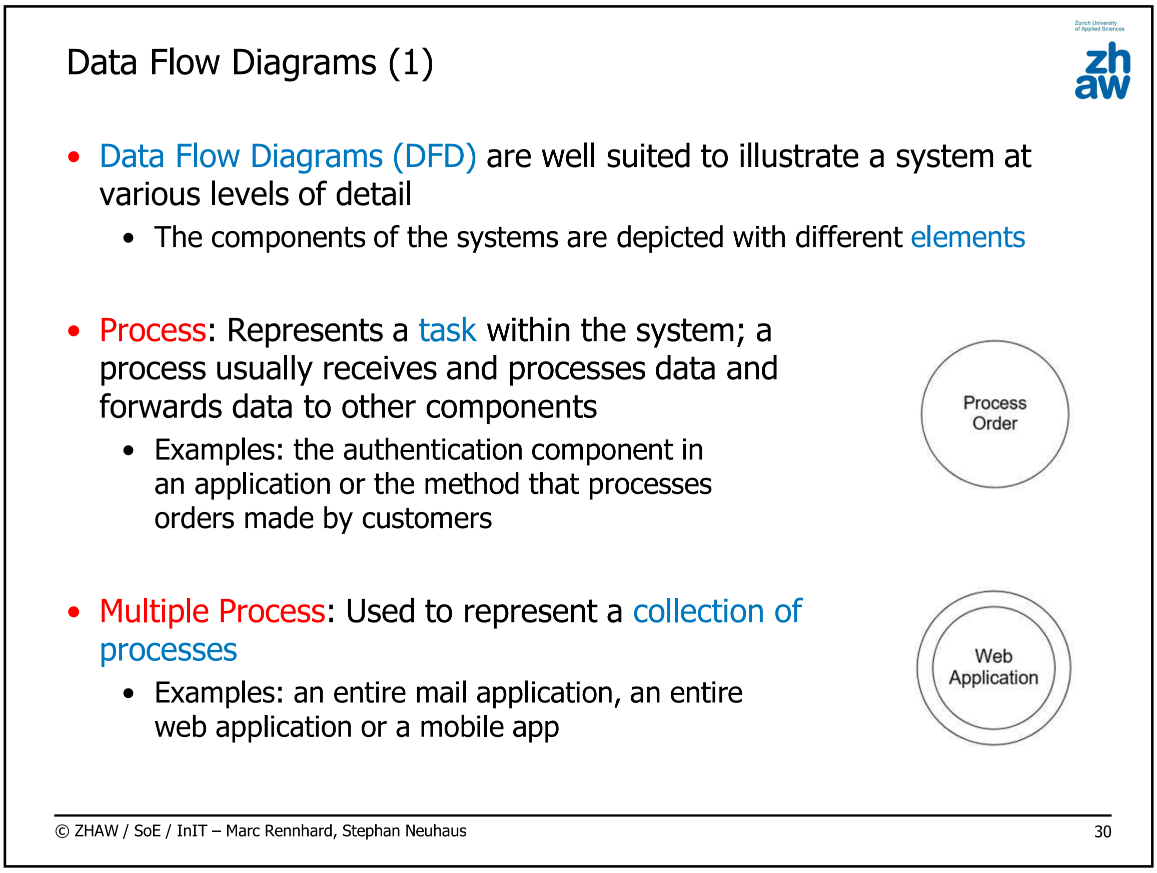
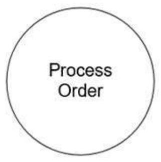
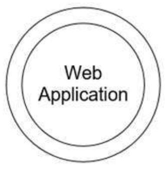


## Data Flow Diagrams (1)

- Data Flow Diagrams (DFD) 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

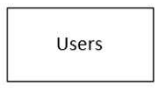


**Process vs. Multiple Process**

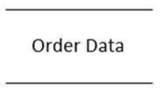
In practice, it's often not so clear if «something» is a process or a multiple process. But in the end, it doesn't matter too much in practice where you draw the line between the two, because when applying STRIDE on the elements, processes and multiple processes are treated equally

## Data Flow Diagrams (2)

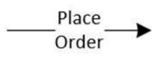
- External Entity: Used to represent any entity outside the system that interacts with the system
  - Examples: users, administrators, but also external components used by / interacting with the system (e.g., a 3rd party web service)

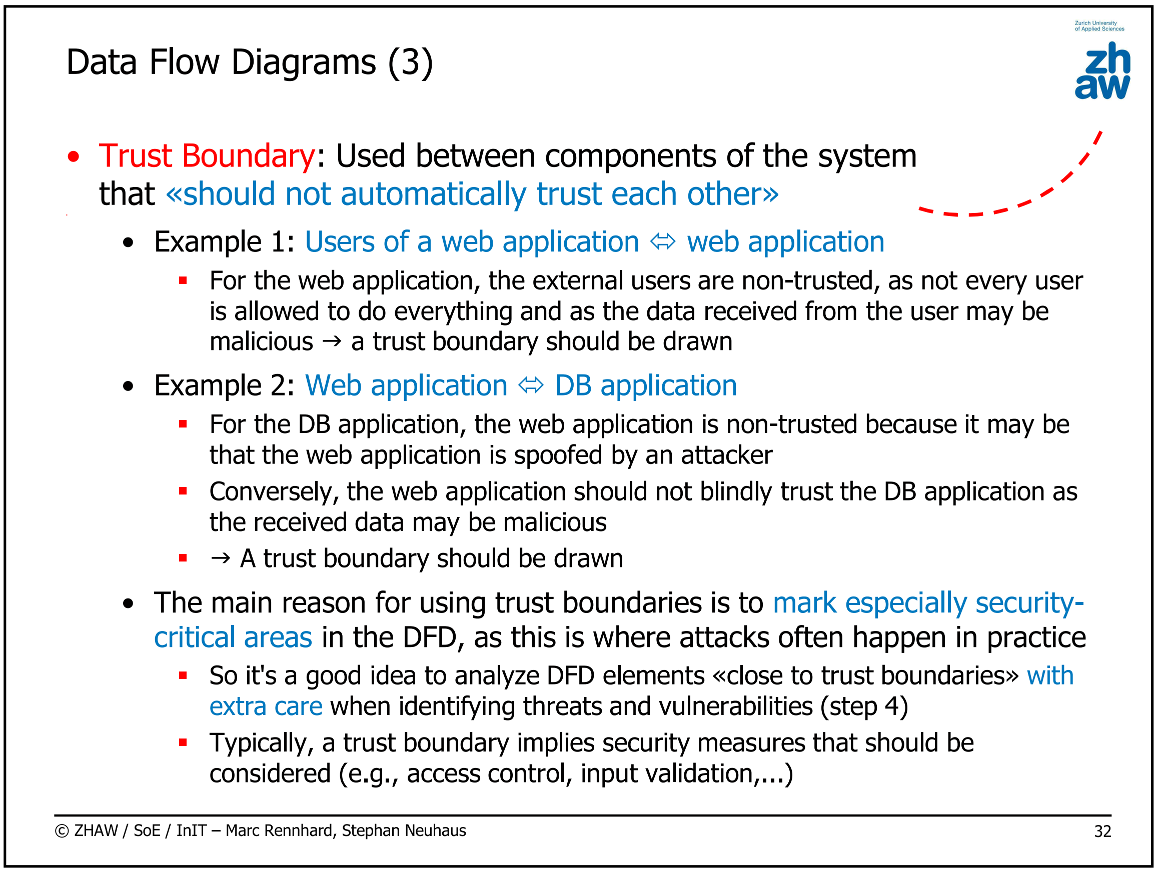


- Data Store: Used to represent locations where data is stored
  - Examples: database files, configuration files, log files



- 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)

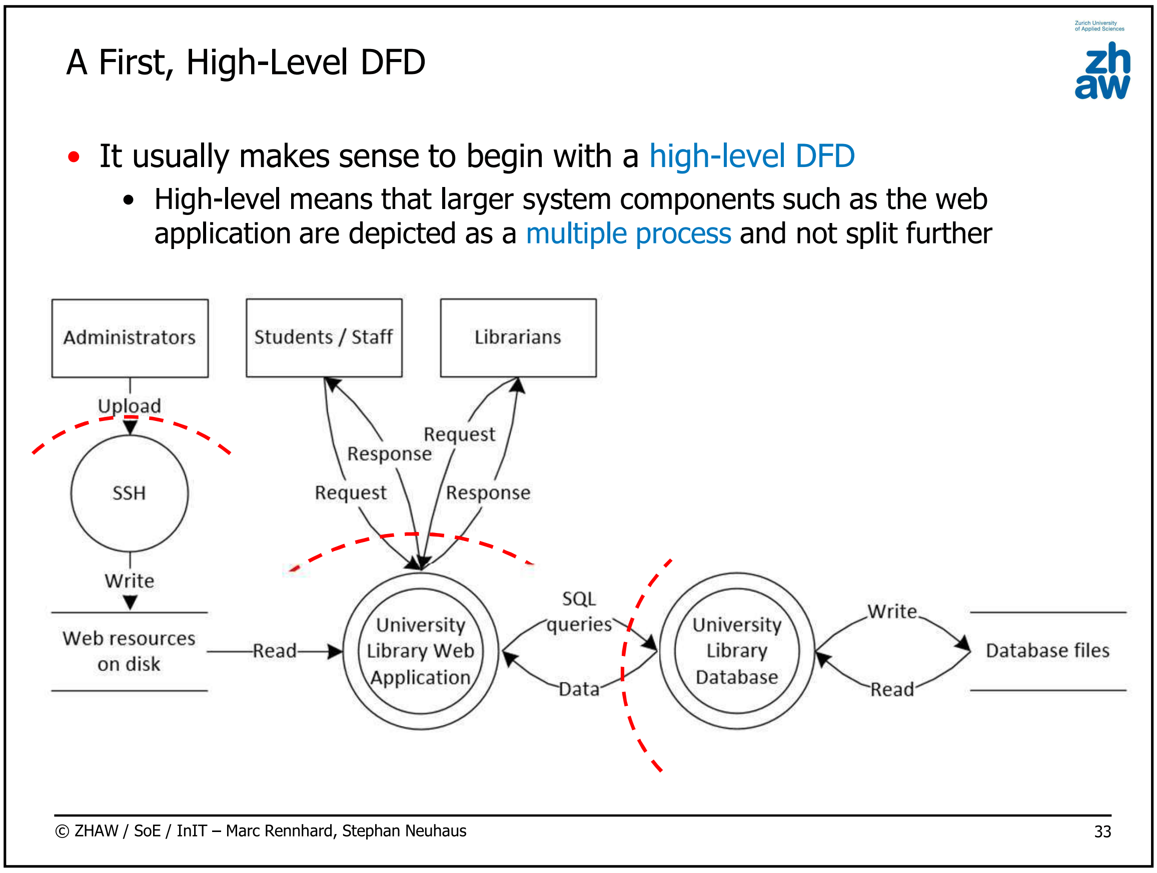
- Trust Boundary: Used between components of the system that «should not automatically trust each other»
  - Example 1: Users of a web application ⇔ web application
    - For the web application, the external users are non-trusted, as not every user is allowed to do everything and as the data received 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,...)

**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.

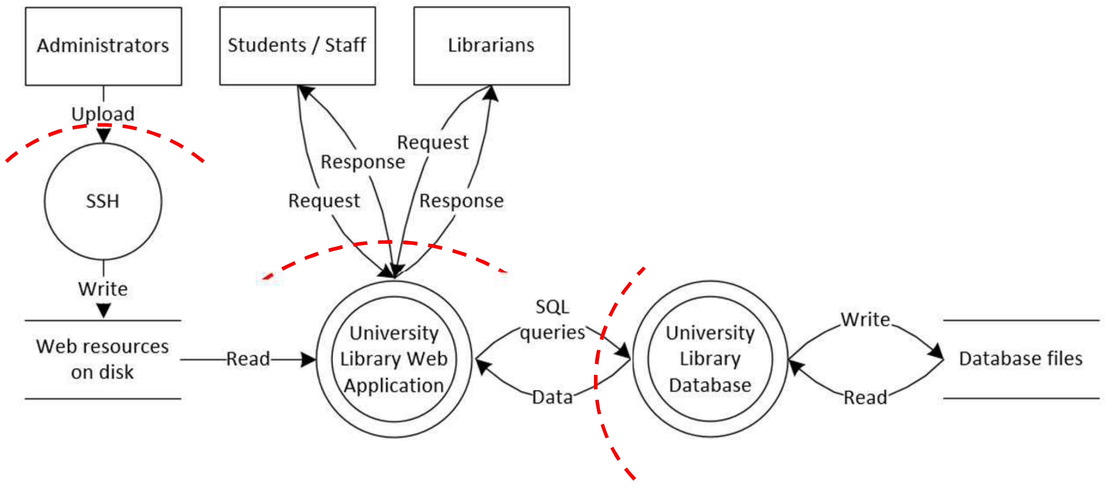
Trust Boundaries also exist between systems. For instance, the web application process may access a database process but most likely, the web application should not be able to do everything (e.g., changing the DB schema) and there may be auditing data in the DB that should not be leaked. So again, a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

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):

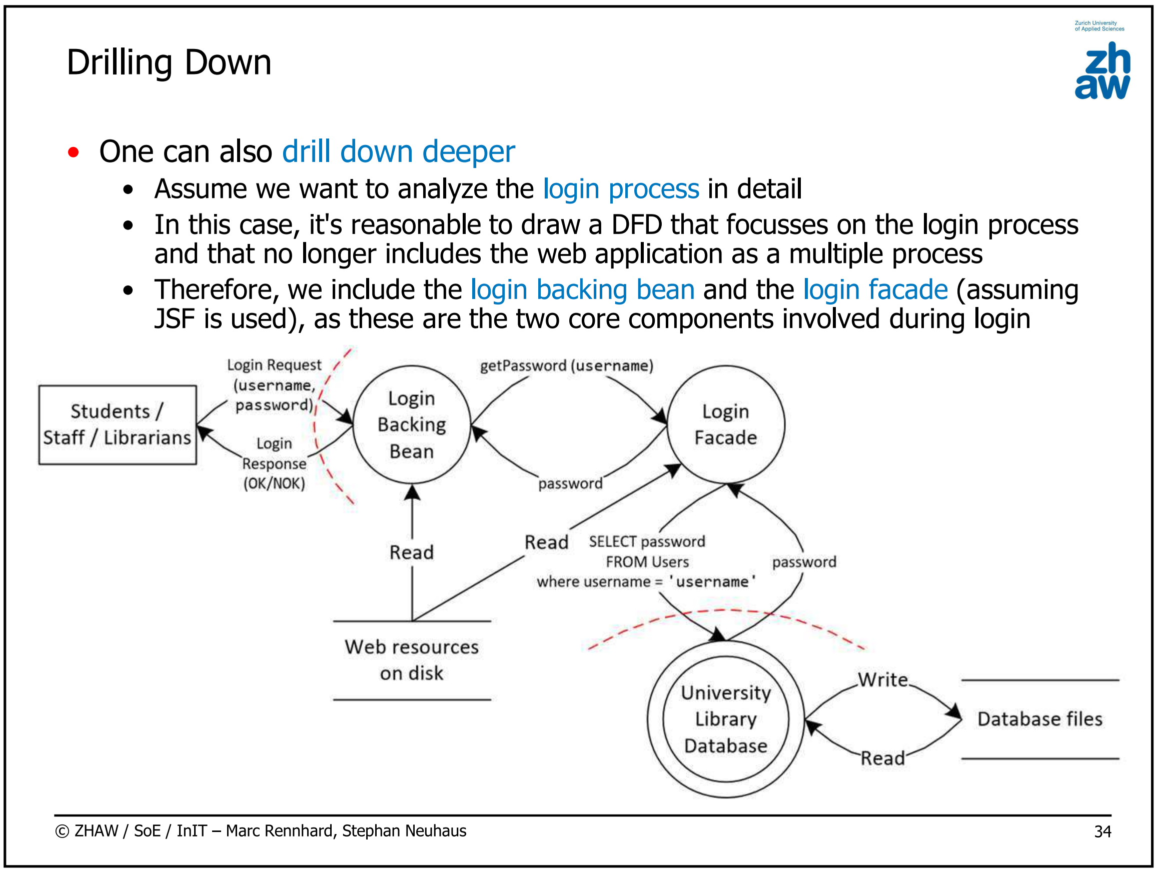
- Both server-side systems (integrity of systems and data) – they are obviously present in the DFD.
- Personal user data (account details, reservations) – that's stored in the database files.
- Credentials of users (username & password of students, staff, librarians) – that's also stored in the database 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).

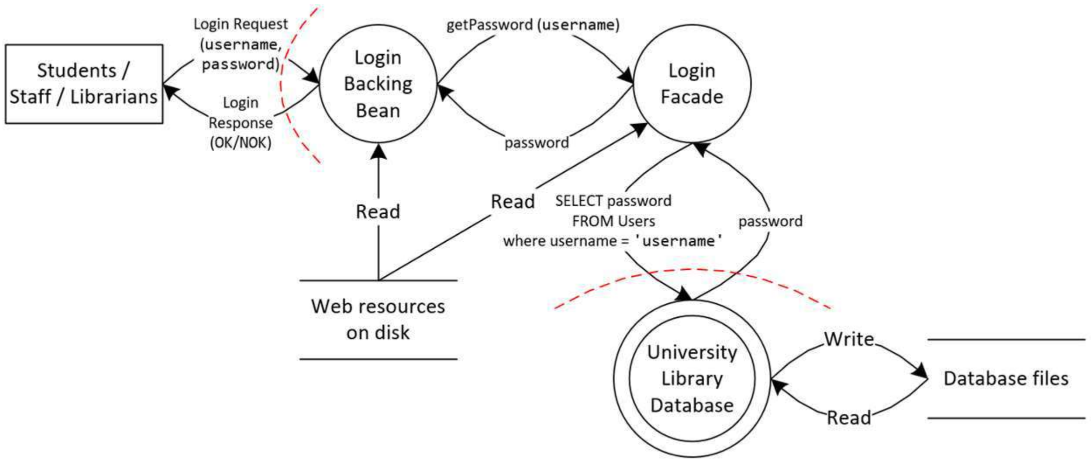
**Where are the Users' Computers / Browsers / Tools?**

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

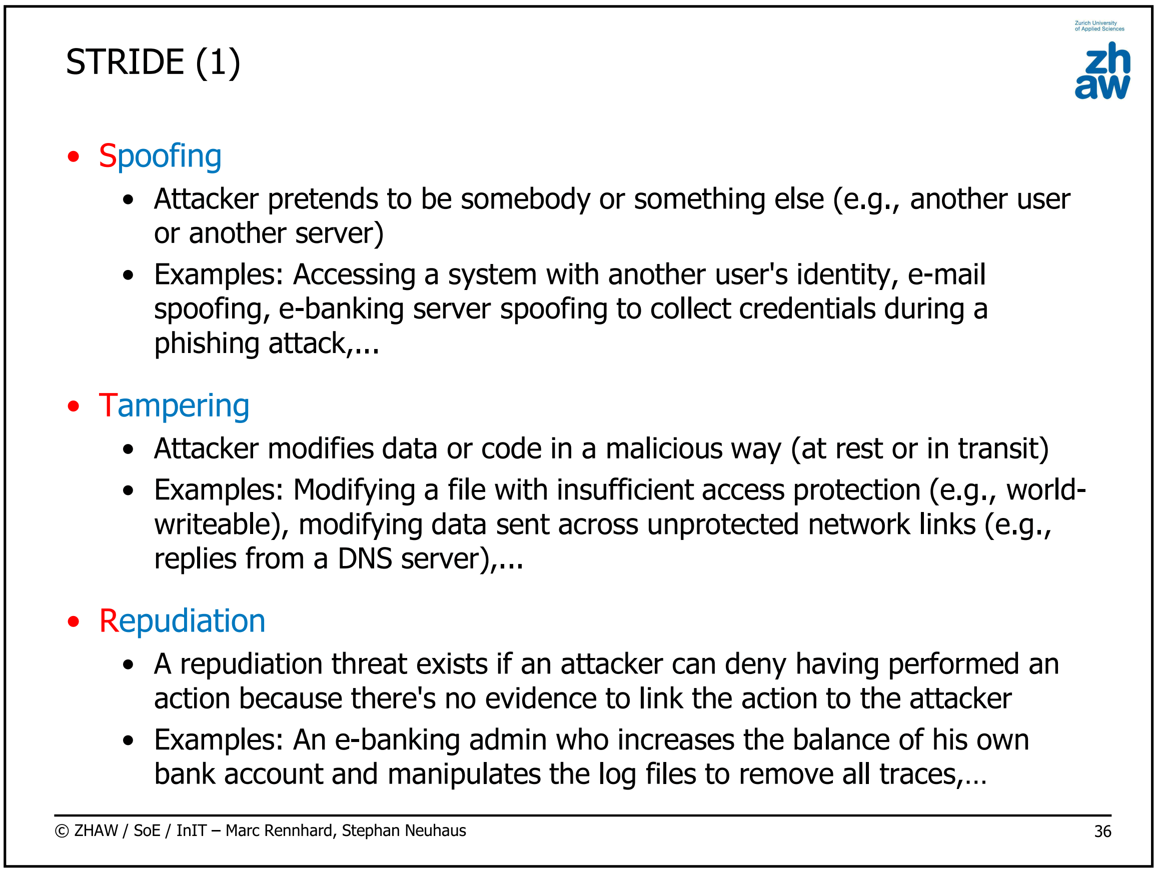
**Drilling Down**

DFDs can be drawn at every possible detail level. 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 (especially with respect to some security-relevant core functions) if necessary.

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 on 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.

## Step 4: Identify Threats and Vulnerabilities

- The next step is to identify threats that are relevant for the system under analysis and to find vulnerabilities based on these threats
- A good way to do this 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)

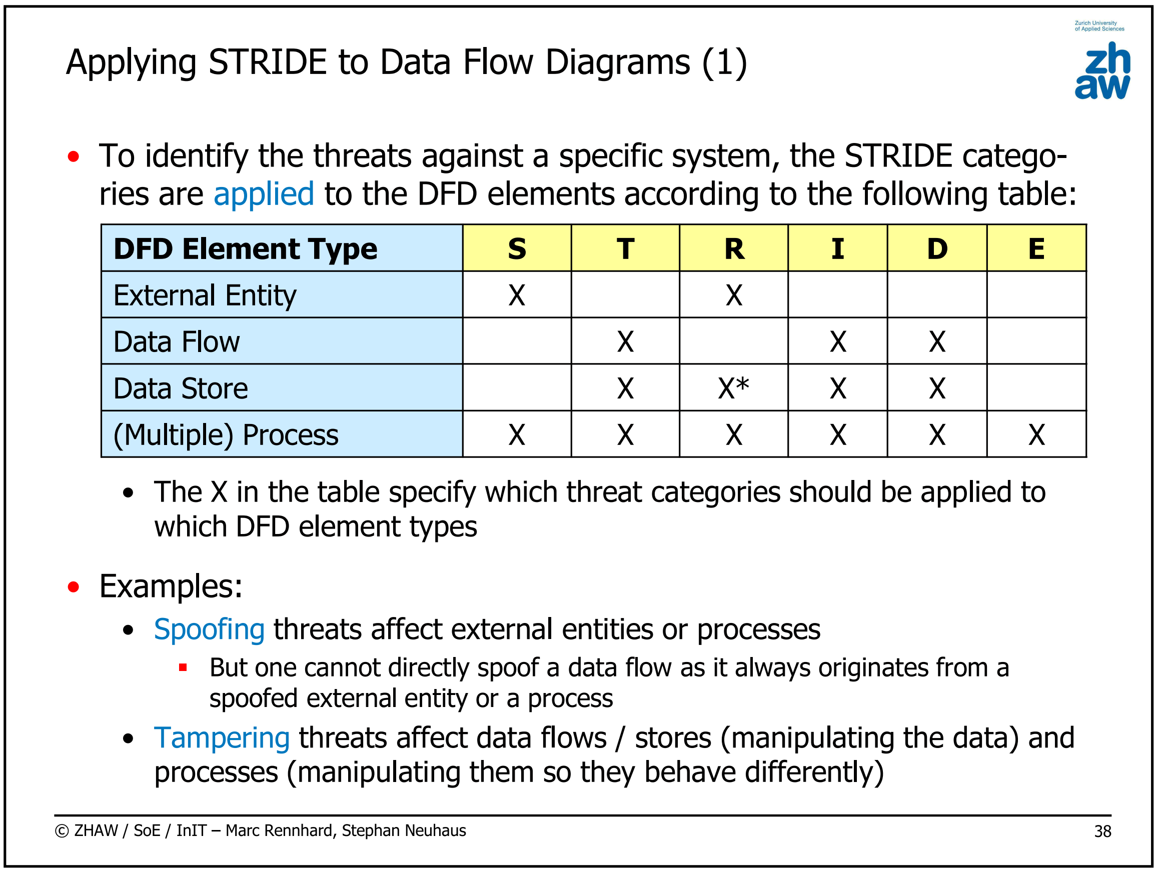
- Spoofing
  - Attacker pretends to be somebody or something else (e.g., another user or another server)
  - Examples: Accessing a system with another user's identity, e-mail spoofing, e-banking server spoofing to collect credentials during a phishing attack,...
- Tampering
  - Attacker modifies data 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)

* 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**

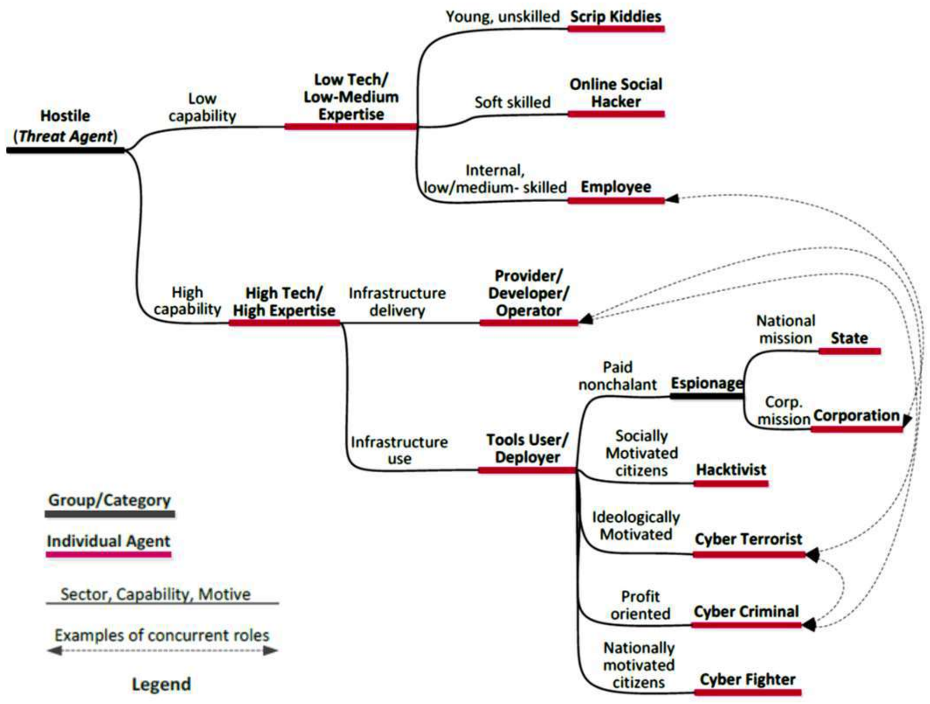
- Spoofing threats affect external entities (e.g., an attacker masquerading as a legitimate user) or processes (e.g., an attacker pretending to be an e-banking server).
- Tampering threats affect data flows (manipulating data in transit), data stores (manipulating stored data) and processes (manipulating 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 conformation 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 male 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

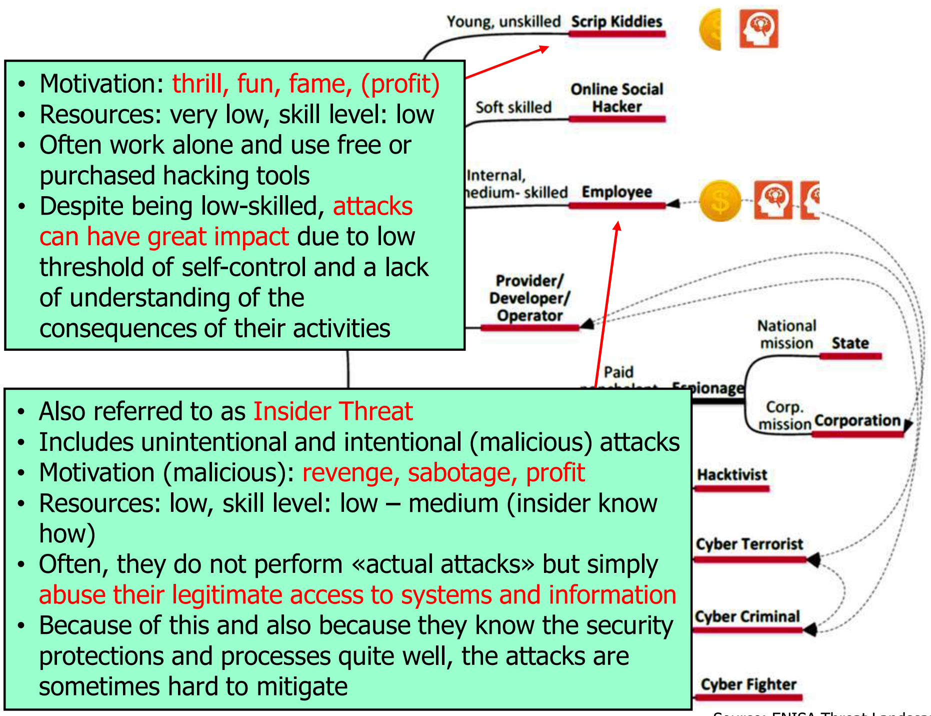
## Threat Agents according to ENISA (1)



Source: ENISA Threat Landscape Report, 2015

**ENISA** = European Union Agency for Network and Information Security

## Threat Agents according to ENISA (2)

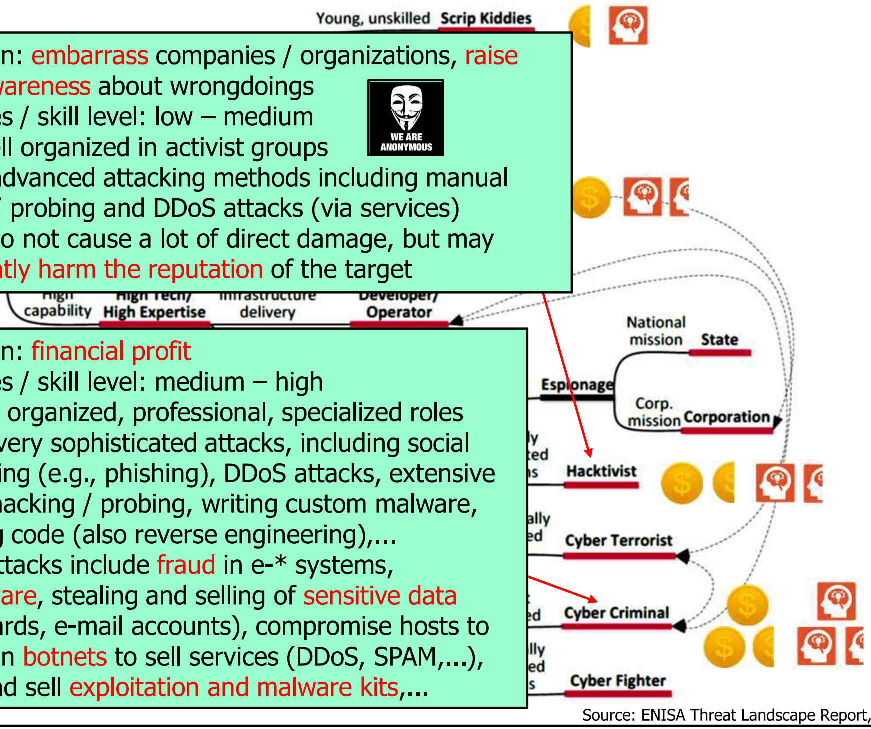


- Motivation: thrill, fun, fame, (profit)
- Resources: very low, skill level: low
- Often work alone and use free or purchased hacking tools
- Despite being low-skilled, attacks can have great impact due to low threshold of self-control and a lack of understanding of the consequences of their activities

- 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)

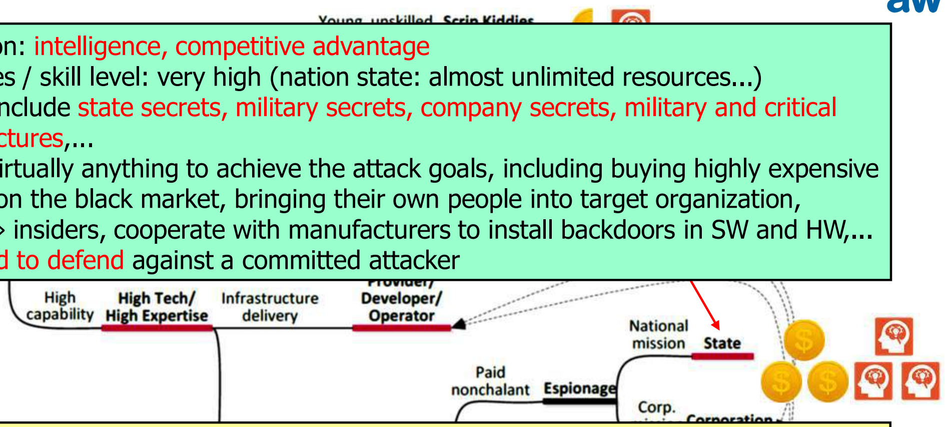


- Motivation: embarrass companies / organizations, raise public awareness about wrongdoings
- Resources / skill level: low – medium
- Often well organized in activist groups
- Employ advanced attacking methods including manual hacking / probing and DDoS attacks (via services)
- Usually do not cause a 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,...), create and sell exploitation and malware kits,...

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:

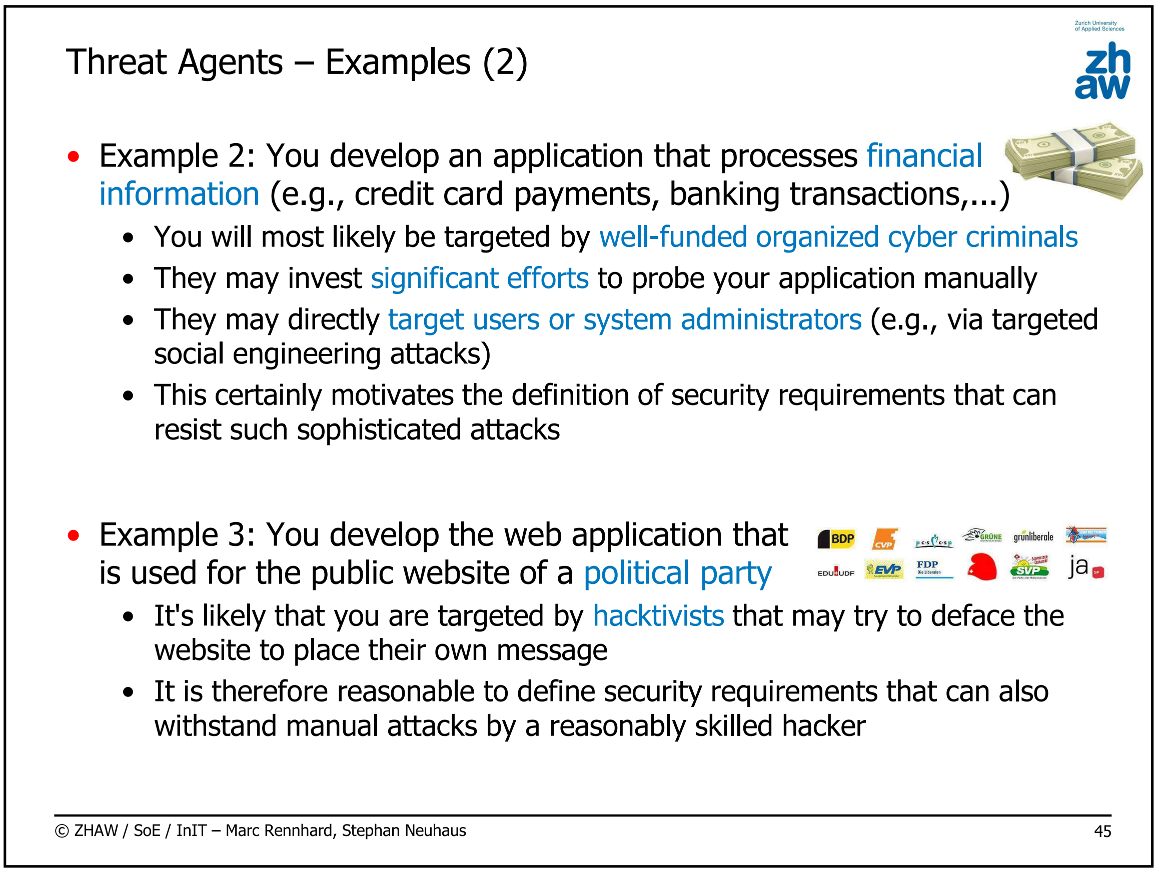
- Nation states have a specific target in mind and will go for it, at all costs
    - So even if the target has very high security standards, the chances are high that the attacker eventually succeeds
- Cyber criminals «just» want to achieve financial profit; in the end, they don't care, where they'll 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



## 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

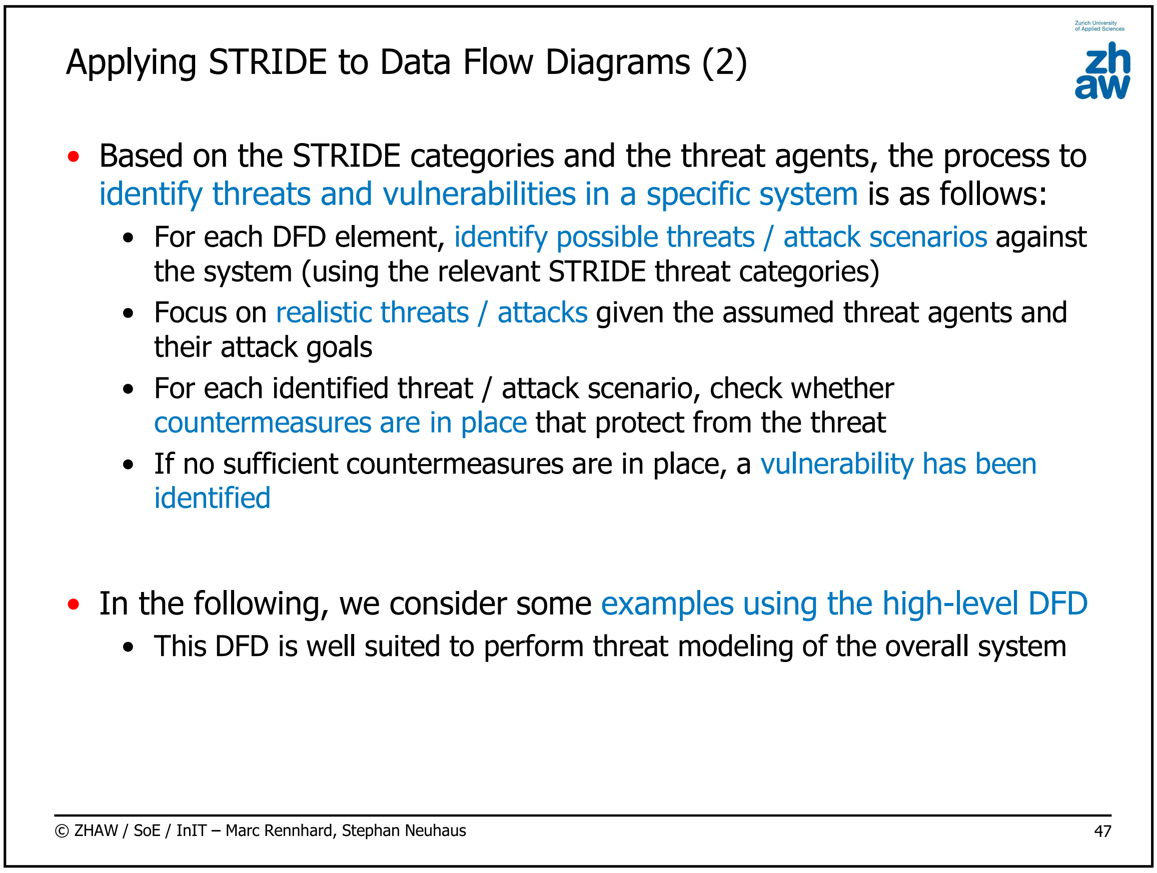
**Security Measures for these Examples**

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)

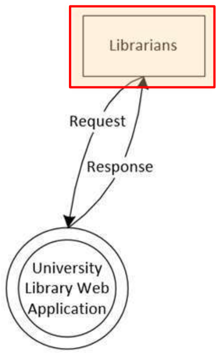
- Based on the STRIDE categories and the threat agents, the process to identify threats and vulnerabilities in a specific system is as follows:
  - For each DFD element, identify possible threats / attack scenarios against the system (using the relevant STRIDE threat categories)
  - Focus on realistic threats / attacks given the assumed threat agents and their attack goals
  - For each identified threat / attack scenario, 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».

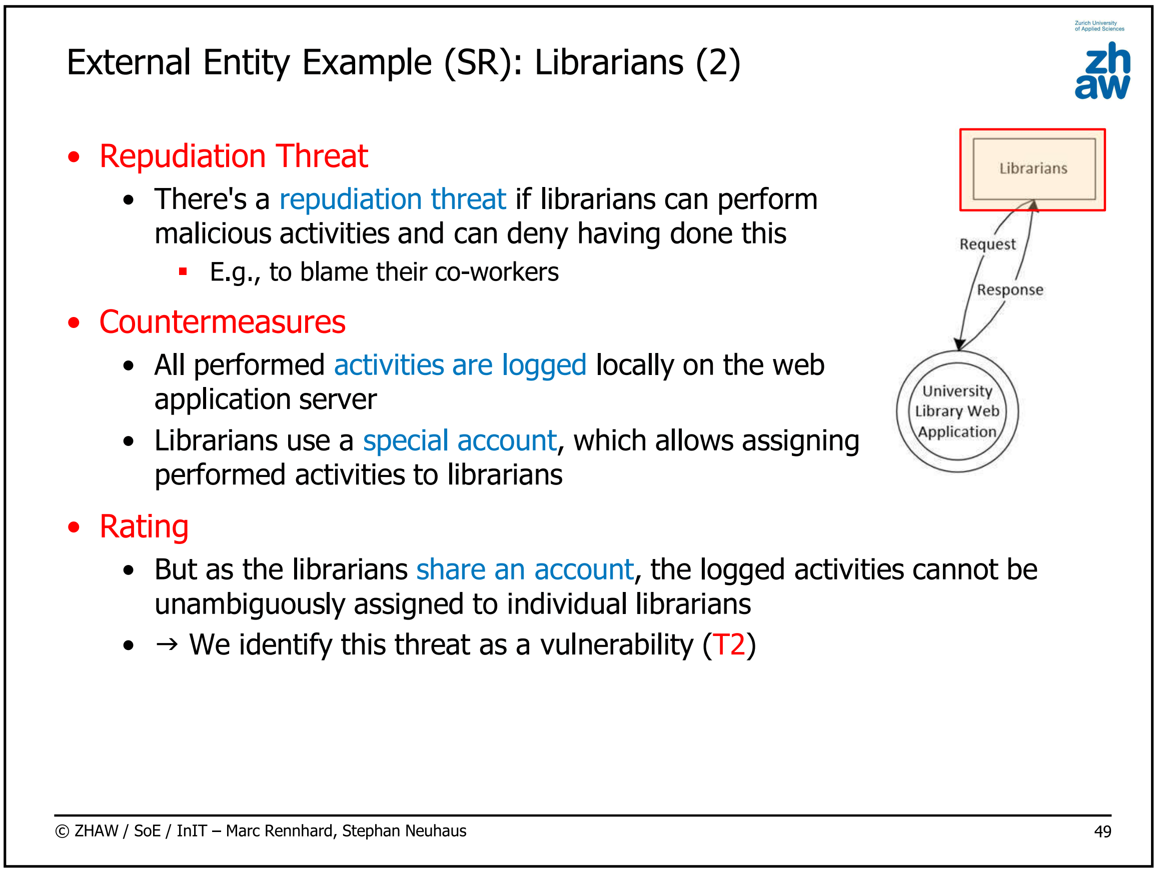
## External Entity Example (SR): Librarians (1)

- 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)



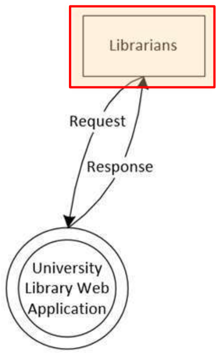
**Spoofing Threats**

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.



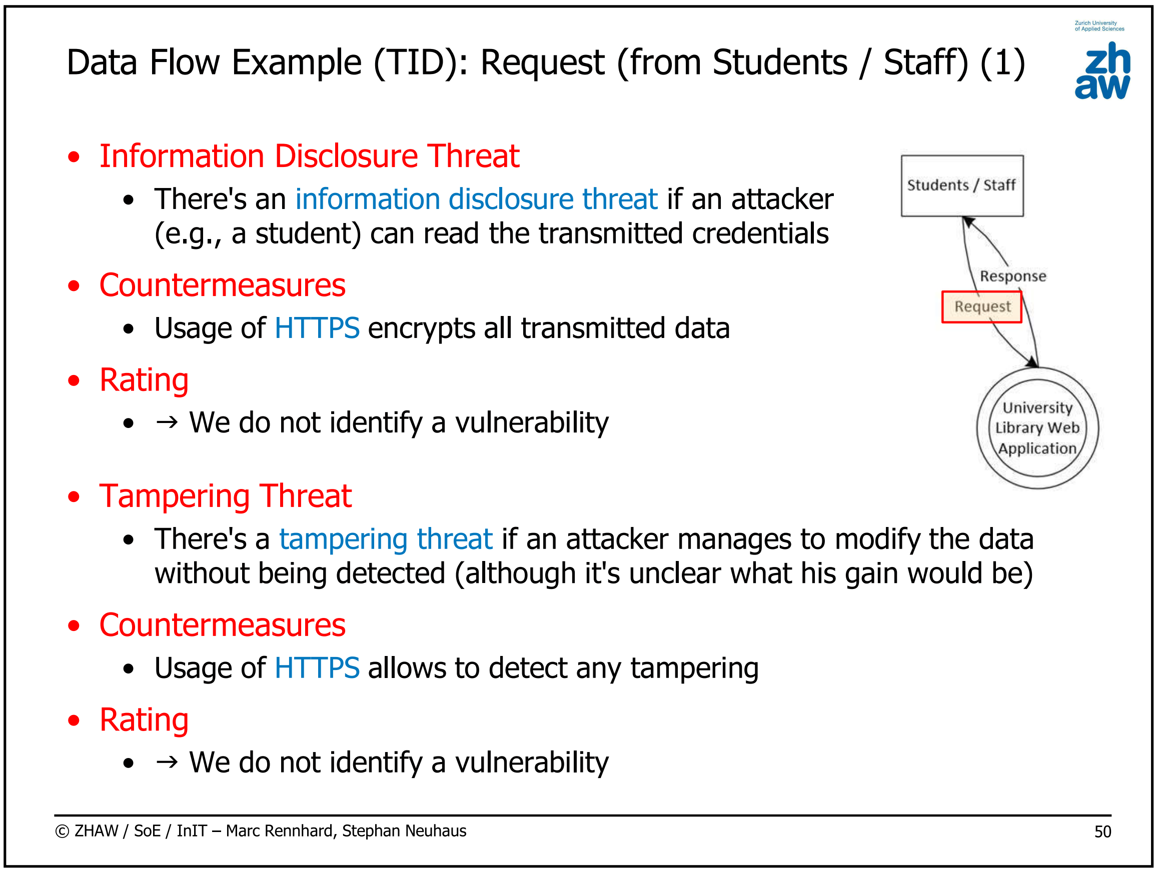
## External Entity Example (SR): Librarians (2)

- 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)



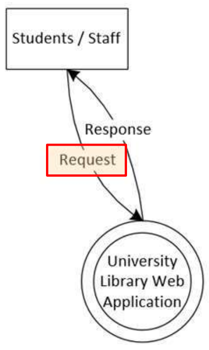
**Assets and Vulnerabilities**

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)

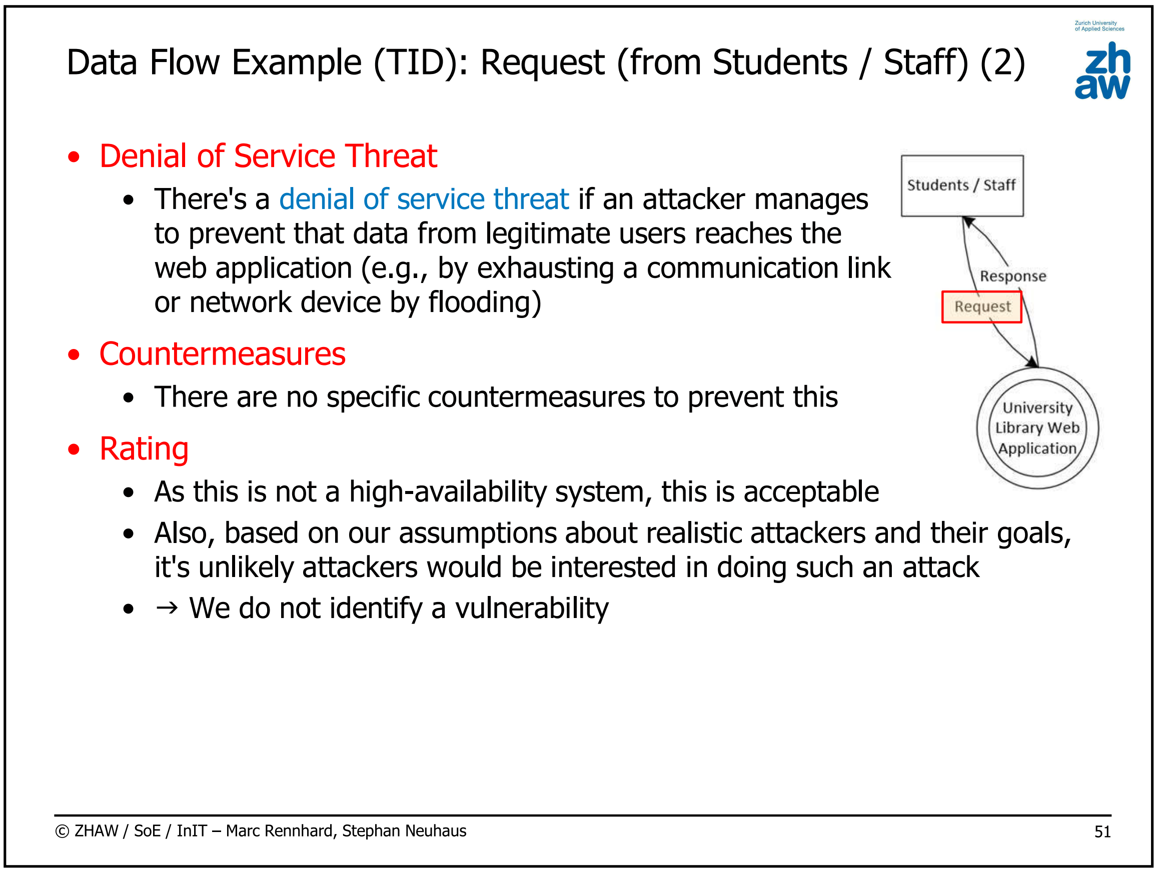
- Information Disclosure Threat
  - There's an information disclosure threat if an attacker (e.g., a student) can read the transmitted credentials
- Countermeasures
  - Usage of HTTPS encrypts all transmitted data
- Rating
  - → We do not identify a vulnerability



- 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

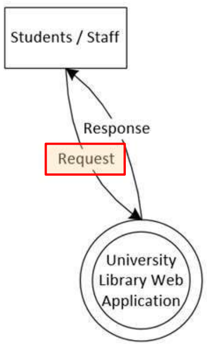
**HTTPS**

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.