

9. Security Requirements Engineering and Threat Modeling – Part 2

Prof. Dr. Marc Rennhard

Institut für angewandte Informationstechnologie InIT

ZHAW School of Engineering

rema@zhaw.ch

Attack Trees

Attack Trees (1)

- **Attack trees** (sometimes called threat trees) is a further method to think about threats on 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 “anything”, 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 less based on a specific technical system description

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

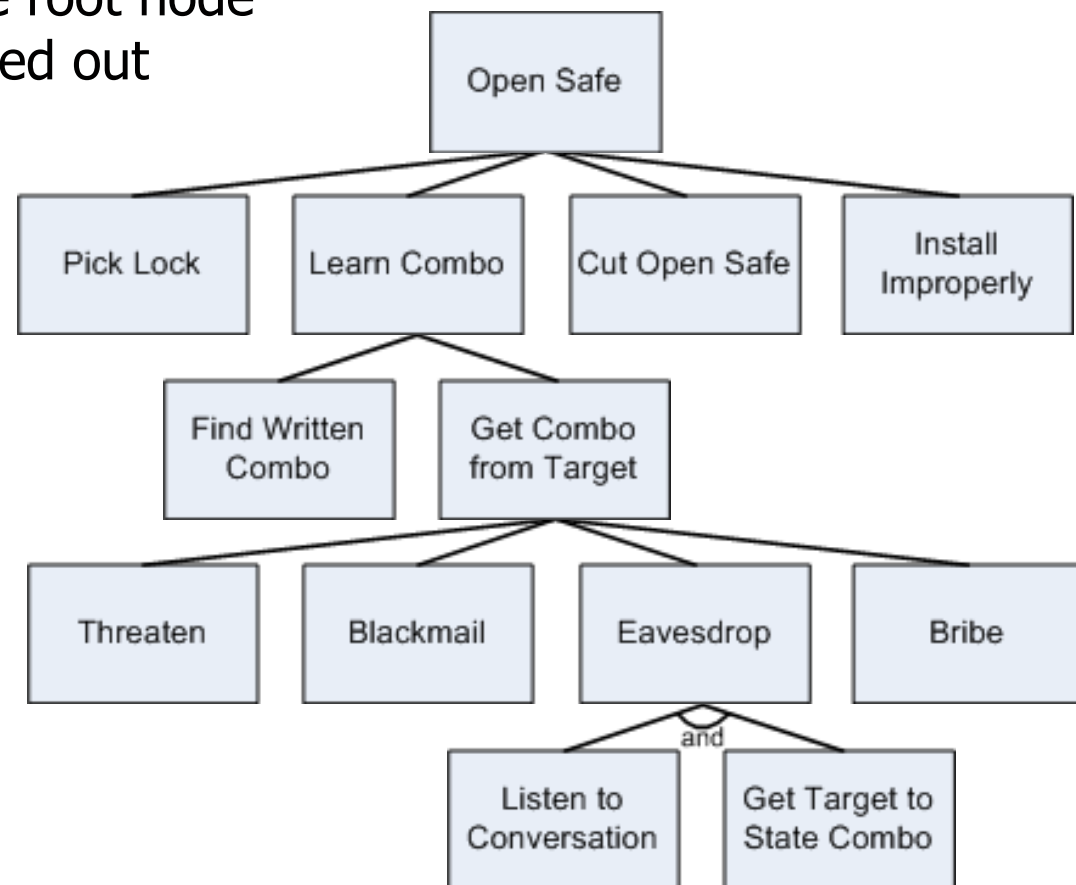
Attack Trees (3)

- 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" or "getting credit card from user via social engineering"
 - The different ways are illustrated in a tree with the goal at the root
- 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

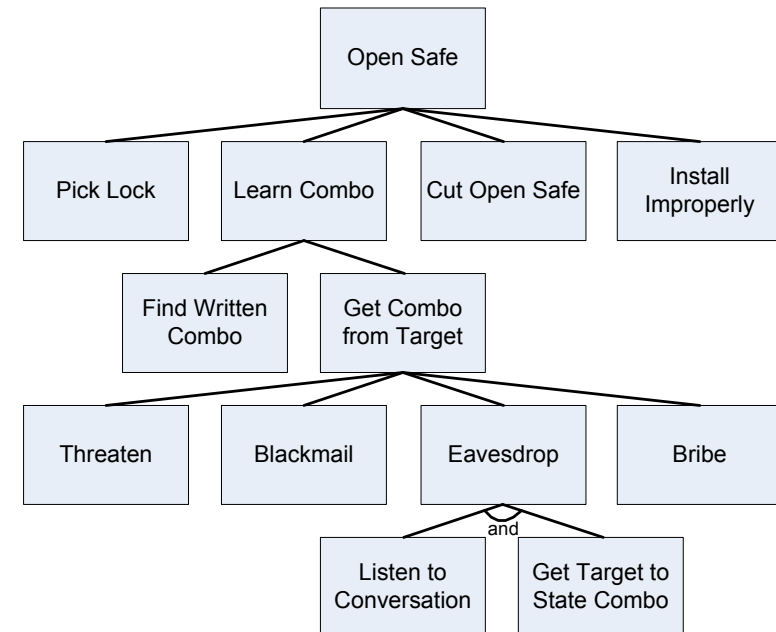
Attack tree to achieve the goal “Open Safe” (root node)

- Think about ways to achieve the goal and write them down 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 and so on
- 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



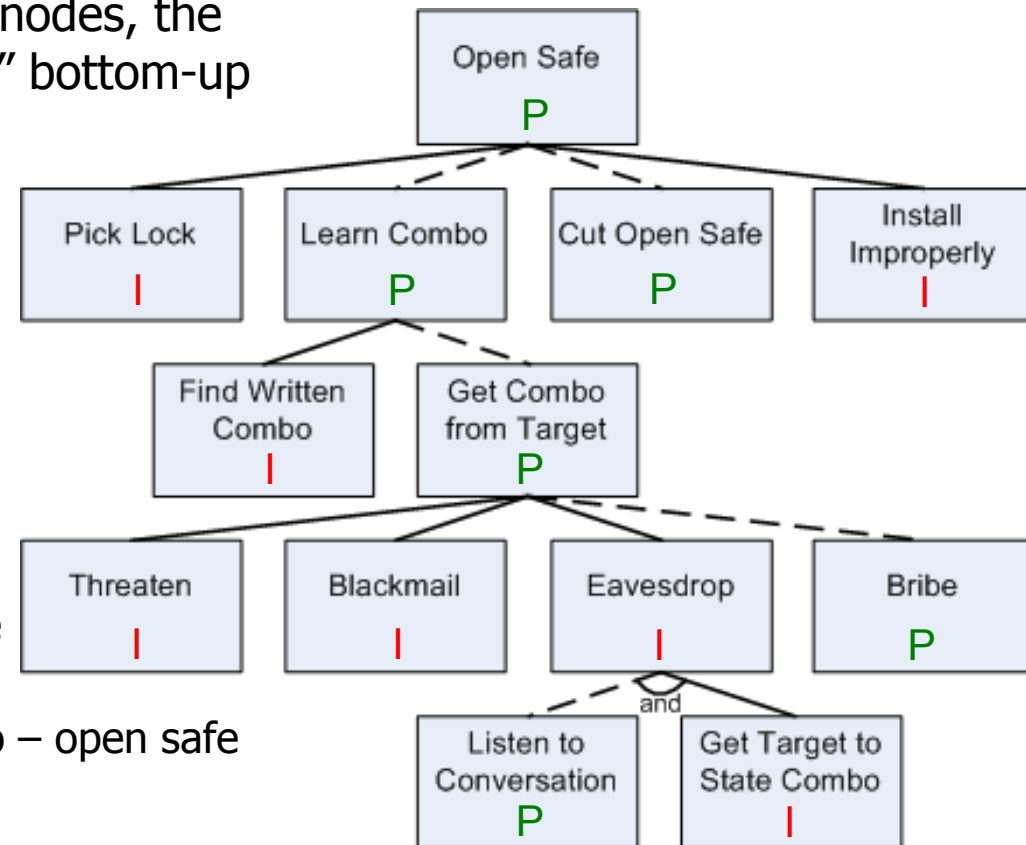
Attack Trees – AND / OR Nodes

- 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...) 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



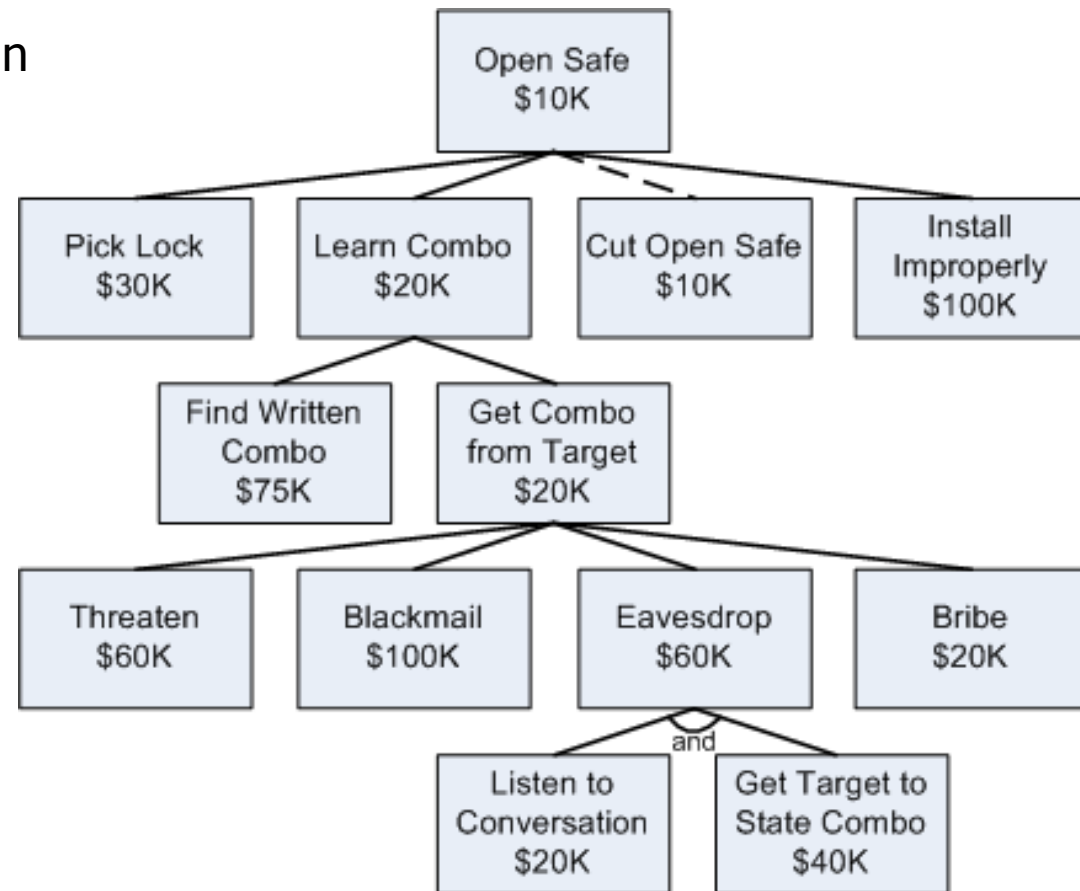
Attack Trees – Opening a Safe – Possible Attacks

- Assigning values to the nodes allows further analyses
 - 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



Attack Trees – Opening a Safe – Cheapest Attack

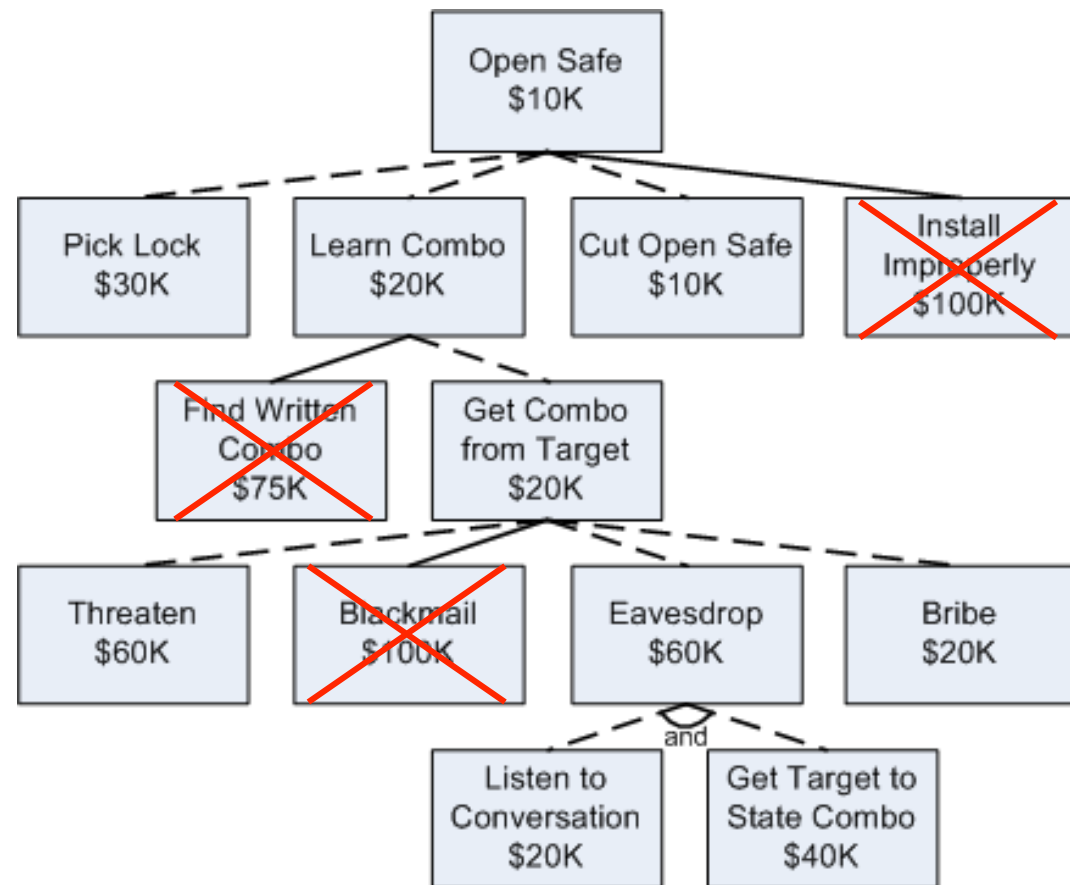
- Assigning monetary values to the leaf nodes allows making statements about the cost of attacks
 - The other nodes are again computed “bottom-up”
 - AND nodes get the sum of their children
 - OR nodes get the value of their cheapest child
 - The cheapest attack has dashed lines and costs just \$10K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



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

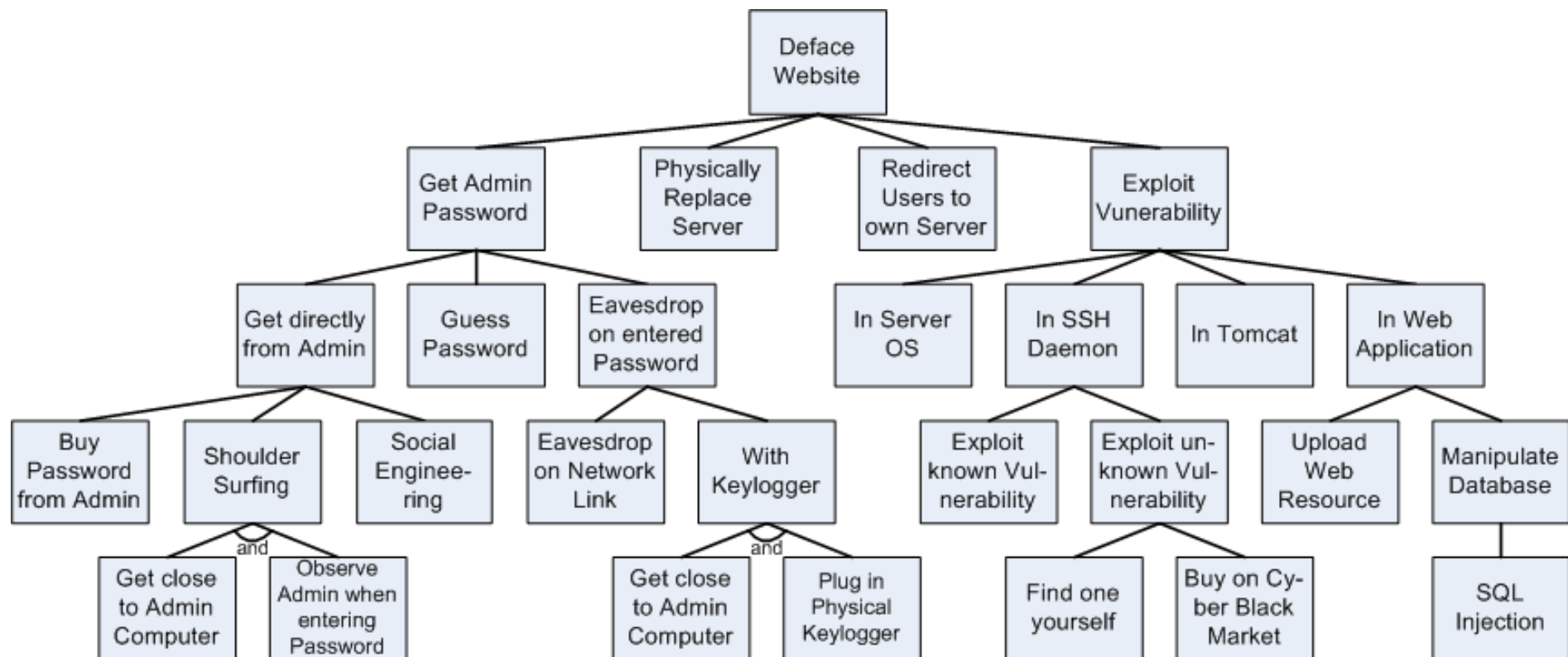
- **Monetary values** can also be used to determine all attacks that are, e.g., cheaper than \$70K (e.g. the amount of money in the safe)

- 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**



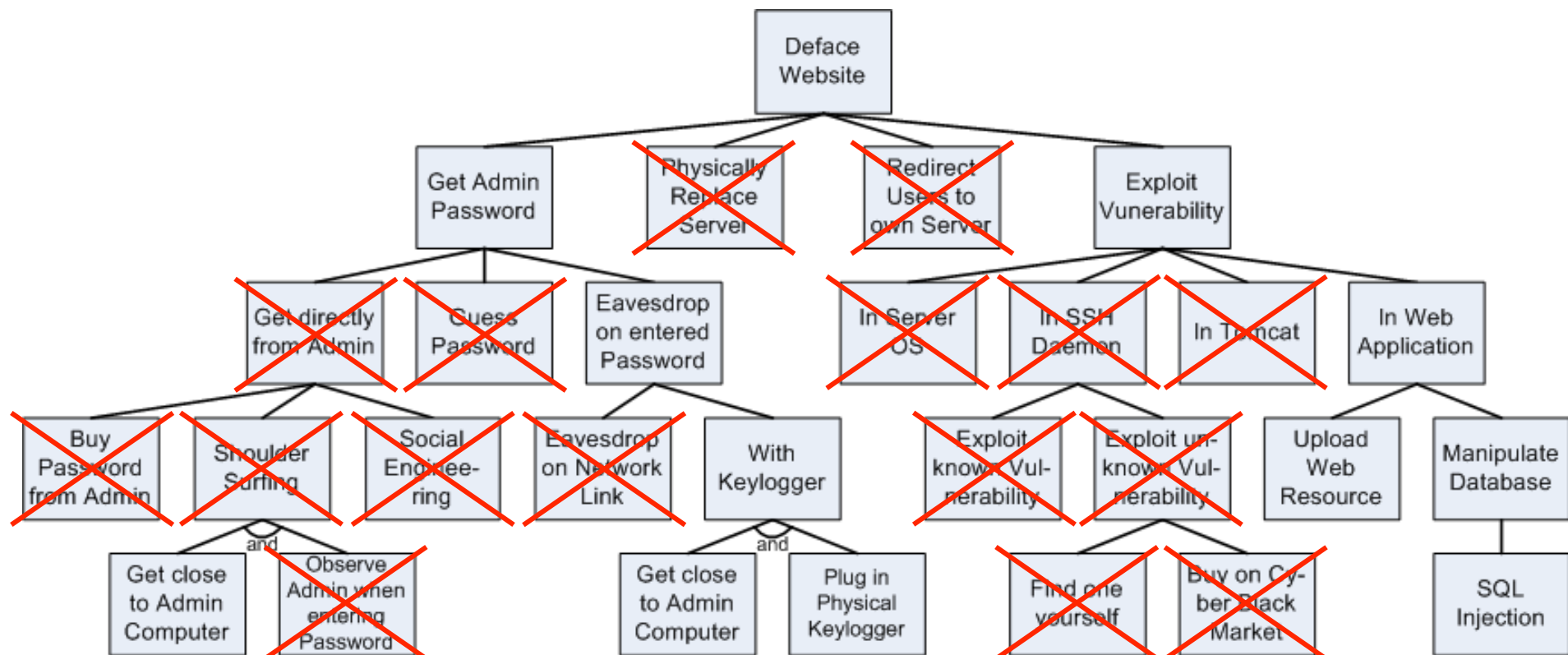
The University Library Application – Attack Tree (1)

- An example (incomplete) Attack Tree for the University Library Application, which considers the attack goal “Deface Website”



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



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 admin PCs are readily accessible for many internal employees 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 in 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

Know Your Enemy

Who may Attack you? (1)

- So far, we haven't made **assumptions about the attacker**, but focused on the attacks / threats themselves
- However, making assumptions about **who may attack you** is important to consider **realistic** attack scenarios and to pick appropriate countermeasures



Who may Attack you? (2)

- Understanding who the **attackers** and their **goals** may be is important for various reasons:
 1. It allows to estimate their **skills and resources**
 - Note that money can buy many skills
 2. It allows to make reasonable assumptions about **what attacks** may likely occur
 3. And it allows to make the **right security decisions** and spend the right amount of money at the right places
- This **very much depends on your application**, e.g. do you develop
 - An e-banking application?
 - May likely attract organized cybercriminals that want to make money
 - A “find-your-partner” platform?
 - May still attract cybercriminals that want to steal sensitive data to blackmail the platform owners or end users
 - The ZHAW Stundenplan application?
 - May attract students that want to have some fun

Who may Attack you – Examples

- Example 1: You develop a web application for your **local sports club**
 - You most likely won't be an the target of determined cybercriminals
 - Still, **attackers may be interested in using your system** as a bot or to host illegal content
 - So some attackers may probe your systems for some easy-to-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
- Example 2: You develop an application that stores **financial information** (e.g. credit cards, bank accounts...)
 - You will most likely be targeted by **well-funded organized cybercriminals**
 - They may invest **significant efforts** to probe your application manually
 - They may directly **target users or system operators** (e.g. via targeted social engineering attacks)
 - This certainly motivates the definition of security requirements that resist such sophisticated attacks

The University Library Application – Exercise

Who may Attack you? (1)

Consider the University Library Application: Identify realistic attackers and their corresponding goals

The University Library Application – Exercise

Who may Attack you? (2)

When constructing the attack tree, we identified the following threats for which no security requirements existed:

- “Getting admin password” by installing a physical keylogger
- “Exploit vulnerability” in the web application

Now that you have identified “realistic attackers”, do you still think these are realistic threats?

Documenting Security Requirements

Documenting Security Requirements (1)

- Documenting Security Requirements is **important as a basis** for all subsequent security activities
- As often with documentation in software projects, **the employed method / template is less important** than the fact that documentation is done at all!
 - Wikis, spread sheets, custom developed tools etc. are all OK as long they are used
 - There is no widely accepted standard about how security requirements should be documented
- Remember that security requirements engineering is usually an iterative process, so documentation has to be **updated during each iteration**
 - With too much documentation, it is likely that not all will be updated correctly

Documenting Security Requirements (2)

- First, we look at how to **document the actual security requirements**, as this is the basis for subsequent security activities
- General rules for documenting security requirements:
 - They should ideally be expressed in **a single sentence**
 - If one sentence is not enough, the requirement may contain too many details about how it should be implemented
 - Remember: security requirements are about what should be done, not how it should be done
 - They should be **precise**
 - They should be **clearly understandable** by project members and not leave too much room for interpretation
 - Optionally, security requirements may be **prioritized and/or categorized**
 - If there are many similar requirements, try to **combine** them
 - When “writing down” a requirement, check whether it motivates obvious other security requirements

The University Library Application – Collection of the Security Requirements Identified

Users and librarians can access the web application server only via cryptographically protected channels (e.g. HTTPS / TCP port 443)

All performed actions are logged locally on the web application server

All data received from users should be considered non-trusted and should first be validated before processed further

When TLS is used, servers are authenticated using Extended Validation (EV) certificates

All queries are logged on the database server

The SSH daemons log successful logins and login failures

Use a technical database user with minimal privileges

Direct access to the web and database servers for configuration and log monitoring is only allowed to administrators and only via SSH

Use individual accounts for librarians with dedicated passwords

The web application must employ an authorization mechanism, Authorization must be checked on every single request (complete mediation)

Administrators must at least have security clearance "medium" (employed ≥ 5 years, regular background checks)

Enforce a minimal password quality (length, character mix, compare with dictionary...)

Accessing the database is only allowed via prepared statements

Website defacement can be prevented by employing appropriate security controls (prepared statements, input validation, no write access rights to web resources...)

The University Library Application – Security Requirements Documentation

Req.	Description
R-01	Accessing the web application is only allowed via cryptographically protected channels using HTTPS with EV server certificates
R-02	Direct access to the web application and database servers for configuration and log monitoring is only allowed to administrators with at least security clearance “medium” and only via SSH
R-03	All performed actions are logged locally on the web application server, including the user that performed the action
R-04	All queries are logged on the database server
R-05	Successful/failed logins must be logged by the web application, the DBMS, and the SSH deamons
R-06	Only personal user accounts are used
R-07	Passwords must have at least the following complexity: ≥ 10 characters, at least one digit, and at least one special character
R-08	Accessing the database is only allowed via prepared statements
R-09	All data received from users should be considered non-trusted and should first be validated before processed further
R-10	Use a technical database user with minimal privileges
R-11	The web application must employ an authorization mechanism, authorization must be checked on every single request (complete mediation)
R-12	All processes should only get the minimal access rights necessary
R-13	Cryptographically protected channels must employ at least 128-bit keys for symmetric encryption and must not use RC4 for symmetric encryption or MD5 for hash/MAC computation

Documenting Security Requirements (3)

- Beyond documenting the actual security requirements, one should also **document the threats** and map the requirements to the threats
 - This helps to understand the necessity of the individual security requirements and to check what threats are considered
- A complete documentation should therefore include:
 - A **complete list of threats** that are considered realistic
 - Should provide enough information so they can at least be understandable by project members
 - A **complete list of security requirements**, concise and precise
 - **Mappings** between them
 - Shows which threats are mitigated by which security requirement and vice versa
 - Also shows whether all threats are in fact mitigated with at least one security requirement
 - All related **artifacts** (data flow diagrams, attack trees...)

The University Library Application – Complete Documentation

Threat	Description	Reqs.
T-01	Spoofing librarians by learning/guessing the credentials of librarians	R-07
T-02	Librarians can deny having performed malicious activities	R-06
T-03	Accessing arbitrary data via SQL injection	R-08 R-09
T-04	Users may elevate their privileges in the web application by exploiting a faulty access control mechanism	R-11
T-05	Defacing the website by exploiting vulnerability in the web application (SQL injection, upload web resource...)	R-08 R-09

Req.	Description	Threats
R-06	Only personal user accounts are used	T-02
R-07	Passwords must have at least the following complexity: ≥ 10 characters, at least one digit, and at least one special character	T-01
R-08	Accessing the database is only allowed via prepared statements	T-03 T-05
R-09	All data received from users should be considered non-trusted and should first be validated before processed further	T-03 T-05
R-11	The web application must employ an authorization mechanism, authorization must be checked on every single request (complete mediation)	T-04

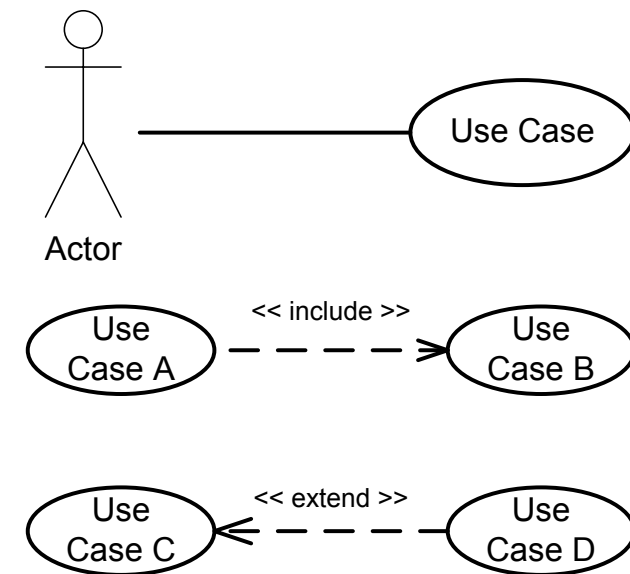
Further Methods: Abuse Cases

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 → **abuse cases**
 - The threats can even be directly drawn in the diagram

- Reminder of UML use case notation

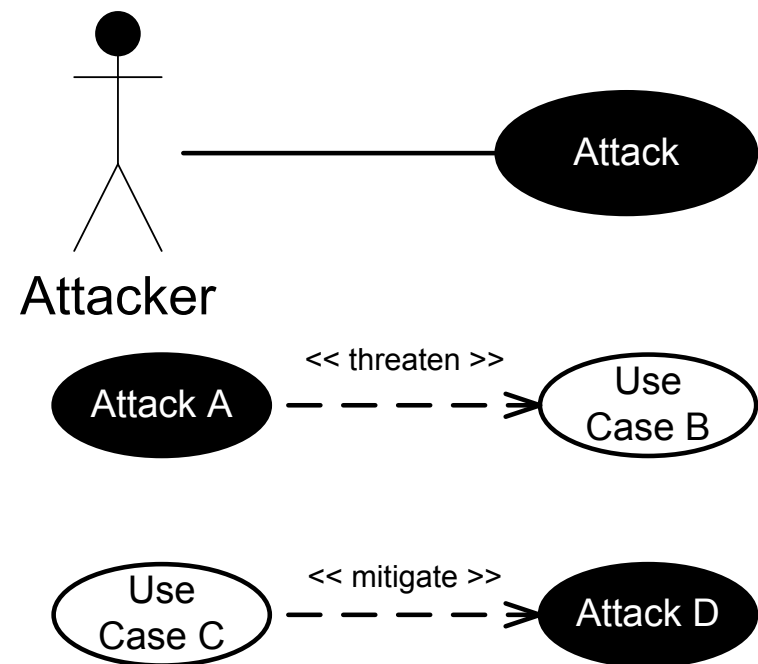
- An **actor** interacts with a **use cases**, which can by any function of the system
- Use case A **includes** use case B, meaning that whenever A is executed, B is executed as part of it
- Use case D **extends** use case C, meaning that when executing C, D may be executed as part of it (depending on conditions)



Abuse Cases and UML Diagrams (2)

Abuse case notation introduces **further diagram types**:

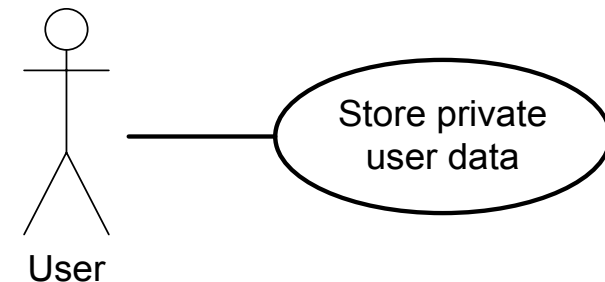
- Not an official UML standard, just a proposal how it can be extended
- We have a new type of actor, the **attacker** (drawn as inverted actor)
- The attacker **executes attacks** (drawn as inverted use case)
- An attack A **threatens** a use case B
- The use case C can **mitigate** attack D



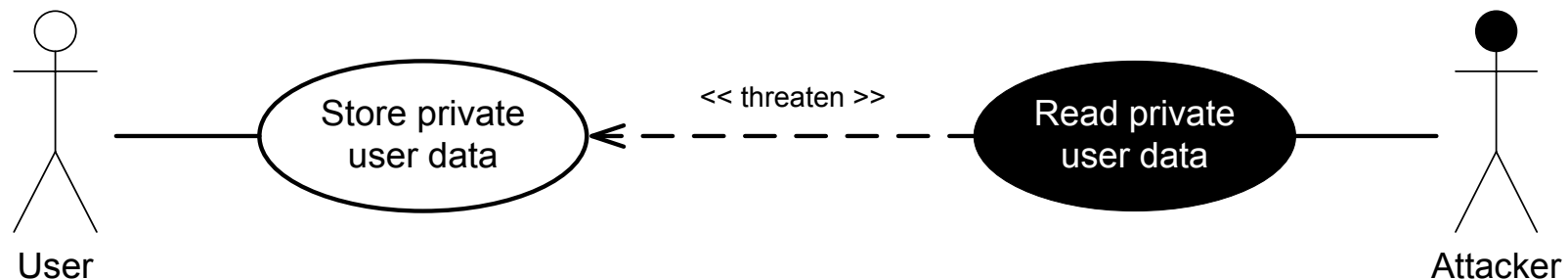
Abuse Cases – Simple Example (1)

- 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**

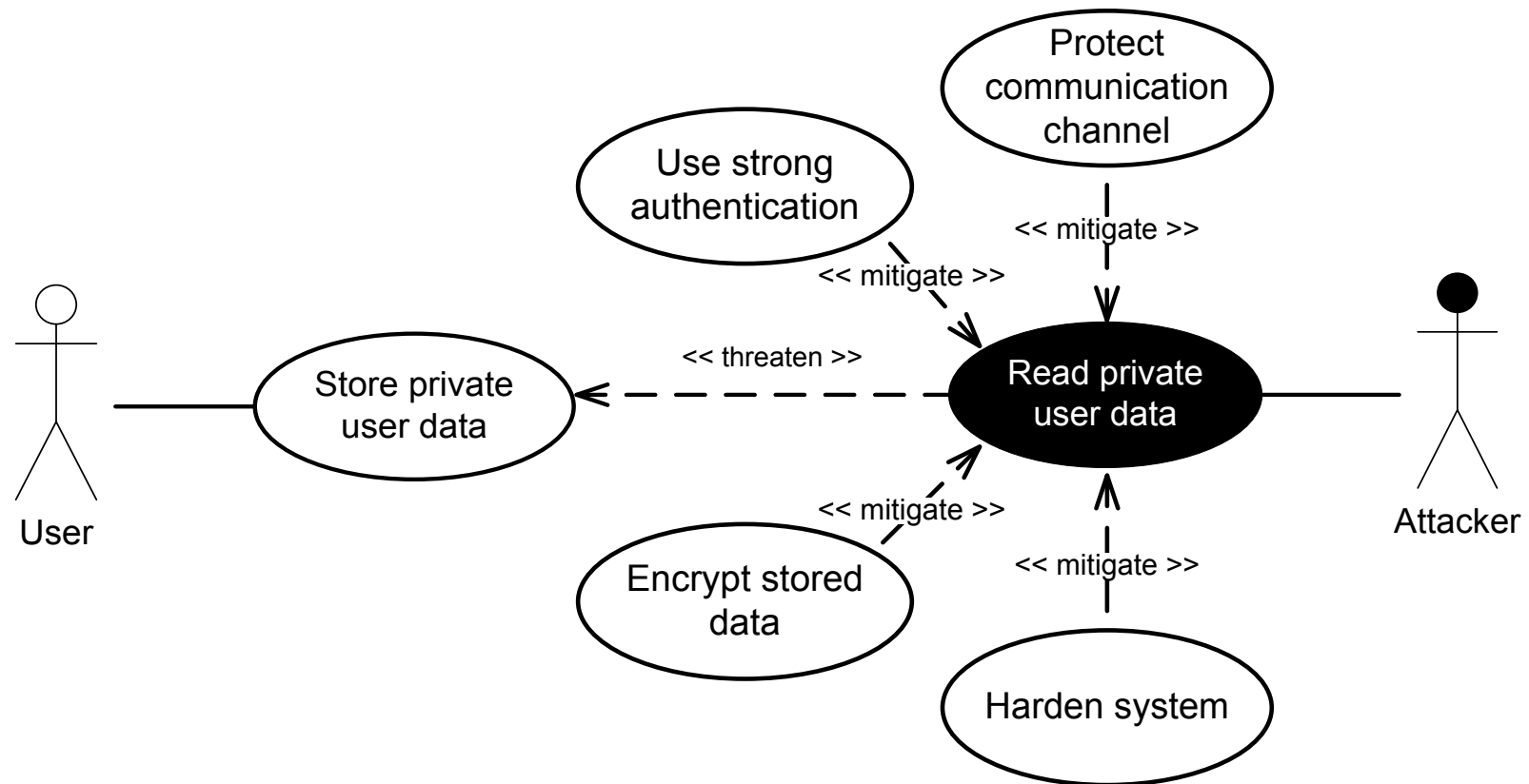


- If this use case is used to think about attacks, then we can identify an **attacker that wants to read the private user data**
 - So we have an attack (abuse case) “**read private user data**” that **threatens** the stored private user data



Abuse Cases – A Simple Example (2)

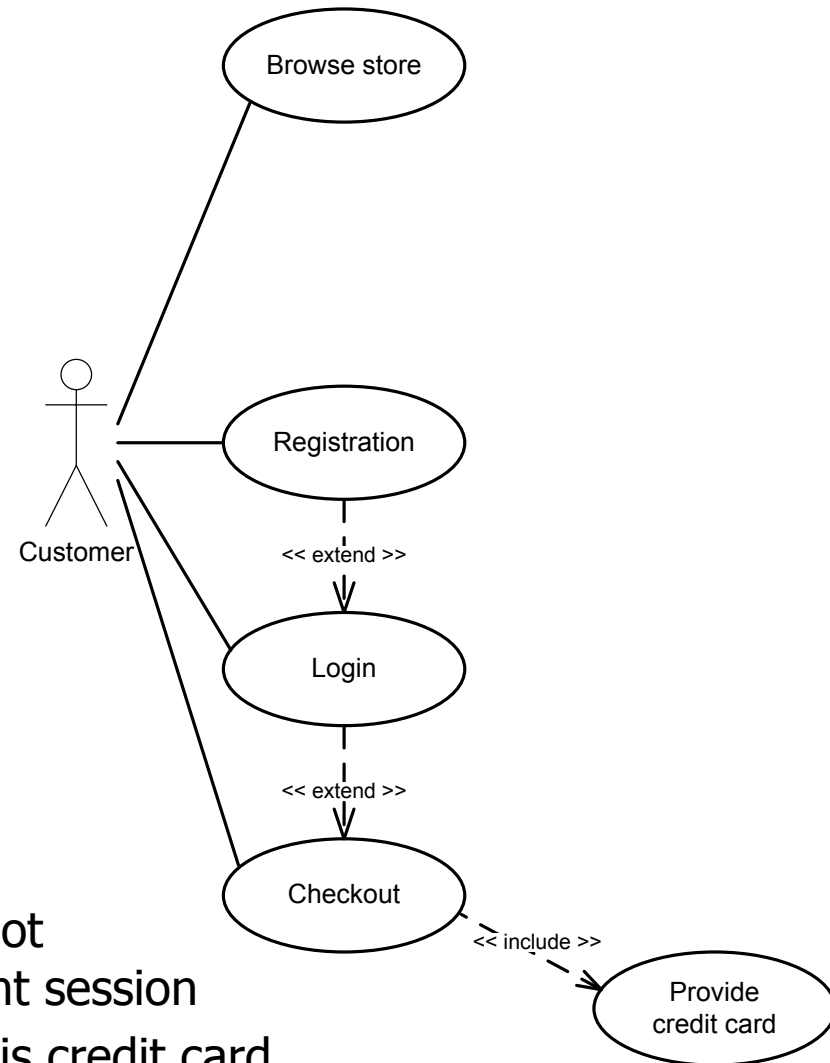
- Based on this scenario, we can now think about security requirements (drawn as use cases) **that can mitigate the attack**



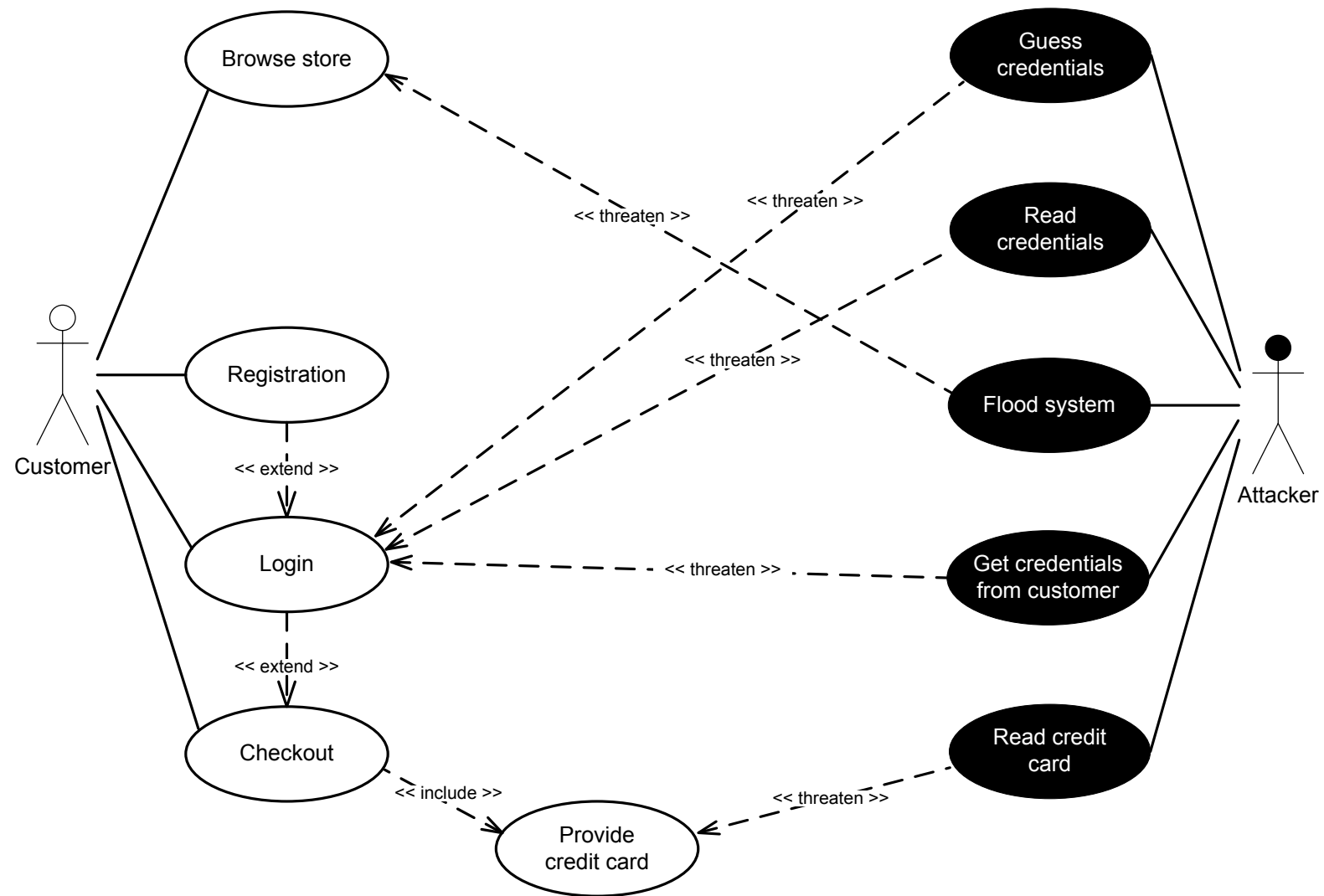
A more Elaborate Example – Basic Use Cases

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

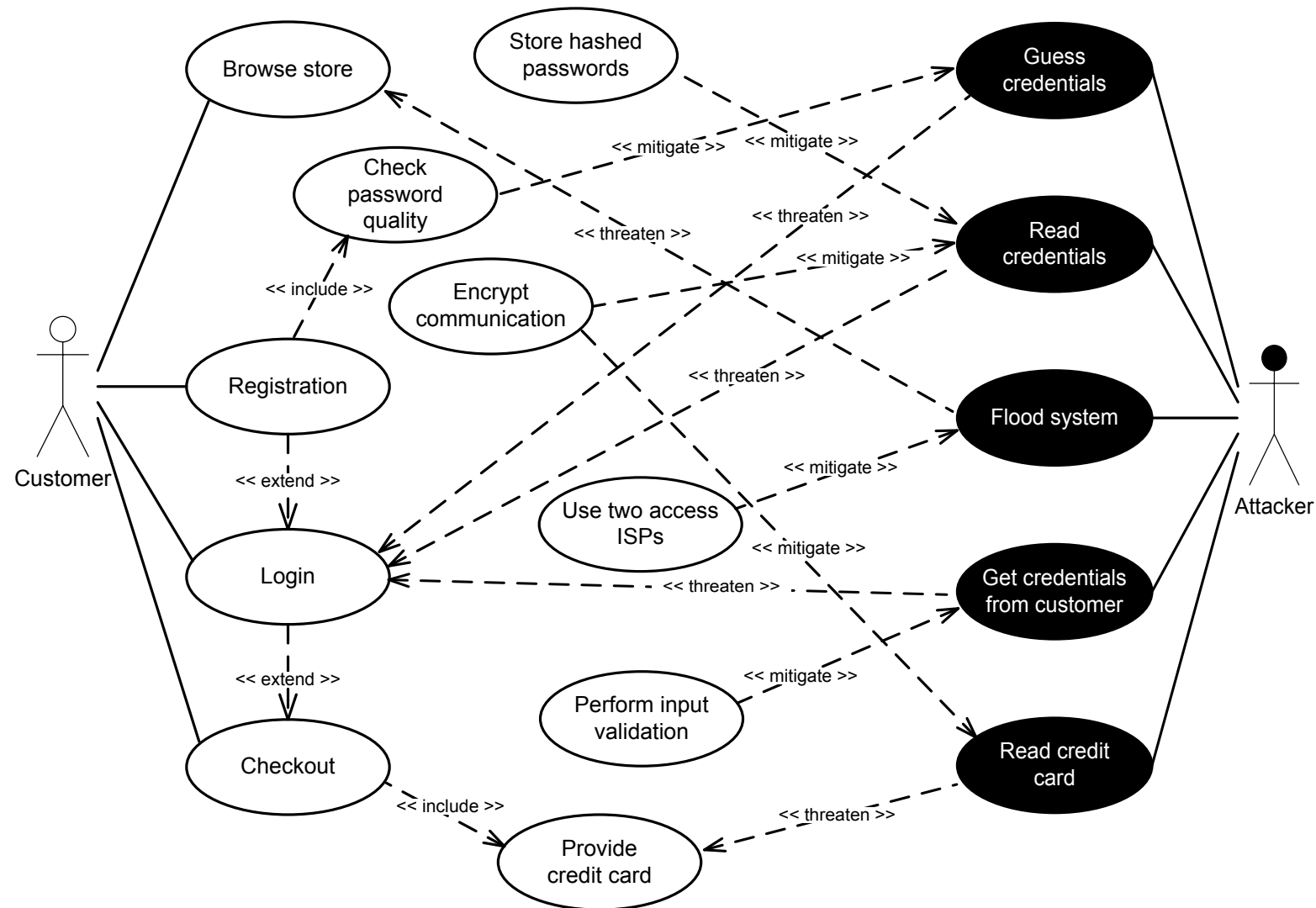
- **Browsing** the store
- Customer **registration**
- **Login**
 - May require registration if customer has not registered before
- **Checkout**
 - May require login if customer has not performed a login during the current session
 - Requires the customer to provide his credit card



A more Elaborate Example – Abuse Cases



A more Elaborate Example – Security Requirements / Mitigation Use Cases



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 when the developer is not aware of the corresponding threats

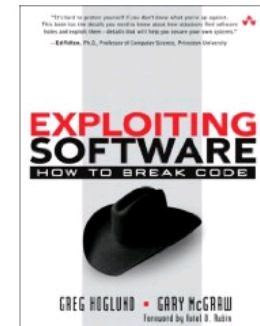
Abuse Cases and UML Use Cases Diagrams – Discussion

- 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

Further Methods: 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 – Overview of Attacks


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:

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	
HasMember		118	Data Leakage Attacks		1000

[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)

Categories

Attack Patterns

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. Successful 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 parenthesis, 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

Summary

- Security Requirements Engineering means **determining security requirements early** in the development lifecycle
- Security requirements provide the **basis for all subsequent security-related activities** during the development lifecycle
 - Designing and implementing security controls, performing security tests
- 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 find out whether the specified security requirements or security controls are **appropriate**
- Security requirements engineering is basically a creative and “not very scientific” process, but **several methods help** to perform threat modeling
 - Data flow diagrams, STRIDE, attack trees, abuse cases, attack patterns