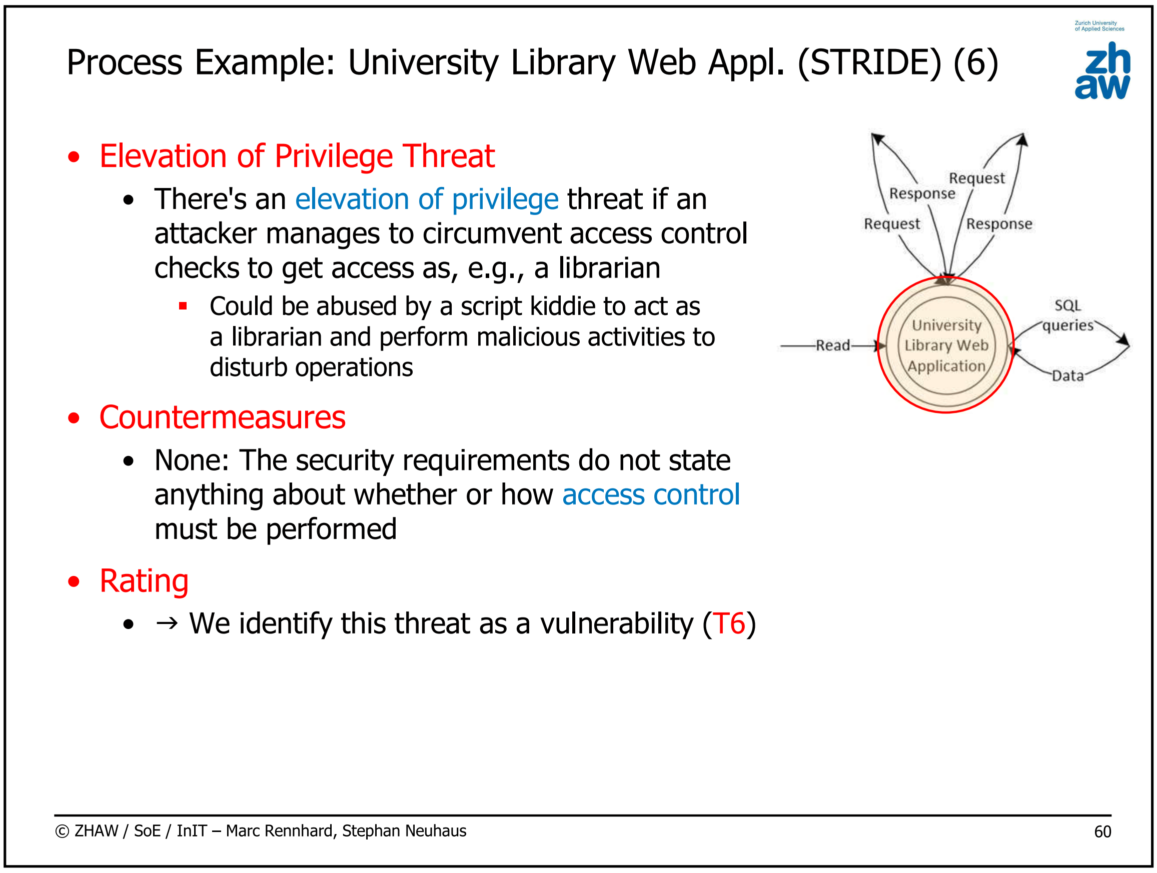
Process Example: University Library Web Appl. (STRIDE) (5)

- Denial of Service Threat
  - There's a denial of service threat if an attacker manages to prevent the web application from offering its services
- Countermeasures
  - 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)
- 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



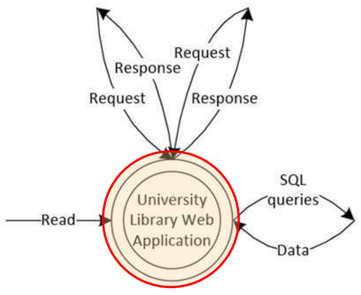
Denial of Service Threat against Processes

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
  - 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
- Countermeasures
  - None: The security requirements do not state anything about whether or how access control must be performed
- Rating
  - → We identify this threat as a vulnerability (T6)

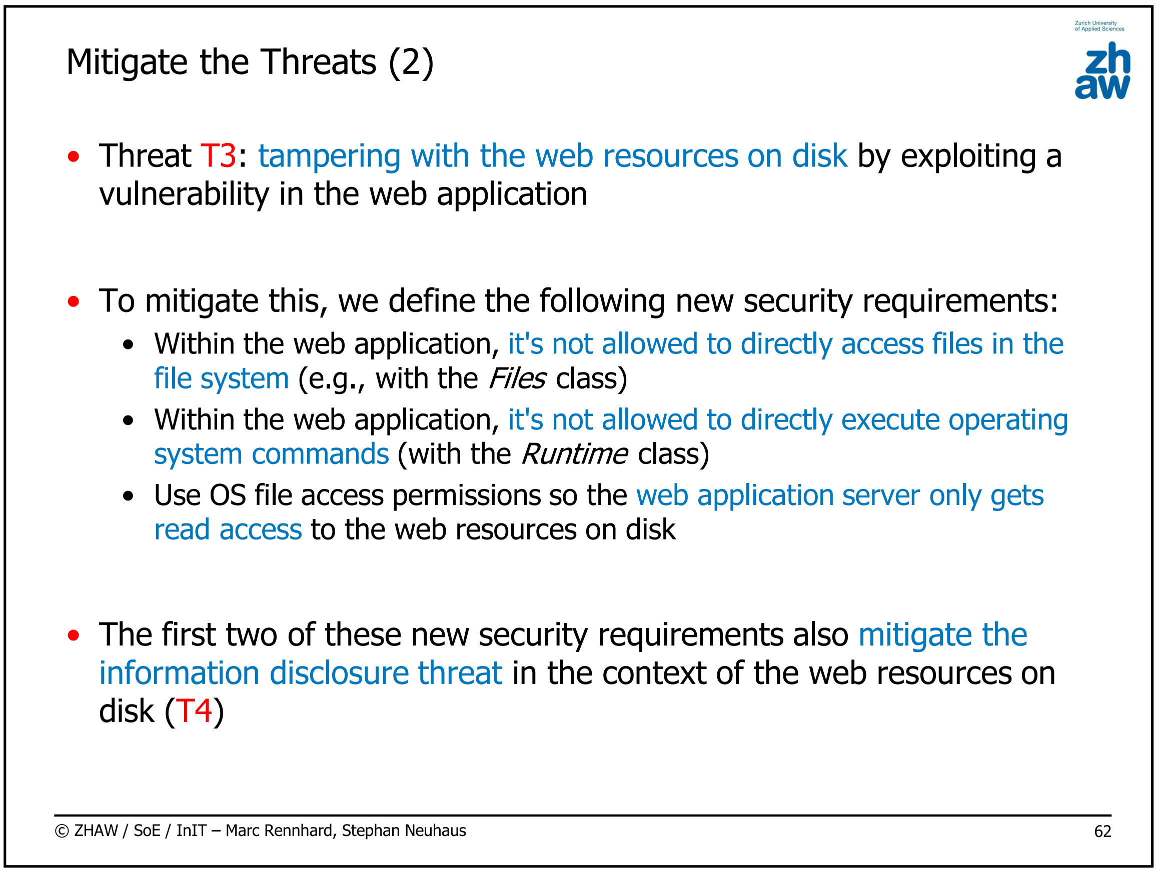


Assets and Vulnerabilities

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: Mitigate the Threats (1)

- The last step is to define new security requirements that eliminate or reduce the identified vulnerabilities
- Threat 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:
  - Use individual accounts for librarians with dedicated passwords
  - Enforce a minimal password quality (length, character mix, compare with dictionary,...)
  - 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 individual accounts



Mitigate the Threats (2)

- Threat T3: 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:
  - Within the web application, it's not allowed to directly access files in the file system (e.g., with the *Files* class)
  - Within the web application, it's not allowed to directly execute operating system commands (with the *Runtime* class)
  - Use OS file access permissions so the web application server only gets read access to the web resources on disk
- The first two of these new security requirements also mitigate the information disclosure threat in the context of the web resources on disk (T4)

Accesses to the Operating System

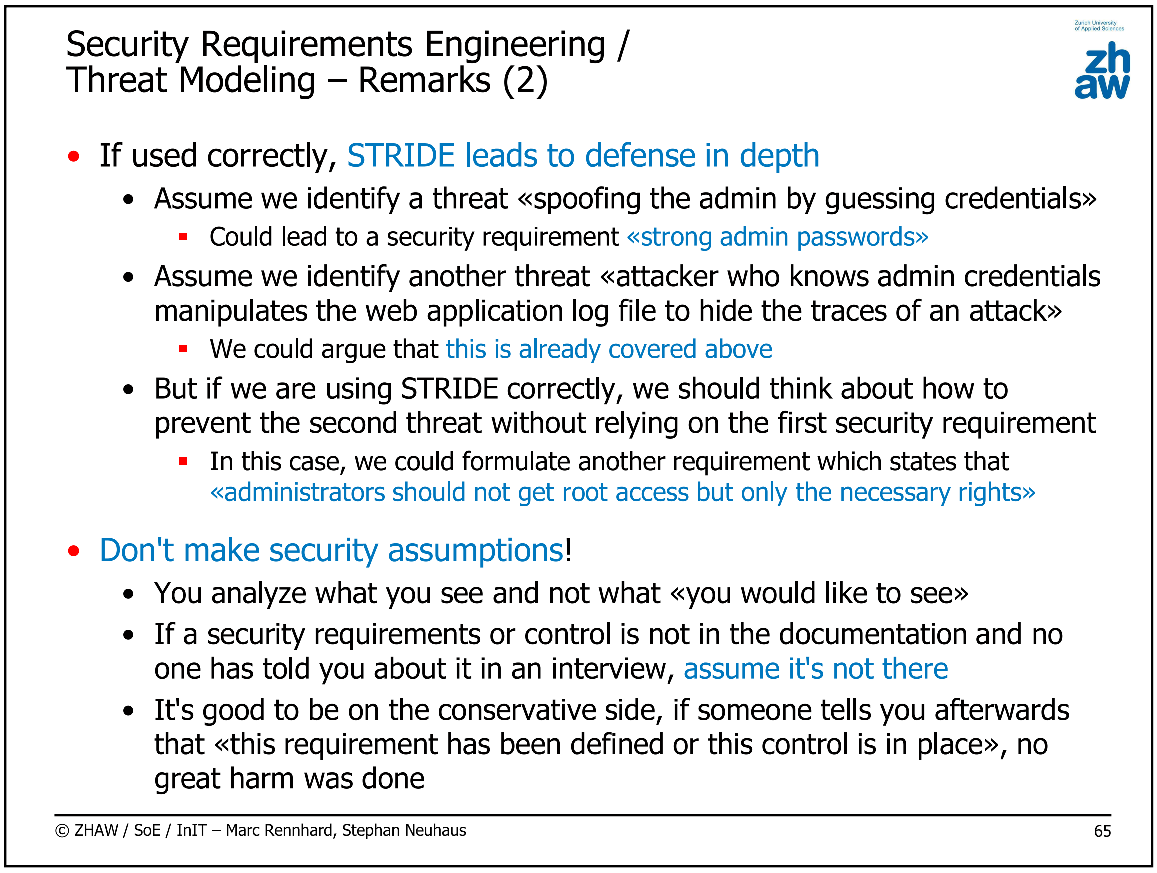
- Directly accessing files from the web application code is risky, as this may allow an attacker to find a way to access the web resources in the file system. And note that this is also a reasonable requirement as usually, such a direct access is not required in the web application code.
- Likewise,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

Mitigate the Threats (3)

- Threat T5: 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:
  - Accessing the database is only allowed via prepared statements
  - All data received from users should be considered non-trusted and should first be validated before processed further
  - Use a technical database user with minimal privileges
- Threat T6: 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:
  - The web application must employ an authorization mechanism
  - Authorization must be checked on every single request (complete mediation)

Security Requirements Engineering / Threat Modeling – Remarks (1)

- As said before, this is not a scientific method but rather a creative process
  - As a result, this is best be 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, it may be reasonable to identify all involved components in detail
    - E.g., the entire payment process in an e-banking system

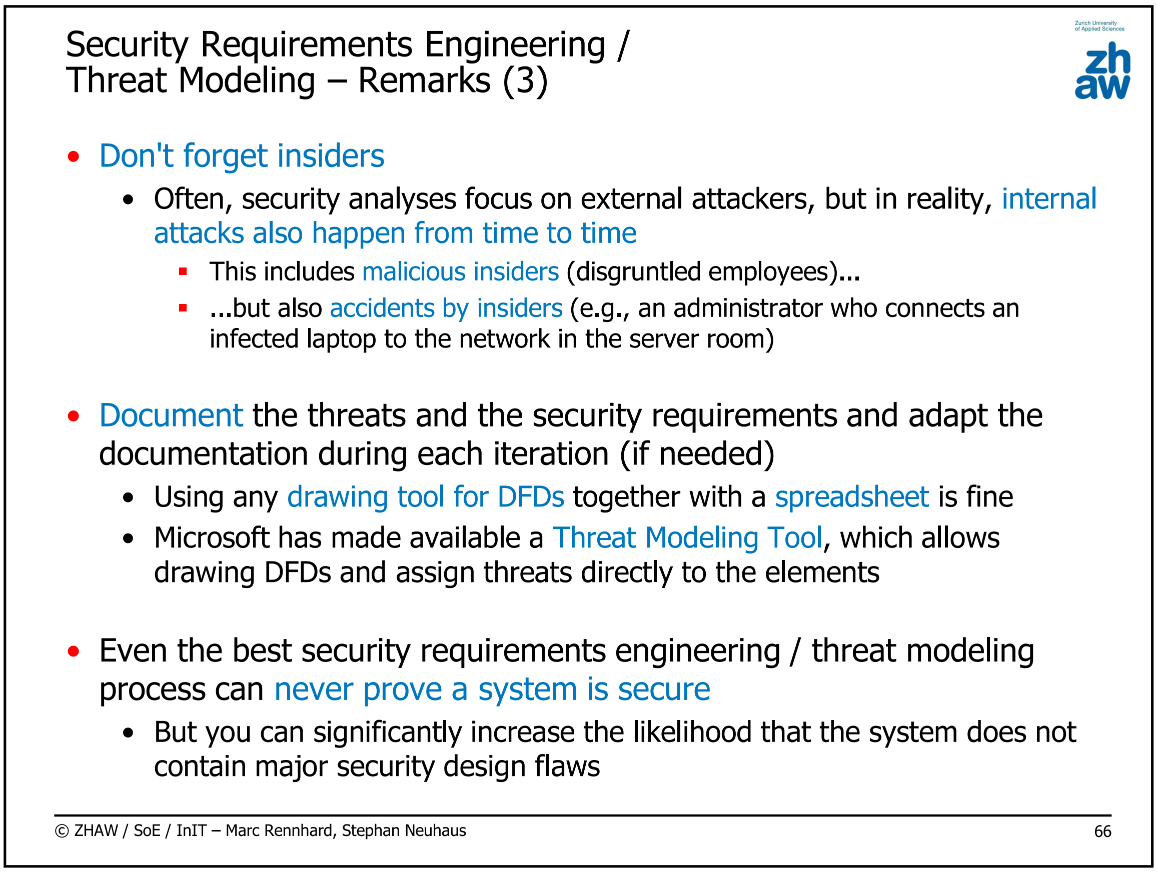


Security Requirements Engineering / Threat Modeling – Remarks (2)

- If used correctly, STRIDE leads to defense in depth
  - Assume we identify a threat «spoofing the admin by guessing credentials»
    - 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 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»
- 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

Defense in Depth

Thinking about threats against each DFD element and defining appropriate security requirements will likely result in multiple security requirements that ultimately protect from the same attack scenario. But that’s OK from a defense in depth approach because defense in depth is usually a desired property. When reviewing the complete list of security requirements, it is 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 3 to 2.



Security Requirements Engineering / Threat Modeling – Remarks (3)

- 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 Threat Modeling Tool, which allows drawing DFDs and assign threats directly to the elements
- 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

SDL Threat Modeling Tools (Windows only)

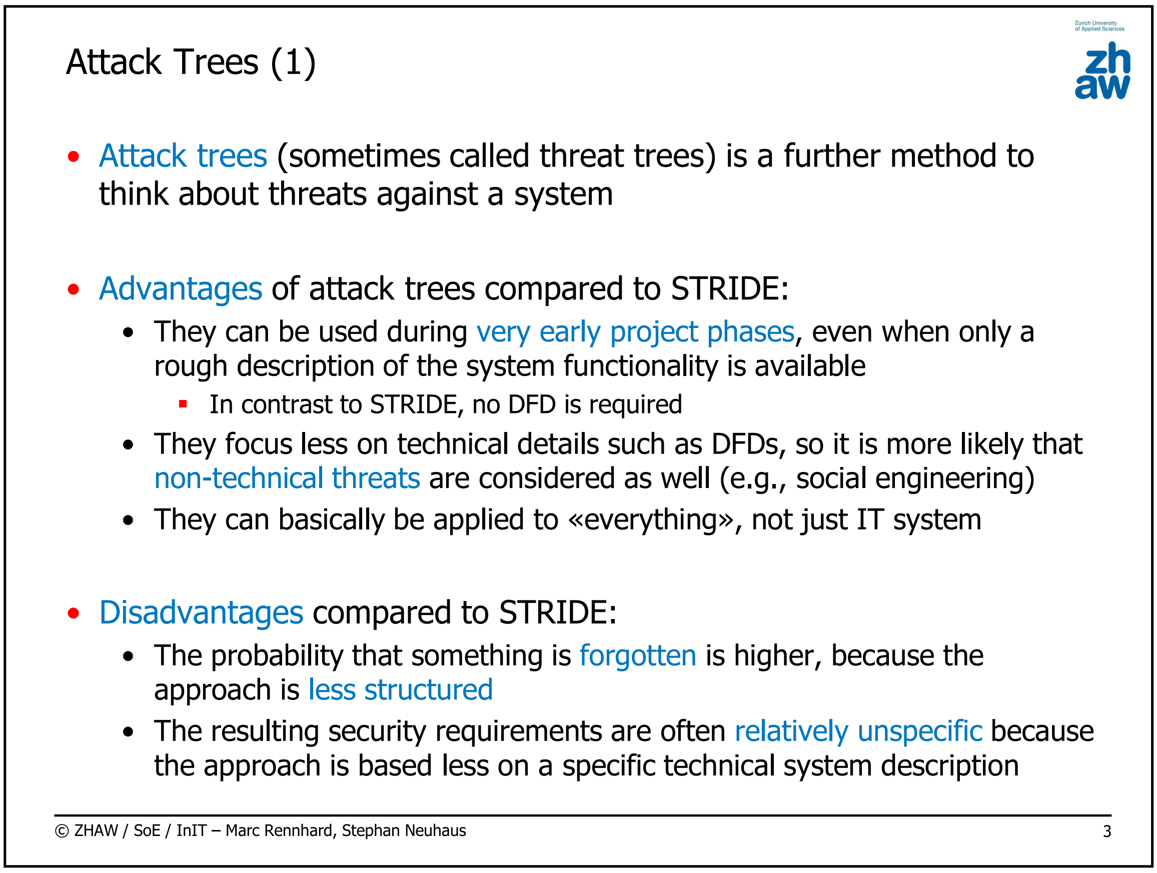
More information – and the tool itself – is available at
https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling

Security Requirements Engineering and Threat Modeling – Part 2/2

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

Zurich University of Applied Sciences

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.

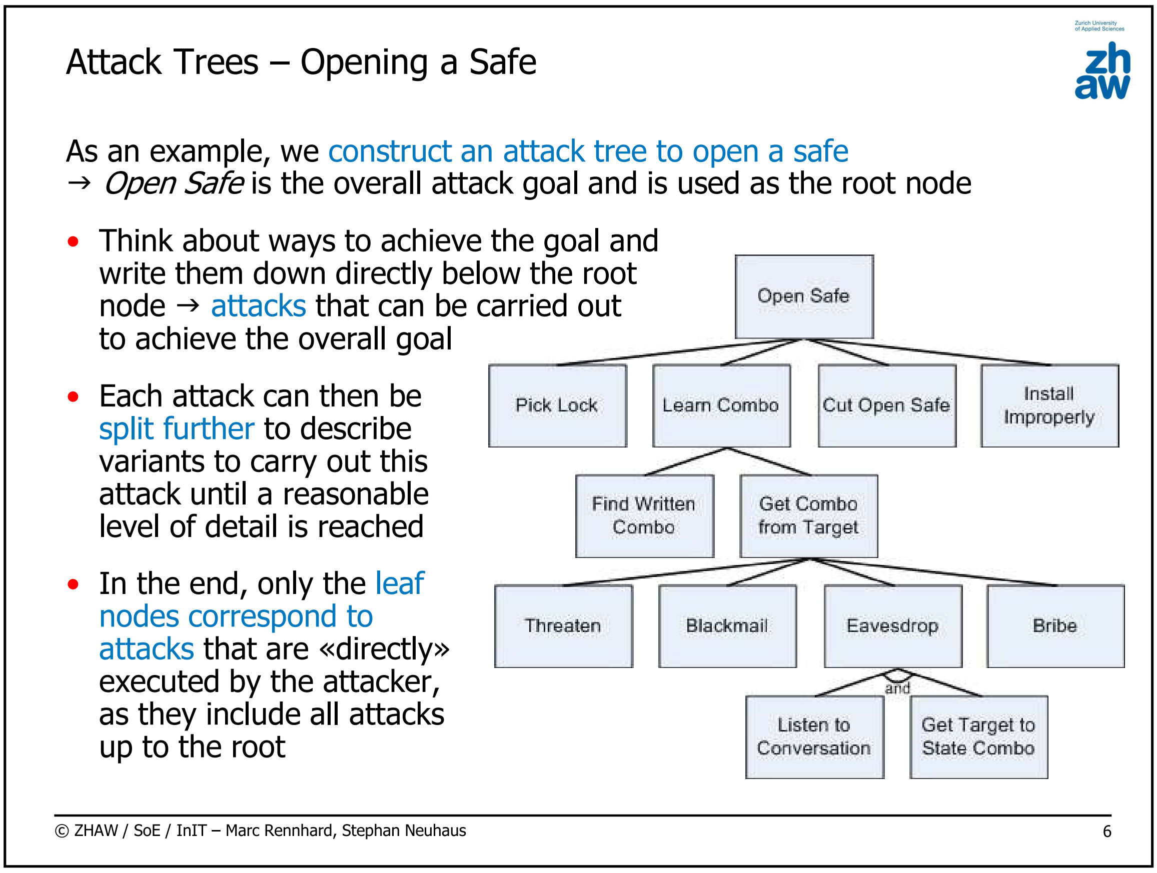
Sometimes, attack trees are also called “threat trees”.

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)

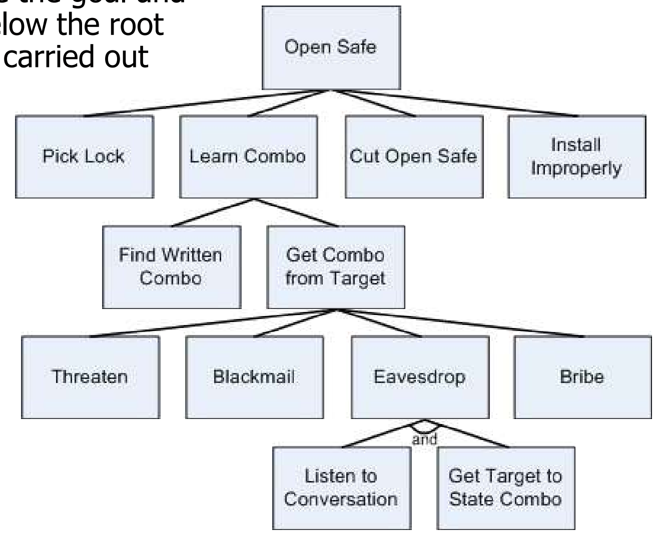
- To use attack trees, start by defining high-level attack goals
  - Steal credit cards, make system unavailable, open safe,...
  - For each attack goal, an attack tree is constructed
- To construct the tree, think about different ways to achieve this goal
  - E.g., «stealing credit cards» can be achieved 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

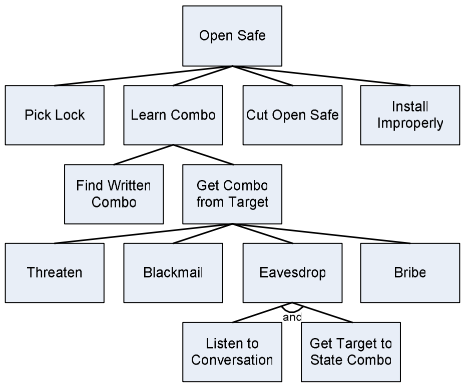
The goal is opening the safe. To open the safe, attackers can pick the lock, learn the combination, cut open the safe, or install the safe improperly so that they can easily open it later. To learn the combination, they either have to find the combination written down or get the combination from the 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

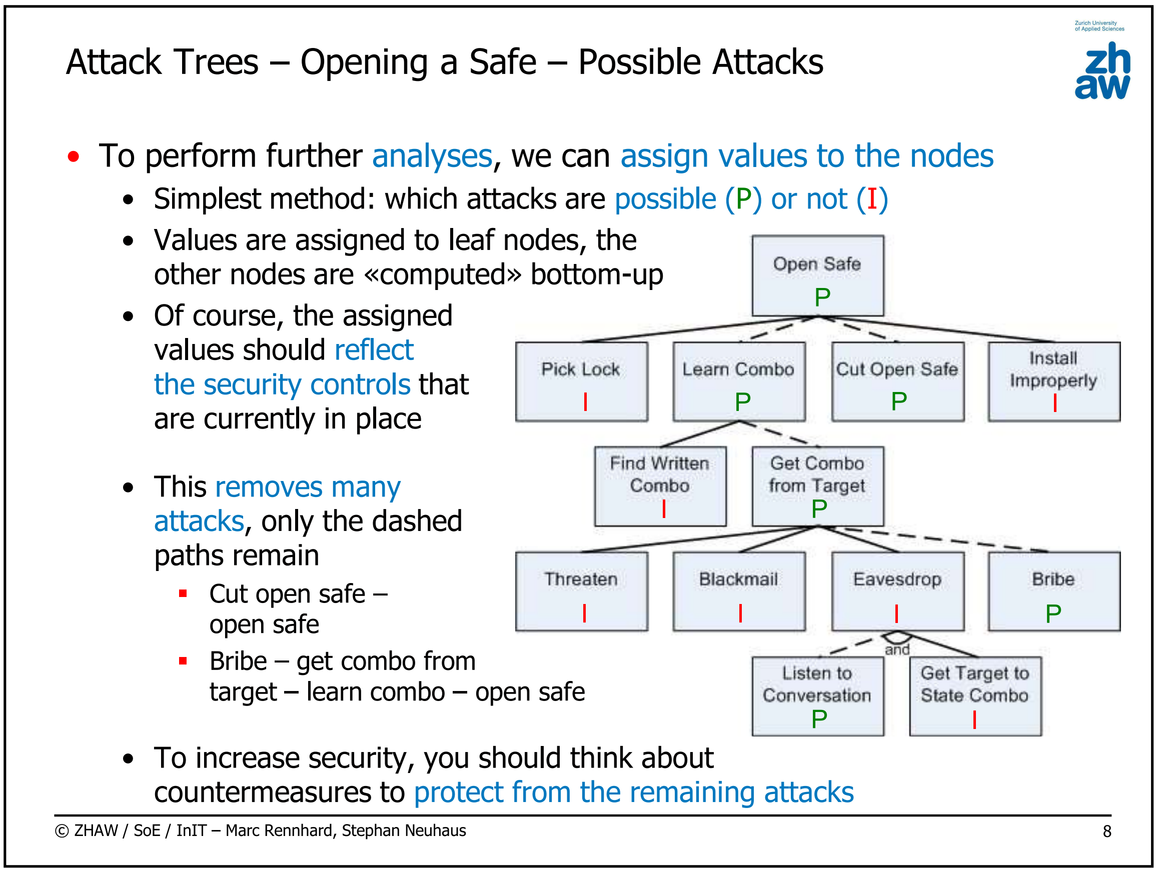
At the end, only the leaf nodes are attacks that are «directly» executed by the attacker as they include all attacks up to the root. So, for example, if the attacker manages to successfully bribe someone who knows the combination (so the attacker «solves» leaf node «Bribe», then node «Get Combo from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

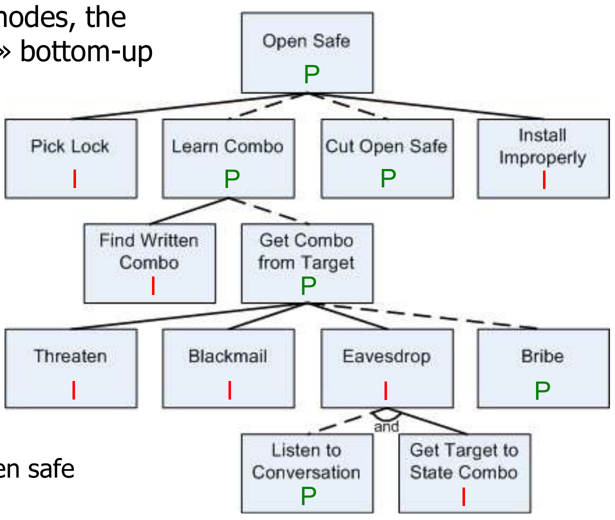


- AND nodes mean that multiple problems must be solved
  - To eavesdrop on someone speaking out the safe combination, attackers have to listen to the conversation AND to get that person to speak out the combination



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

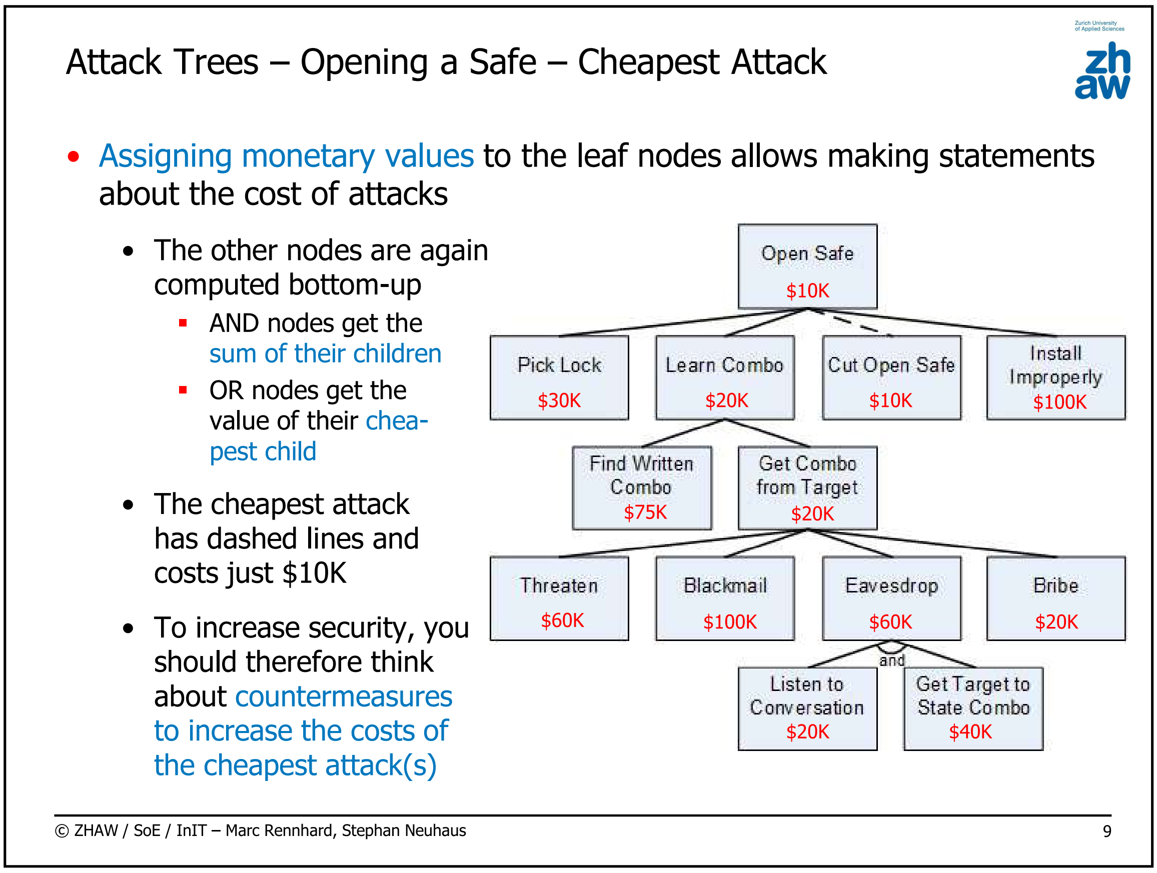
For instance:

- «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

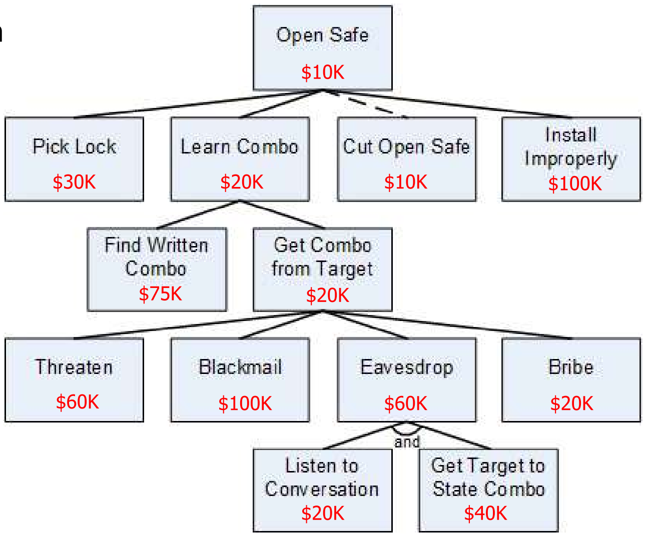
Assigning other boolean values is possible

- 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

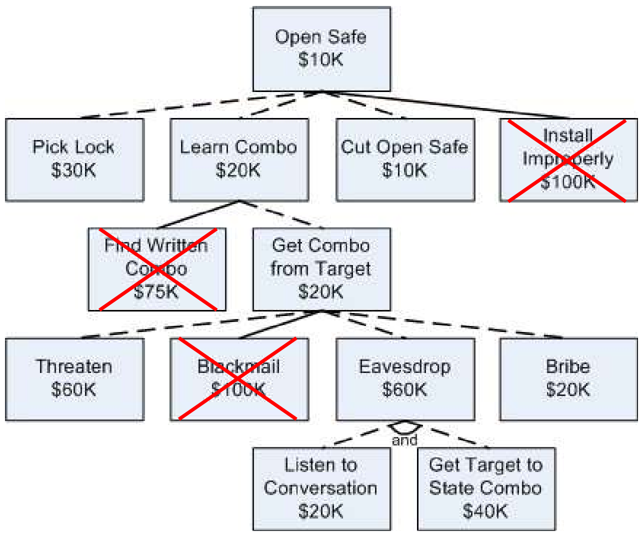
Note that the actual numbers are completely fictional as this is just an example.

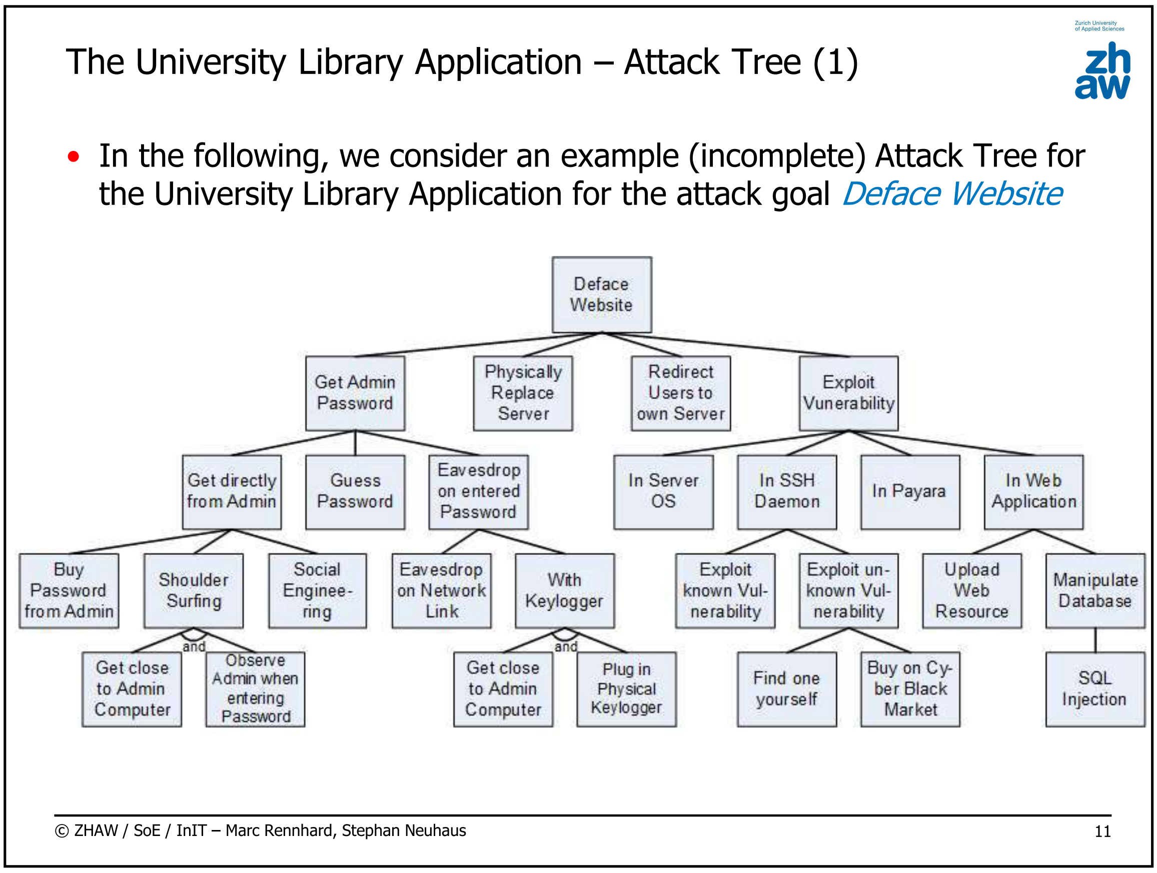
Computing Bottom-Up

- «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

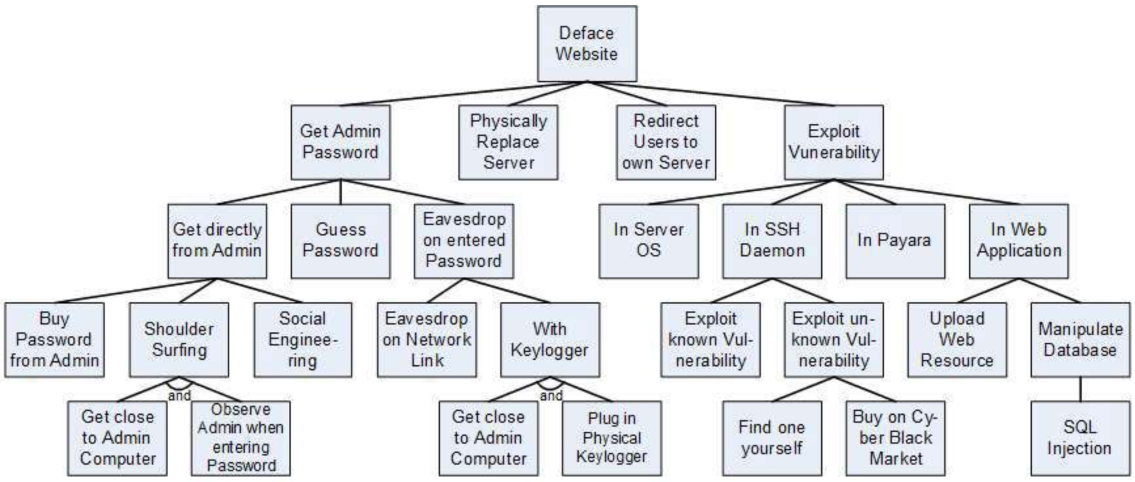
- Monetary values can also be used to determine all attacks that are, e.g., cheaper than $70K
  - If we assume that there $70K in the safe, the attacker probably won't invest more in an attack
  - One simply removes the nodes with costs of $70K or higher, which eliminates the corresponding attacks
 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

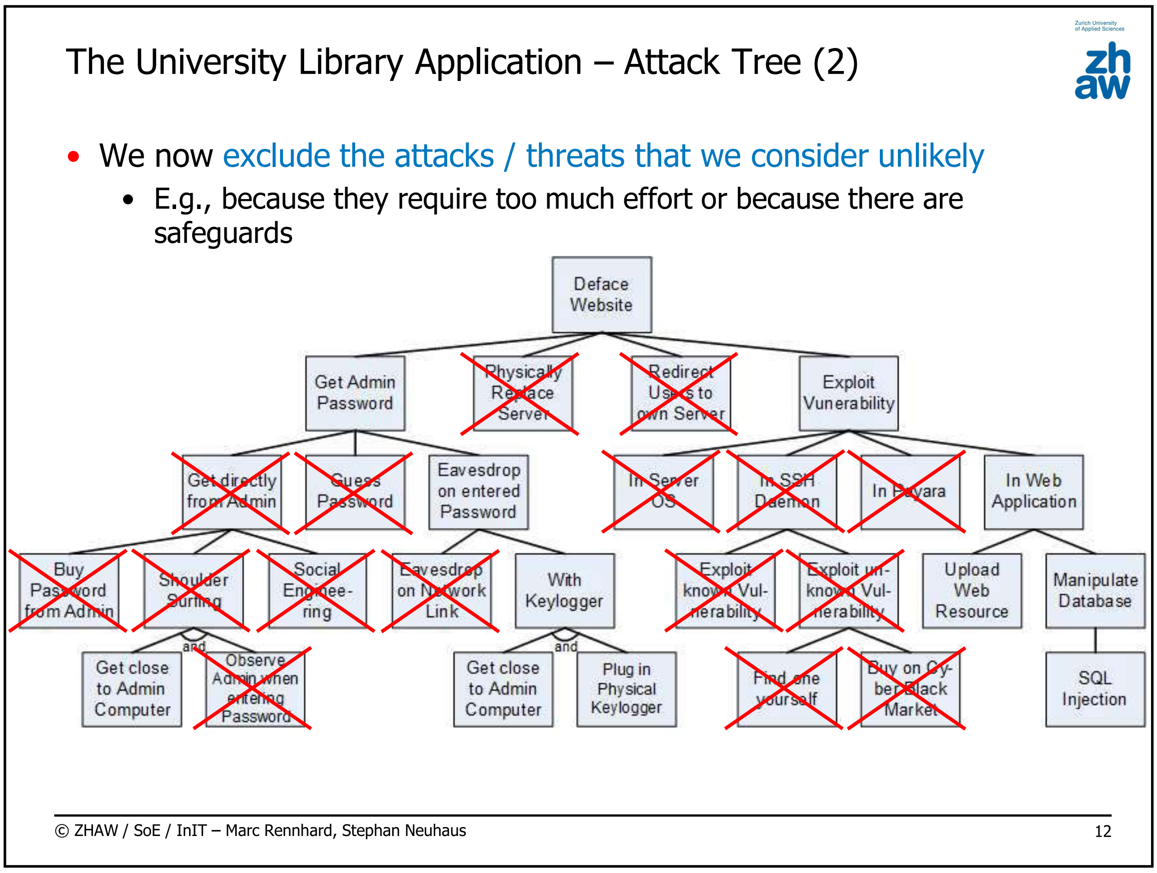
© ZHAW / SoE / InIT – Marc Rennhard, Stephan Neuhaus 11

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

© ZHAW / SoE / InIT – Marc Rennhard, Stephan Neuhaus 12

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