SWEN 6301 Software Construction

Module 10: System Dependability and Security

Ahmed Tamrawi

Dependable Systems

Topics Covered

- Dependability properties
- Sociotechnical systems
- Redundancy and diversity
- Dependable processes
- Formal methods and dependability

System Dependability

- For many computer-based systems, the most important system property is the **dependability of the system**.
- The dependability of a system reflects the **user's degree of trust in that system**. It reflects the extent of the user's confidence that it will operate as users expect and that it will not 'fail' in normal use.
- Dependability covers the related systems attributes of **reliability**, **availability** and **security**. These are all **inter-dependent**.

Importance of Dependability

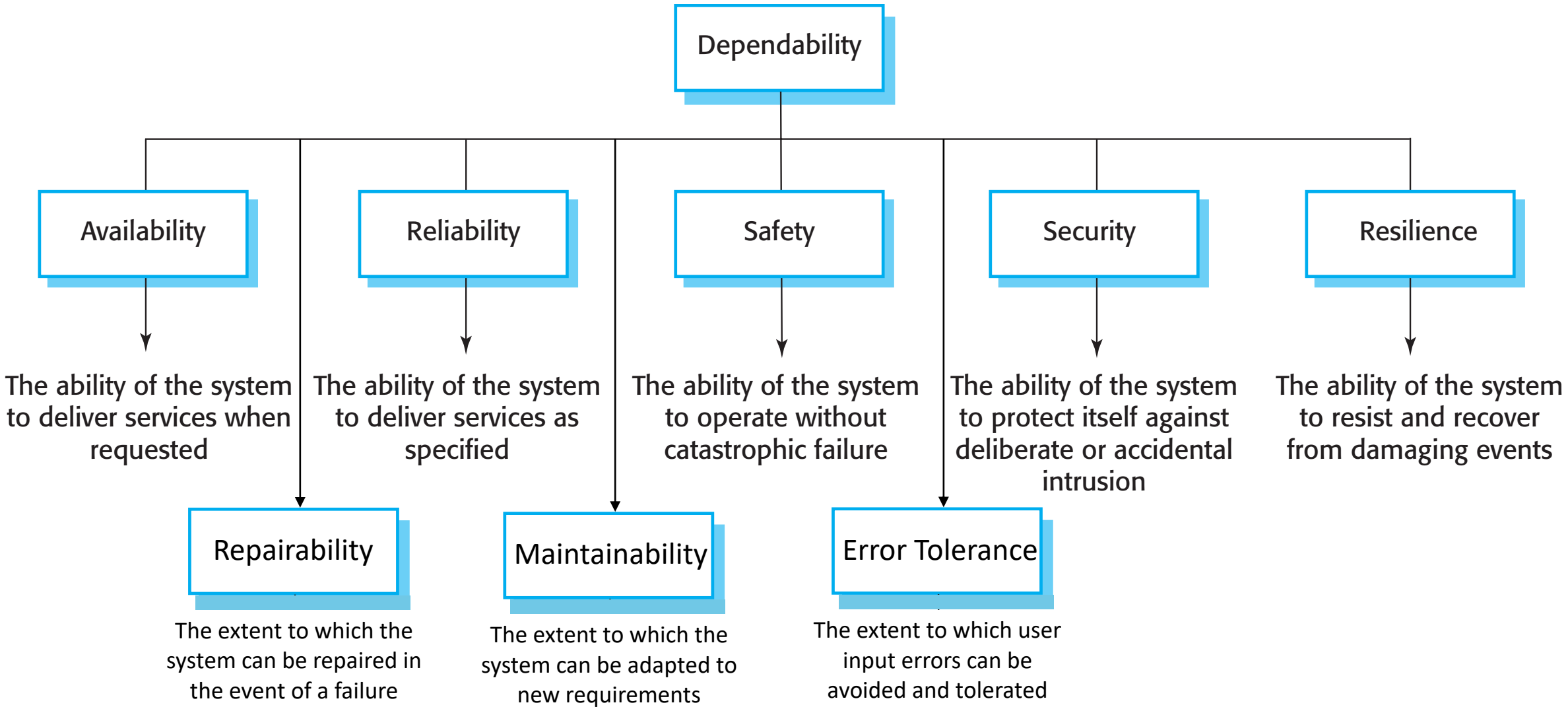
- **System failures** may have widespread effects with large numbers of people affected by the failure.
- Systems that are not dependable and are unreliable, unsafe or insecure may be rejected by their users.
- The costs of system failure may be **very high** if the failure leads to *economic losses or physical damage*.
- Undependable systems may cause **information loss** with a high consequent recovery cost.

Causes of Failure

- **Hardware failure**
  - Hardware fails because of design and manufacturing errors or because components have reached the end of their natural life.
- **Software failure**
  - Software fails due to errors in its specification, design or implementation.
- **Operational failure**
  - Human operators make mistakes. Now perhaps the largest single cause of system failures in socio-technical systems.

Dependability Properties

Principal Dependability Properties



Dependability Attribute Dependencies

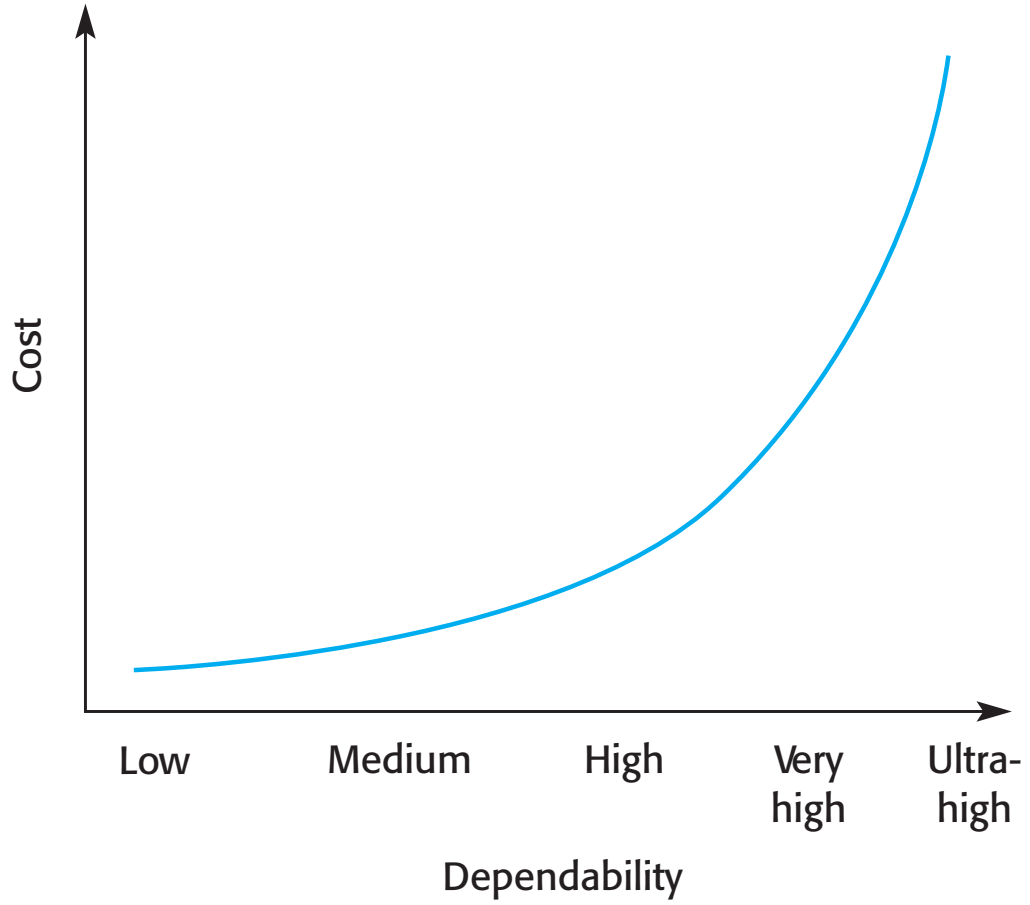
- **Safe system** operation depends on the system being **available** and operating **reliably**.
- A system may be **unreliable** because its data has been corrupted by an external **attack**.
- Denial of service **attacks** on a system are intended to make it **unavailable**.
- If a system is infected with a **virus**, you cannot be **confident** in its **reliability** or **safety**.

Dependability Achievement

- Avoid the introduction of **accidental errors** when developing the system.
- Design V & V processes that are effective in discovering residual errors in the system.
- Design systems to be **fault tolerant** so that they can continue in operation when faults occur
- **Design protection mechanisms** that guard against external attacks.
- Configure the system **correctly** for its operating environment.
- Include system capabilities to **recognize** and **resist cyberattacks**.
- Include **recovery mechanisms** to help restore normal system service after a failure.

Dependability Costs

- Dependability costs tend to increase **exponentially** as increasing levels of dependability are required.
- There are two reasons for this:
  - The use of more **expensive development techniques and hardware** that are required to achieve the higher levels of dependability.
  - The **increased testing and system validation** that is required to convince the system client and regulators that the required levels of dependability have been achieved.



Dependability Economics

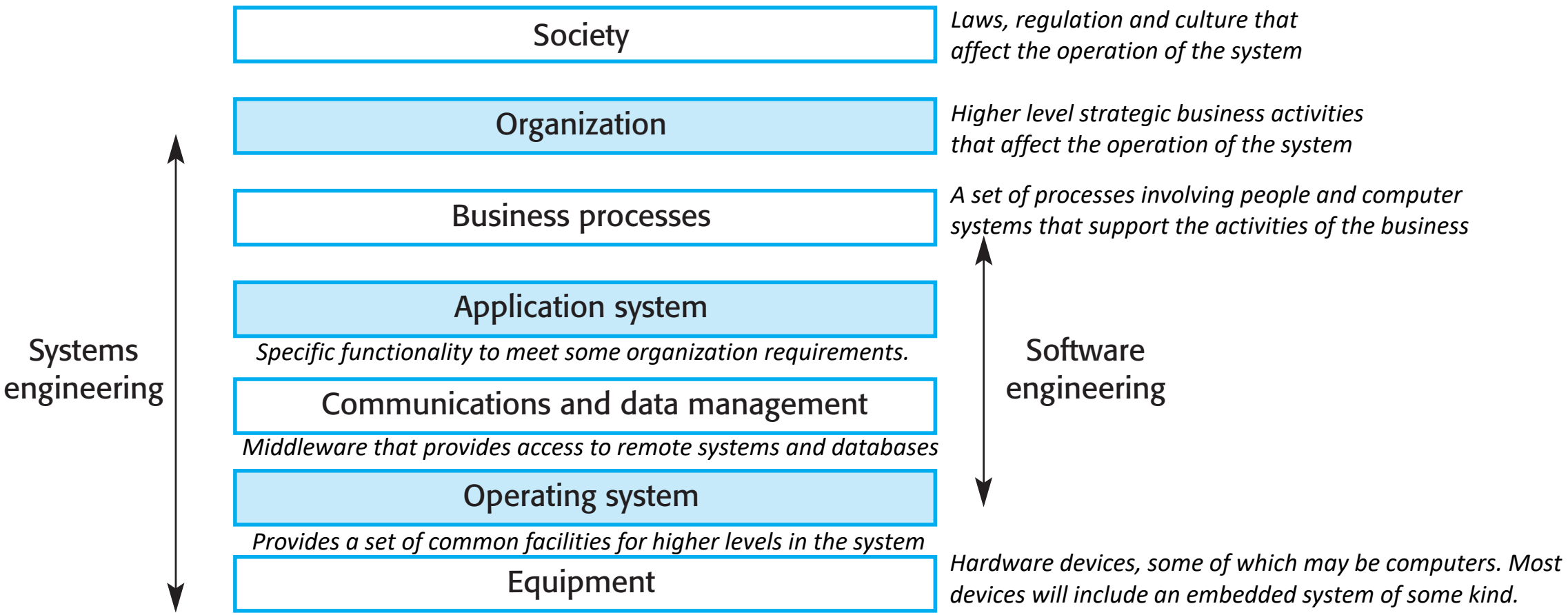
- Because of very **high costs of dependability achievement**, it may be more cost effective to accept untrustworthy systems and pay for failure costs
- However, this depends on social and political factors. A reputation for products that can’t be trusted may lose future business
- Depends on system type - for business systems in particular, modest levels of dependability may be adequate

Sociotechnical Systems

Systems and Software

- Software engineering is not an isolated activity but is part of a broader systems engineering process.
- Software systems are therefore not isolated systems but are essential components of broader systems **that have a human, social or organizational purpose.**
- Example
  - The **wilderness weather system** is part of broader weather recording and forecasting systems
  - These include **hardware and software**, **forecasting processes**, **system users**, the **organizations** that depend on weather forecasts, etc.

Sociotechnical Systems Stack



Holistic System Design

- There are interactions and dependencies between the layers in a system and changes at one level **ripple through the other levels**
  - **Example**: Change in regulations (society) leads to changes in business processes and application software.
- For dependability, a **systems perspective is essential**
  - Contain software failures within the enclosing layers of the STS stack.
  - Understand how faults and failures in adjacent layers may affect the software in a system.

Regulation and Compliance

- The general model of economic organization that is now almost universal in the world is that privately owned companies offer goods and services and make a profit on these.
- To ensure the safety of their citizens, most governments regulate (limit the freedom of) privately owned companies so that they must follow certain standards to ensure that their products are safe and secure.

Regulated Systems

- Many critical systems are **regulated systems**, which means that their use must be **approved by an external regulator before the systems go into service**.
  - Nuclear systems
  - Air traffic control systems
  - Medical devices
- A safety and dependability case has to be approved by the regulator. Therefore, critical systems development has to create the evidence to convince a regulator that the system is dependable, safe and secure.

Safety Regulation

- Regulation and compliance (following the rules) applies to the sociotechnical system as a whole and not simply the software element of that system.
- Safety-related systems may have to be certified as safe by the regulator.
- **To achieve certification**, companies that are developing safety-critical systems have to produce an **extensive safety case that shows that rules and regulations have been followed**.
- It can be as **expensive develop the documentation for certification** as it is to develop the system itself.

Redundancy and Diversity

Redundancy and Diversity

- **Redundancy:** Keep more than a single version of critical components so that if one fails then a backup is available.
- **Diversity:** Provide the same functionality in different ways in different components so that they will not fail in the same way.
- **Redundant and diverse components** should be **independent** so that they will not suffer from **‘common-mode’** failures
  - **For example**, components implemented in different programming languages means that a compiler fault will not affect all of them.

Diversity and Redundancy examples

- **Redundancy**. Where availability is critical (e.g. in e-commerce systems), companies normally keep backup servers and switch to these automatically if failure occurs.
- **Diversity**. To provide resilience against external attacks, different servers may be implemented using different operating systems (e.g. Windows and Linux)

Process Diversity and Redundancy

- Process activities, such as validation, should not depend on a single approach, such as testing, to validate the system.
- Redundant and diverse process activities are important especially for verification and validation.
- Multiple, different process activities the complement each other and allow for cross-checking help to avoid process errors, which may lead to errors in the software.

Problems with Redundancy and Diversity

- Adding diversity and redundancy to a system inc**reases the system complexity**.
- This can **increase the chances of error** because of unanticipated interactions and dependencies between the redundant system components.
- Some engineers therefore advocate simplicity and extensive V & V as a more effective route to software dependability.
- Airbus FCS architecture is redundant/diverse; Boeing 777 FCS architecture has no software diversity

Dependable Processes

Dependable Processes

- To ensure a minimal number of software faults, it is important to have a well-defined, repeatable software process.
- A well-defined repeatable process is one that does not depend entirely on individual skills; rather can be enacted by different people.
- Regulators use information about the process to check if good software engineering practice has been used.
- For fault detection, it is clear that the process activities should include significant effort devoted to verification and validation.

Dependable Process Characteristics

- **Explicitly defined**
  - A process that has a defined process model that is used to drive the software production process. Data must be collected during the process that proves that the development team has followed the process as defined in the process model.
- **Repeatable**
  - A process that does not rely on individual interpretation and judgment. The process can be repeated across projects and with different team members, irrespective of who is involved in the development.

Attributes of Dependable Processes

Process Characteristic	Description
Auditable	The process should be understandable by people apart from process participants, who can check that process standards are being followed and make suggestions for process improvement.
Diverse	The process should include redundant and diverse verification and validation activities.
Documentable	The process should have a defined process model that sets out the activities in the process and the documentation that is to be produced during these activities.
Robust	The process should be able to recover from failures of individual process activities.
Standardized	A comprehensive set of software development standards covering software production and documentation should be available.

Dependable Process Activities

- Requirements reviews to check that the requirements are, as far as possible, complete and consistent.
- Requirements management to ensure that changes to the requirements are controlled and that the impact of proposed requirements changes is understood.
- Formal specification, where a mathematical model of the software is created and analyzed.
- System modeling, where the software design is explicitly documented as a set of graphical models, and the links between the requirements and these models are documented.

Dependable Process Activities

- Design and program inspections, where the different descriptions of the system are inspected and checked by different people.
- Static analysis, where automated checks are carried out on the source code of the program.
- Test planning and management, where a comprehensive set of system tests is designed.
  - The testing process has to be carefully managed to demonstrate that these tests provide coverage of the system requirements and have been correctly applied in the testing process.

Dependable Processes and Agility

- Dependable software often requires certification so both process and product documentation has to be produced.
- Up-front requirements analysis is also essential to discover requirements and requirements conflicts that may compromise the safety and security of the system.
- These **conflict** with the general approach in agile development of co-development of the requirements and the system and minimizing documentation.

Dependable Processes and Agility

- An agile process may be defined that incorporates techniques such as **iterative development**, **test-first development** and **user involvement in the development team**.
- So long as the team follows that process and documents their actions, agile methods can be used.
- However, additional documentation and planning is essential so ‘pure agile’ is impractical for dependable systems engineering.

Formal Methods and Dependability

Formal Specification

- **Formal methods** are approaches to software development that are based on **mathematical representation and analysis of software**.
- Formal methods include
  - Formal specification;
  - Specification analysis and proof;
  - Transformational development;
  - Program verification.
- Formal methods significantly **reduce some types of programming errors** and can be cost-effective for dependable systems engineering.

Formal Approaches

- **Verification-based approaches**
  - Different representations of a software system such as a specification and a program implementing that specification are proved to be equivalent.
  - This demonstrates the absence of implementation errors.
- **Refinement-based approaches**
  - A representation of a system is systematically transformed into another, lower-level representation e.g. a specification is transformed automatically into an implementation.
  - This means that, if the transformation is correct, the representations are equivalent.

Use of Formal Methods

- The principal benefits of formal methods are in reducing the number of faults in systems.
- Consequently, their main area of applicability is in dependable systems engineering. There have been several successful projects where formal methods have been used in this area.
- In this area, the use of formal methods is most likely to be cost-effective because high system failure costs must be avoided.

Classes of Error

- Specification and design errors and omissions.
  - Developing and analysing a formal model of the software may reveal errors and omissions in the software requirements. If the model is generated automatically or systematically from source code, analysis using model checking can find undesirable states that may occur such as deadlock in a concurrent system.
- Inconsistences between a specification and a program.
  - If a refinement method is used, mistakes made by developers that make the software inconsistent with the specification are avoided. Program proving discovers inconsistencies between a program and its specification.

Benefits of Formal Specification

- Developing a formal specification requires the system requirements to be analyzed in detail. This helps to detect problems, inconsistencies and incompleteness in the requirements.
- As the specification is expressed in a formal language, it can be automatically analyzed to discover inconsistencies and incompleteness.
- If you use a formal method such as the B method, you can transform the formal specification into a ‘correct’ program.
- Program testing costs may be reduced if the program is formally verified against its specification.

Acceptance of Formal Methods

- Formal methods have had **limited impact on practical software development**:
  - Problem owners **cannot understand a formal specification** and so cannot assess if it is an accurate representation of their requirements.
  - It is easy to assess the costs of developing a formal specification but harder to assess the benefits. **Managers may therefore be unwilling to invest in formal methods.**
  - Software engineers are unfamiliar with this approach and are therefore reluctant to propose the use of formal methods.
  - Formal methods are still **hard to scale up to large systems**.
  - Formal specification is not **really compatible with agile development methods**.

Security Engineering

Topics Covered

- Security and dependability
- Security and organizations
- Security requirements
- Secure systems design
- Security testing and assurance

Security Engineering

- Tools, techniques and methods to support the development and maintenance of systems that can **resist malicious attacks that are intended to damage a computer-based system or its data**.
- A sub-field of the broader field of **computer security**.

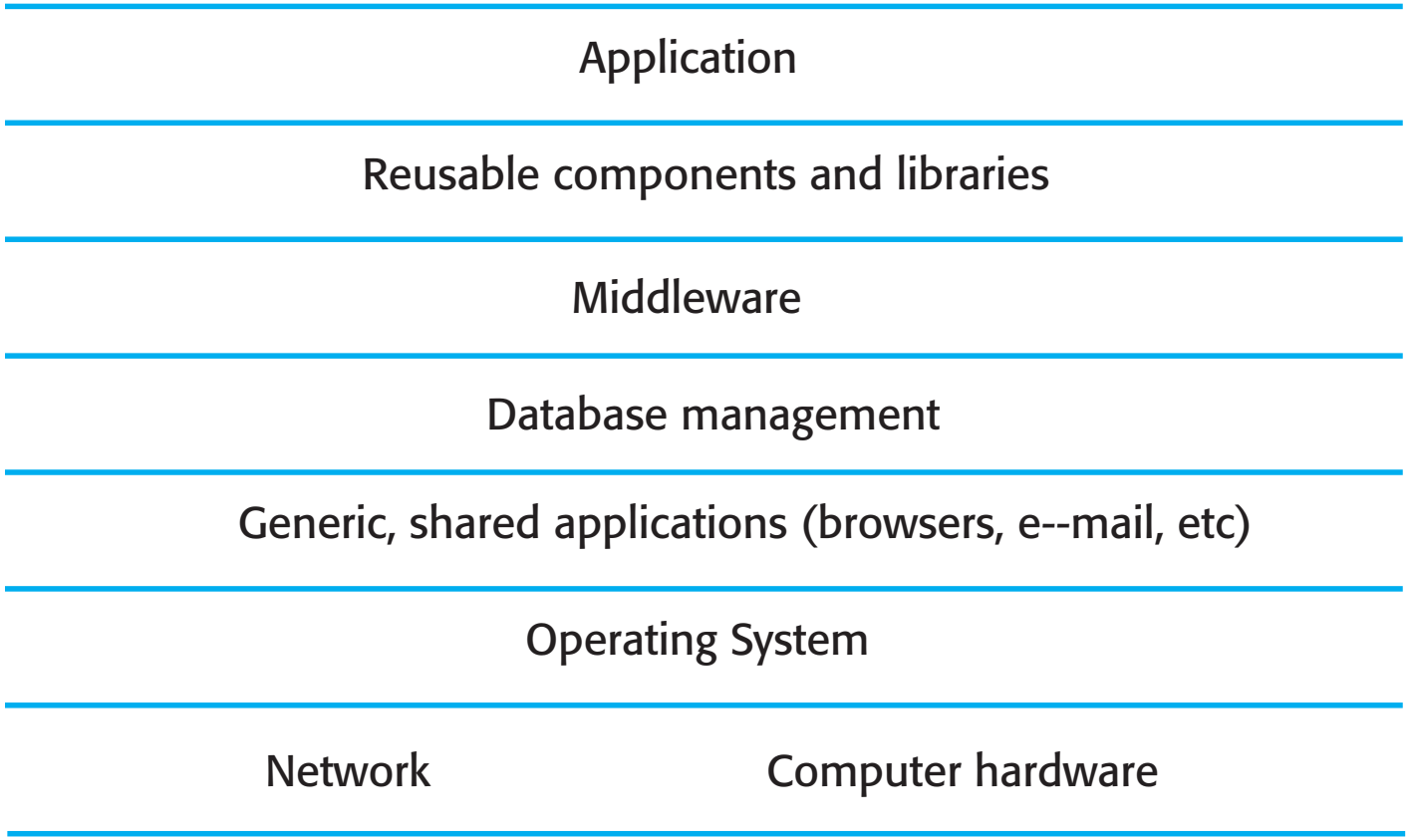
Security Dimensions

- ***Confidentiality***
  - Information in a system may be disclosed or made accessible to people or programs that are not authorized to have access to that information.
- ***Integrity***
  - Information in a system may be damaged or corrupted making it unusual or unreliable.
- ***Availability***
  - Access to a system or its data that is normally available may not be possible.

Security Levels

- **Infrastructure** security, which is concerned with maintaining the security of all systems and networks that provide an infrastructure and a set of shared services to the organization.
- **Application** security, which is concerned with the security of individual application systems or related groups of systems.
- **Operational** security, which is concerned with the secure operation and use of the organization’s systems.

System Layers where Security may be Compromised



Application/Infrastructure Security

- Application security is a software engineering problem where the system is **designed** to resist attacks.
- Infrastructure security is a systems management problem where the infrastructure is **configured** to resist attacks.
- The focus of this chapter is application security rather than infrastructure security.

System Security Management

- **User and permission management**
  - Adding and removing users from the system and setting up appropriate permissions for users
- **Software deployment and maintenance**
  - Installing application software and middleware and configuring these systems so that vulnerabilities are avoided.
- **Attack monitoring, detection and recovery**
  - Monitoring the system for unauthorized access, design strategies for resisting attacks and develop backup and recovery strategies.

Operational Security

- Primarily a human and social issue
- Concerned with ensuring the people do not take actions that may compromise system security
  - E.g. Tell others passwords, leave computers logged on
- Users sometimes take insecure actions to make it easier for them to do their jobs
- There is therefore a trade-off between system security and system effectiveness.

Security and Dependability

Security

- The security of a system is a system property that reflects the system's ability to protect itself from accidental or deliberate external attack.
- Security is essential as most systems are networked so that external access to the system through the Internet is possible.
- Security is an essential pre-requisite for **availability**, **reliability** and **safety**.

Fundamental Security

- If a system is a networked system and is insecure then statements about its reliability and its safety are unreliable.
- These statements depend on the executing system and the developed system being the same. However, intrusion can change the executing system and/or its data.
- Therefore, the reliability and safety assurance is no longer valid.

Security Terminology

Term	Definition
Asset	Something of value which has to be protected. The asset may be the software system itself or data used by that system.
Attack	An exploitation of a system's vulnerability. Generally, this is from outside the system and is a deliberate attempt to cause some damage.
Control	A protective measure that reduces a system's vulnerability. Encryption is an example of a control that reduces a vulnerability of a weak access control system
Exposure	Possible loss or harm to a computing system. This can be loss or damage to data, or can be a loss of time and effort if recovery is necessary after a security breach.
Threat	Circumstances that have potential to cause loss or harm. You can think of these as a system vulnerability that is subjected to an attack.
Vulnerability	A weakness in a computer-based system that may be exploited to cause loss or harm.

Examples of Security Terminology (Mentcare)

Term	Example
Asset	The records of each patient that is receiving or has received treatment.
Exposure	Potential financial loss from future patients who do not seek treatment because they do not trust the clinic to maintain their data. Financial loss from legal action by the sports star. Loss of reputation.
Vulnerability	A weak password system which makes it easy for users to set guessable passwords. User ids that are the same as names.
Attack	An impersonation of an authorized user.
Threat	An unauthorized user will gain access to the system by guessing the credentials (login name and password) of an authorized user.
Control	A password checking system that disallows user passwords that are proper names or words that are normally included in a dictionary.

Threat Types

- **Interception threats** that allow an attacker to gain access to an asset.
  - A possible threat to the Mentcare system might be a situation where an attacker gains access to the records of an individual patient.
- **Interruption threats** that allow an attacker to make part of the system unavailable.
  - A possible threat might be a denial of service attack on a system database server so that database connections become impossible.
- **Modification threats** that allow an attacker to tamper with a system asset.
  - In the Mentcare system, a modification threat would be where an attacker alters or destroys a patient record.
- **Fabrication threats** that allow an attacker to insert false information into a system.
  - This is perhaps not a credible threat in the Mentcare system but would be a threat in a banking system, where false transactions might be added to the system that transfer money to the perpetrator’s bank account.

Security Assurance

- **Vulnerability avoidance**
  - The system is designed so that vulnerabilities do not occur.
  - For example, if there is no external network connection then external attack is impossible
- **Attack detection and elimination**
  - The system is designed so that attacks on vulnerabilities are detected and neutralised before they result in an exposure.
  - For example, virus checkers find and remove viruses before they infect a system
- **Exposure limitation and recovery**
  - The system is designed so that the adverse consequences of a successful attack are minimised.
  - For example, a backup policy allows damaged information to be restored

Security and Dependability

- *Security and reliability*
  - If a system is attacked and the system or its data are corrupted as a consequence of that attack, then this may induce system failures that compromise the reliability of the system.
- *Security and availability*
  - A common attack on a web-based system is a denial of service attack, where a web server is flooded with service requests from a range of different sources. The aim of this attack is to make the system unavailable.

Security and Dependability

- *Security and safety*
  - An attack that corrupts the system or its data means that assumptions about safety may not hold. Safety checks rely on analysing the source code of safety critical software and assume the executing code is a completely accurate translation of that source code. If this is not the case, safety-related failures may be induced and the safety case made for the software is invalid.
- *Security and resilience*
  - Resilience is a system characteristic that reflects its ability to resist and recover from damaging events. The most probable damaging event on networked software systems is a cyberattack of some kind so most of the work now done in resilience is aimed at deterring, detecting and recovering from such attacks.

Security and Organizations

Security is a Business Issue

- Security is expensive and it is important that security decisions are made in a cost-effective way
  - There is no point in spending more than the value of an asset to keep that asset secure.
- Organizations use a risk-based approach to support security decision making and should have a defined security policy based on security risk analysis
- Security risk analysis is a business rather than a technical process

Organizational Security Policies

- Security policies should set out general information access strategies that should apply across the organization.
- The point of security policies is to inform everyone in an organization about security so these should not be long and detailed technical documents.
- From a security engineering perspective, the security policy defines, in broad terms, the security goals of the organization.
- The security engineering process is concerned with implementing these goals.

Security Policies

- *The assets that must be protected*
  - It is not cost-effective to apply stringent security procedures to all organizational assets. Many assets are not confidential and can be made freely available.
- *The level of protection that is required for different types of asset*
  - For sensitive personal information, a high level of security is required; for other information, the consequences of loss may be minor so a lower level of security is adequate.

Security Policies

- *The responsibilities of individual users, managers and the organization*
  - The security policy should set out what is expected of users e.g. strong passwords, log out of computers, office security, etc.
- *Existing security procedures and technologies that should be maintained*
  - For reasons of practicality and cost, it may be essential to continue to use existing approaches to security even where these have known limitations.

Security Risk Assessment and Management

- Risk assessment and management is concerned with assessing the possible losses that might ensue from attacks on the system and balancing these losses against the costs of security procedures that may reduce these losses.
- Risk management should be driven by an **organizational security policy**.
- Risk management involves
  - Preliminary risk assessment
  - Life cycle risk assessment
  - Operational risk assessment

Preliminary Risk Assessment

- The aim of this initial risk assessment is to identify generic risks that are applicable to the system and to decide if an adequate level of security can be achieved at a reasonable cost.
- The risk assessment should focus on the identification and analysis of high-level risks to the system.
- The outcomes of the risk assessment process are used to help identify security requirements.

Design Risk Assessment

- This risk assessment takes place during the system development life cycle and is informed by the technical system design and implementation decisions.
- The results of the assessment may lead to changes to the security requirements and the addition of new requirements.
- Known and potential vulnerabilities are identified, and this knowledge is used to inform decision making about the system functionality and how it is to be implemented, tested, and deployed.

Operational Risk Assessment

- This risk assessment process focuses on the use of the system and the possible risks that can arise from human behavior.
- Operational risk assessment should continue after a system has been installed to take account of how the system is used.
- Organizational changes may mean that the system is used in different ways from those originally planned. These changes lead to new security requirements that have to be implemented as the system evolves.

Security Requirements

Security Specification

- Security specification has something in common with safety requirements specification – in both cases, your concern is to avoid something bad happening.
- Four major differences
  - Safety problems are accidental – the software is not operating in a hostile environment. In security, you must assume that attackers have knowledge of system weaknesses
  - When safety failures occur, you can look for the root cause or weakness that led to the failure. When failure results from a deliberate attack, the attacker may conceal the cause of the failure.
  - Shutting down a system can avoid a safety-related failure. Causing a shut down may be the aim of an attack.
  - Safety-related events are not generated from an intelligent adversary. An attacker can probe defenses over time to discover weaknesses.

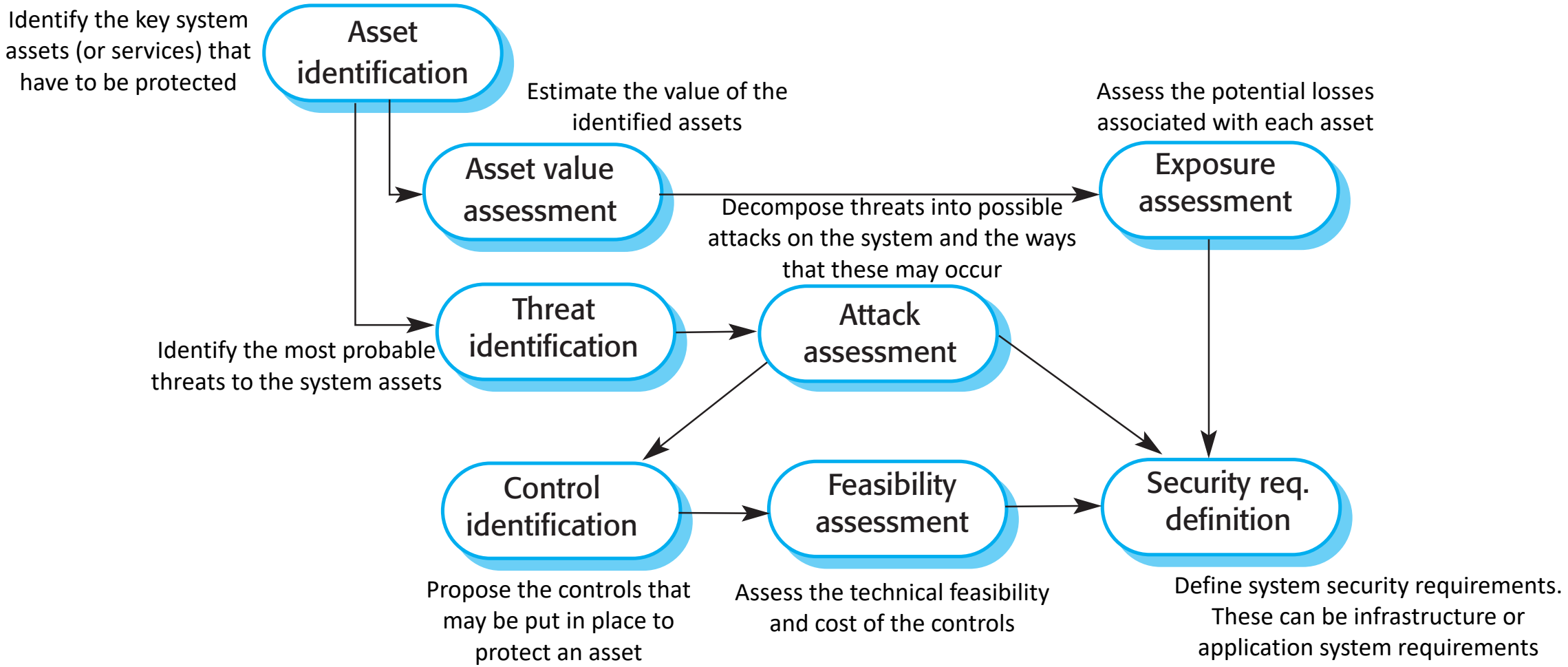
Types of Security Requirement

- Identification requirements.
- Authentication requirements.
- Authorisation requirements.
- Immunity requirements.
- Integrity requirements.
- Intrusion detection requirements.
- Non-repudiation requirements.
- Privacy requirements.
- Security auditing requirements.
- System maintenance security requirements.

Security Requirement Classification

- Risk avoidance requirements set out the risks that should be avoided by designing the system so that these risks simply cannot arise.
- Risk detection requirements define mechanisms that identify the risk if it arises and neutralise the risk before losses occur.
- Risk mitigation requirements set out how the system should be designed so that it can recover from and restore system assets after some loss has occurred.

Preliminary Risk Assessment Process for Security Requirements



Assessment Report for the Mentcare System

Asset	Value	Exposure
The information system	High. Required to support all clinical consultations. Potentially safety-critical.	High. Financial loss as clinics may have to be canceled. Costs of restoring system. Possible patient harm if treatment cannot be prescribed.
The patient database	High. Required to support all clinical consultations. Potentially safety-critical.	High. Financial loss as clinics may have to be canceled. Costs of restoring system. Possible patient harm if treatment cannot be prescribed.
An individual patient record	Normally low although may be high for specific high-profile patients.	Low direct losses but possible loss of reputation.

Threat and Control Analysis in a Preliminary Risk Assessment Report

Threat	Probability	Control	Feasibility
An unauthorized user gains access as system manager and makes system unavailable	Low	Only allow system management from specific locations that are physically secure.	Low cost of implementation but care must be taken with key distribution and to ensure that keys are available in the event of an emergency.
An unauthorized user gains access as system user and accesses confidential information	High	Require all users to authenticate themselves using a biometric mechanism. Log all changes to patient information to track system usage.	Technically feasible but high-cost solution. Possible user resistance. Simple and transparent to implement and also supports recovery.

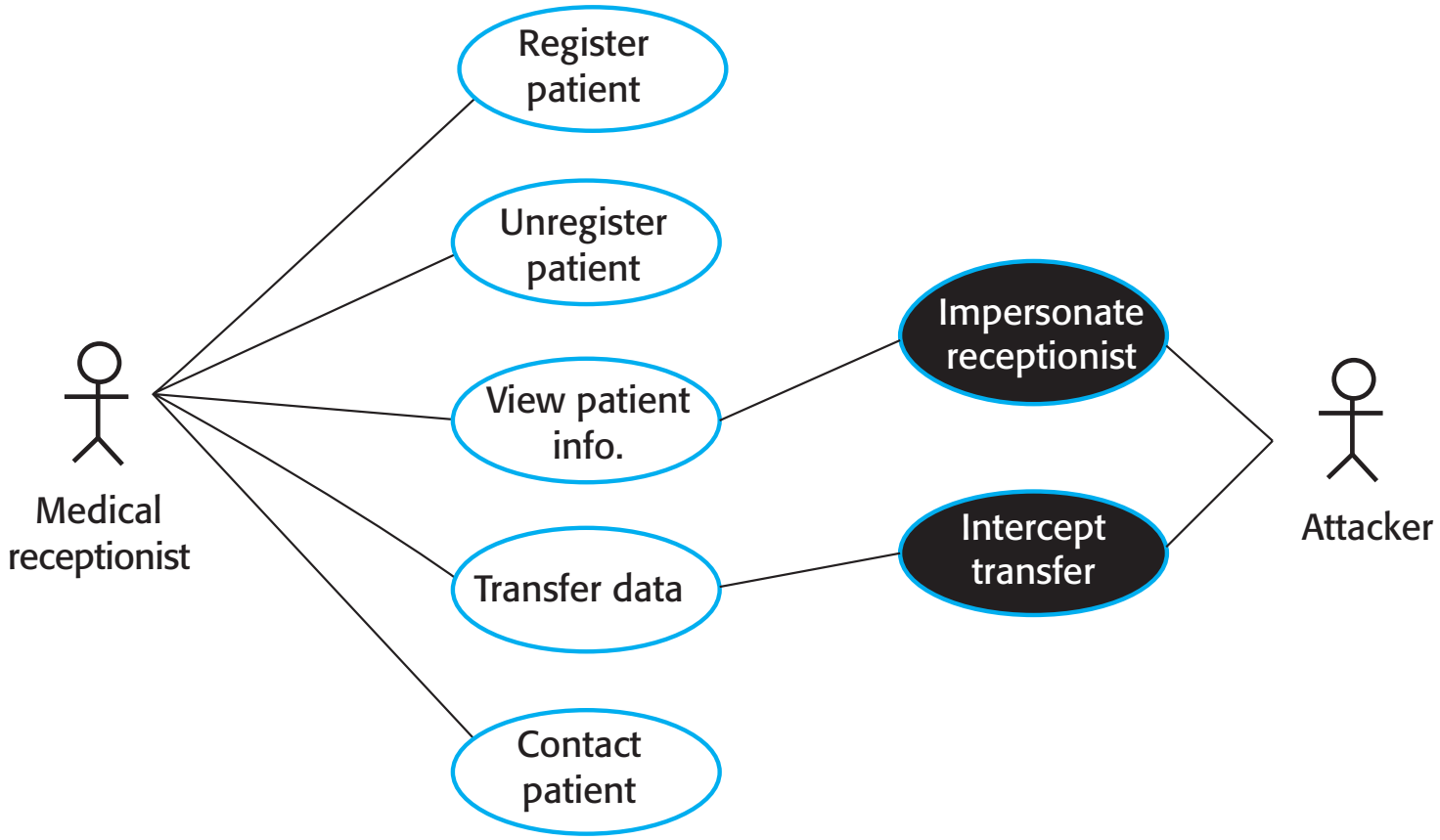
Security Requirements for the Mentcare System

- Patient information shall be downloaded at the start of a clinic session to a secure area on the system client that is used by clinical staff.
- All patient information on the system client shall be encrypted.
- Patient information shall be uploaded to the database after a clinic session has finished and deleted from the client computer.
- A log on a separate computer from the database server must be maintained of all changes made to the system database.

Misuse Cases

- Misuse cases are instances of threats to a system
- Interception threats
  - Attacker gains access to an asset
- Interruption threats
  - Attacker makes part of a system unavailable
- Modification threats
  - A system asset if tampered with
- Fabrication threats
  - False information is added to a system

Misuse Cases



Secure Systems Design

Secure Systems Design

- Security should be designed into a system – it is very difficult to make an insecure system secure after it has been designed or implemented
- Architectural design
  - how do architectural design decisions affect the security of a system?
- Good practice
  - what is accepted good practice when designing secure systems?

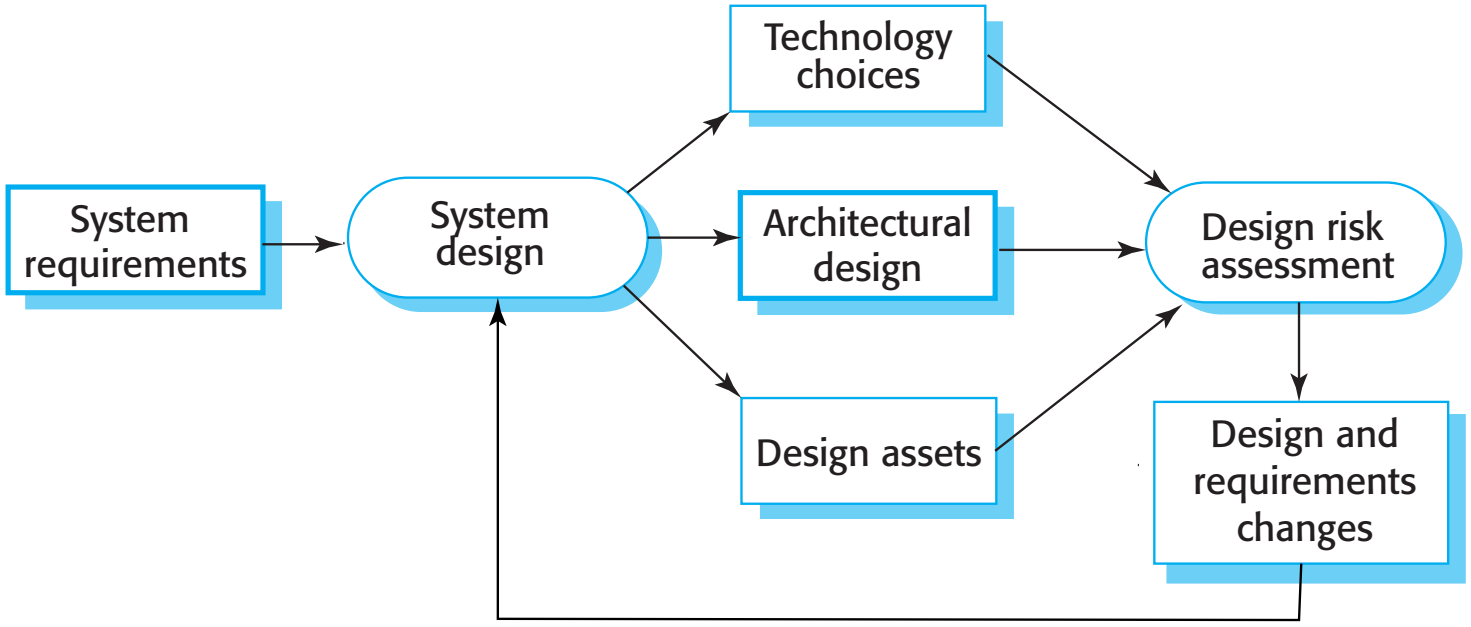
Design Compromises

- Adding security features to a system to enhance its security affects other attributes of the system
- Performance
  - Additional security checks slow down a system so its response time or throughput may be affected
- Usability
  - Security measures may require users to remember information or require additional interactions to complete a transaction. This makes the system less usable and can frustrate system users.

Design Risk Assessment

- Risk assessment while the system is being developed and after it has been deployed
- More information is available - system platform, middleware and the system architecture and data organisation.
- Vulnerabilities that arise from design choices may therefore be identified.

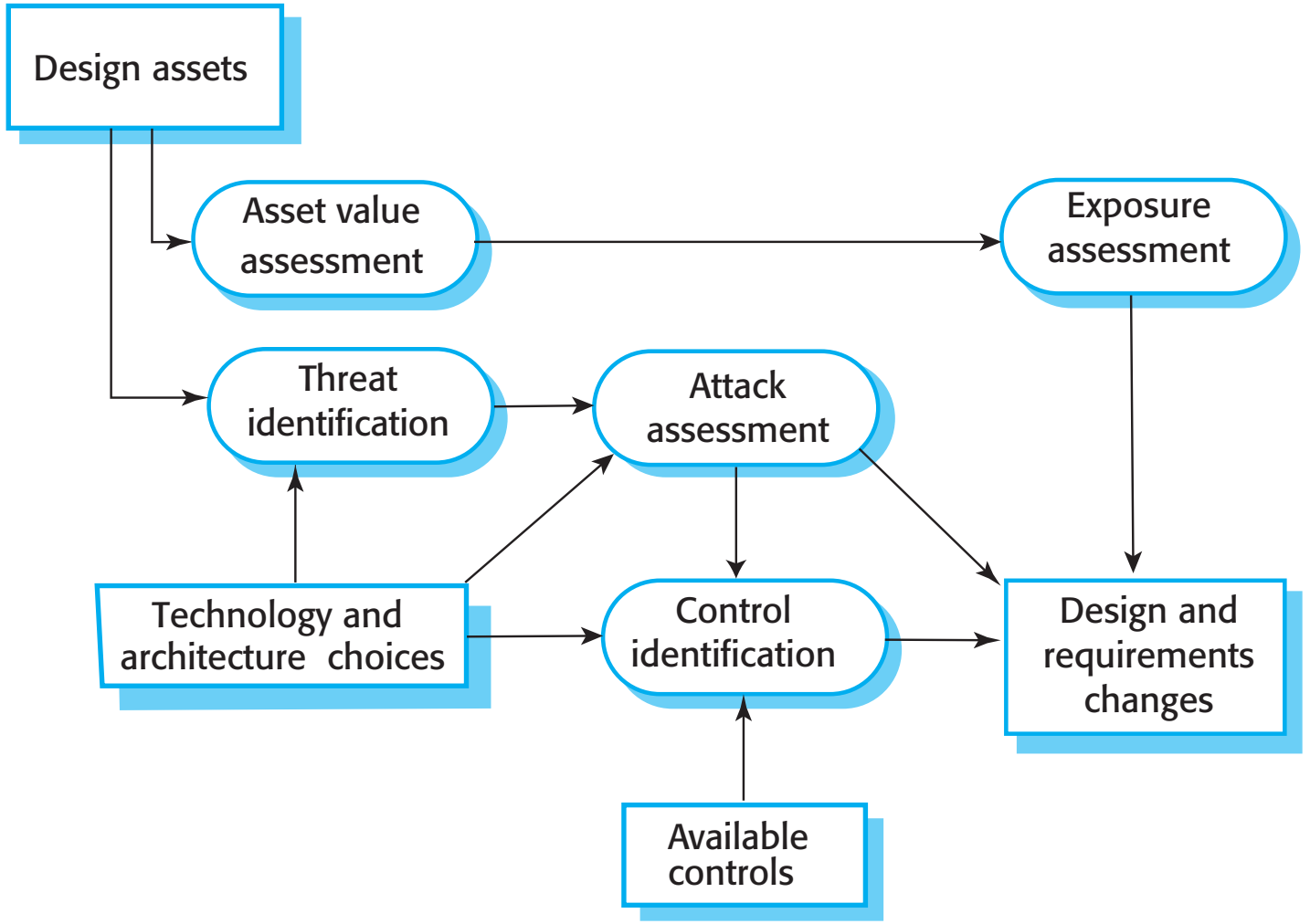
Design and Risk Assessment



Protection Requirements

- Protection requirements may be generated when knowledge of information representation and system distribution
- Separating patient and treatment information limits the amount of information (personal patient data) that needs to be protected
- Maintaining copies of records on a local client protects against denial of service attacks on the server
  - But these may need to be encrypted

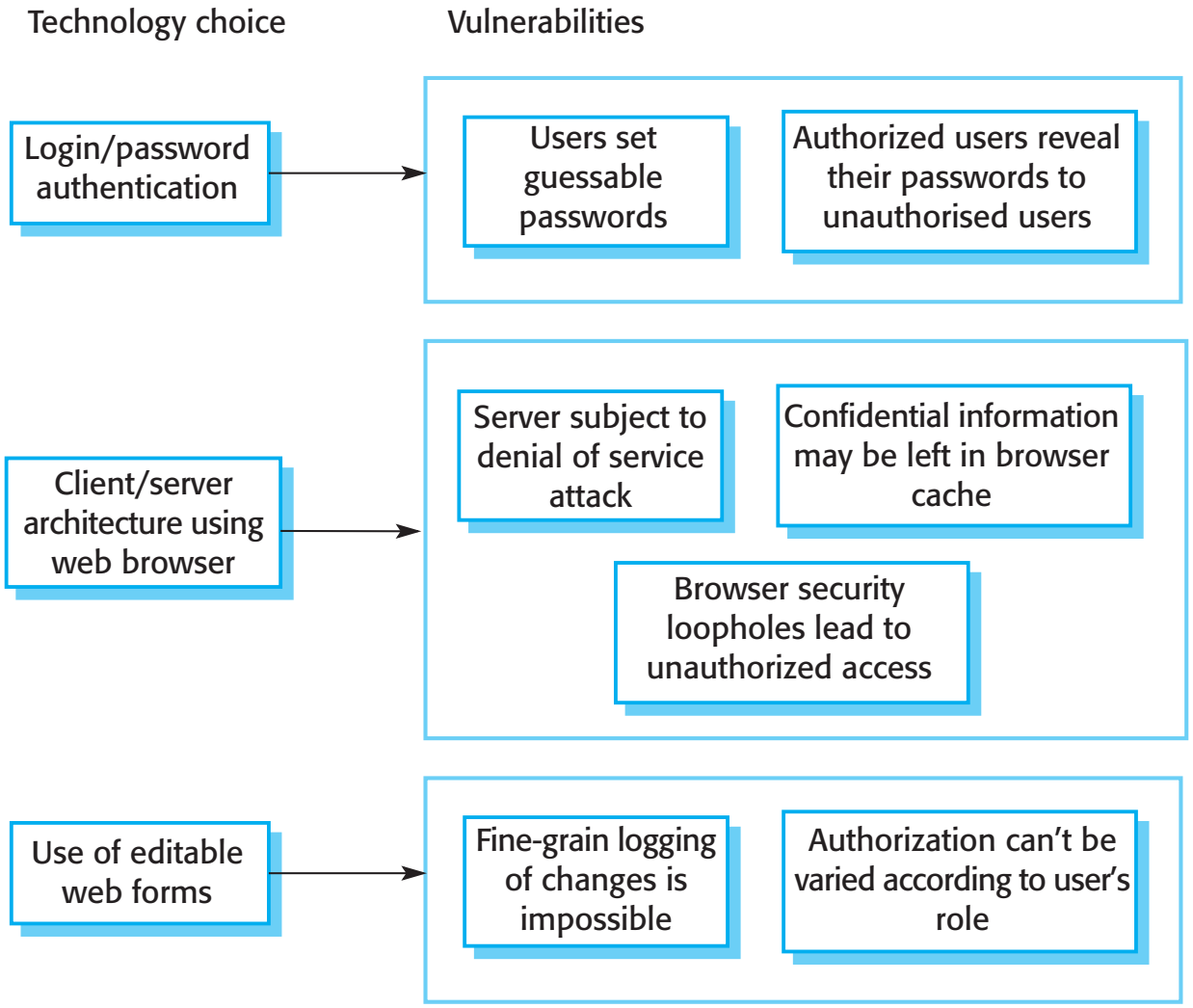
Design Risk Assessment



Design Decisions from use of COTS

- System users authenticated using a name/password combination.
- The system architecture is client-server with clients accessing the system through a standard web browser.
- Information is presented as an editable web form.

Vulnerabilities Associated with Technology Choices



Security Requirements

- A password checker shall be made available and shall be run daily. Weak passwords shall be reported to system administrators.
- Access to the system shall only be allowed by approved client computers.
- All client computers shall have a single, approved web browser installed by system administrators.

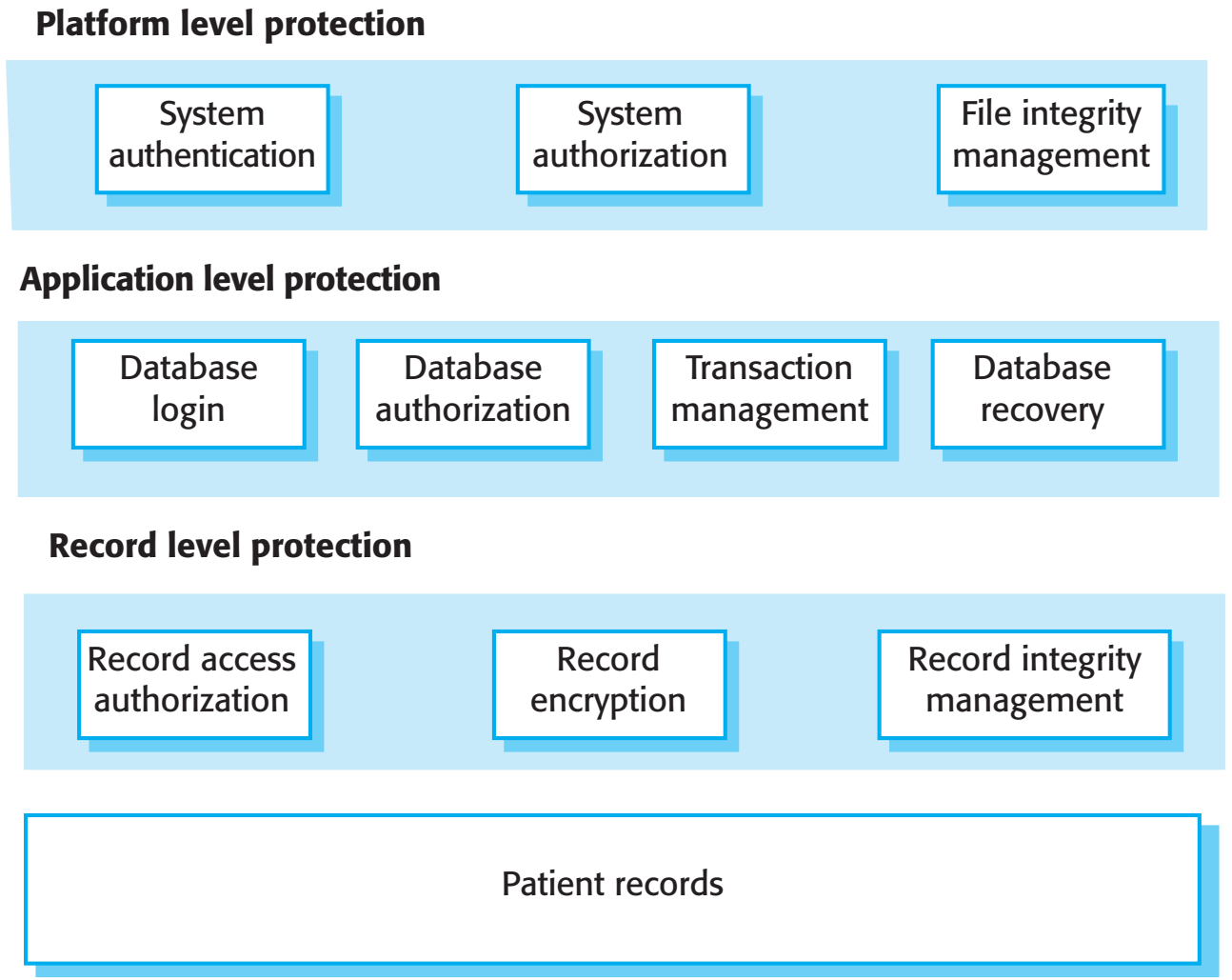
Architectural Design

- Two fundamental issues have to be considered when designing an architecture for security.
  - Protection
    - How should the system be organized so that critical assets can be protected against external attack?
  - Distribution
    - How should system assets be distributed so that the effects of a successful attack are minimized?
- These are potentially conflicting
  - If assets are distributed, then they are more expensive to protect. If assets are protected, then usability and performance requirements may be compromised.

Protection

- Platform-level protection
  - Top-level controls on the platform on which a system runs.
- Application-level protection
  - Specific protection mechanisms built into the application itself e.g. additional password protection.
- Record-level protection
  - Protection that is invoked when access to specific information is requested
- These lead to a layered protection architecture

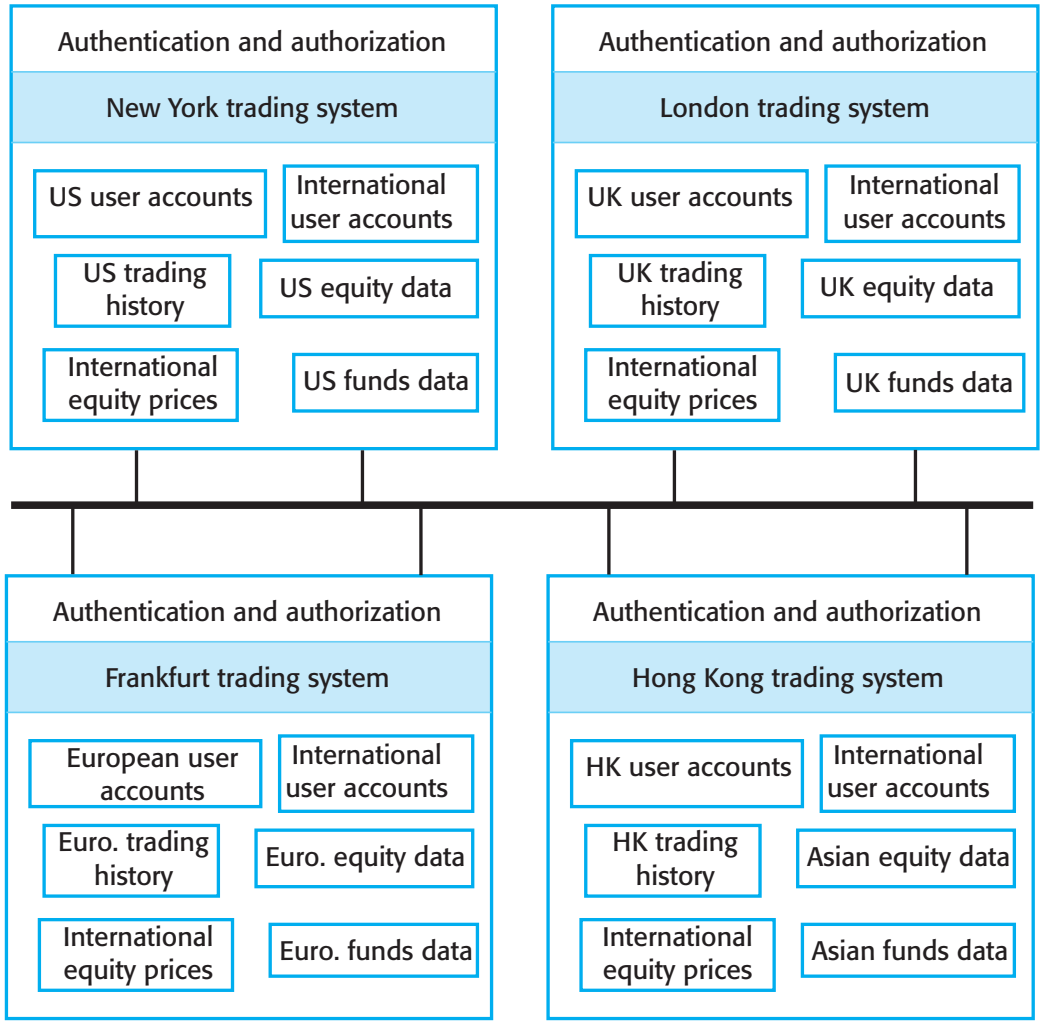
Layered Protection Architecture



Distribution

- Distributing assets means that attacks on one system do not necessarily lead to complete loss of system service
- Each platform has separate protection features and may be different from other platforms so that they do not share a common vulnerability
- Distribution is particularly important if the risk of denial of service attacks is high

Distributed Assets in an Equity Trading System



Design Guidelines for Security Engineering

- Design guidelines encapsulate good practice in secure systems design
- Design guidelines serve two purposes:
  - They raise awareness of security issues in a software engineering team. Security is considered when design decisions are made.
  - They can be used as the basis of a review checklist that is applied during the system validation process.
- Design guidelines here are applicable during software specification and design

Design Guidelines 1-3

- Base decisions on an explicit security policy
  - Define a security policy for the organization that sets out the fundamental security requirements that should apply to all organizational systems.
- Avoid a single point of failure
  - Ensure that a security failure can only result when there is more than one failure in security procedures. For example, have password and question-based authentication.
- Fail securely
  - When systems fail, for whatever reason, ensure that sensitive information cannot be accessed by unauthorized users even although normal security procedures are unavailable.

Design Guidelines 4-6

- Balance security and usability
  - Try to avoid security procedures that make the system difficult to use. Sometimes you have to accept weaker security to make the system more usable.
- Log user actions
  - Maintain a log of user actions that can be analyzed to discover who did what. If users know about such a log, they are less likely to behave in an irresponsible way.
- Use redundancy and diversity to reduce risk
  - Keep multiple copies of data and use diverse infrastructure so that an infrastructure vulnerability cannot be the single point of failure.

Design Guidelines 7-10

- Specify the format of all system inputs
  - If input formats are known then you can check that all inputs are within range so that unexpected inputs don’t cause problems.
- Compartmentalize your assets
  - Organize the system so that assets are in separate areas and users only have access to the information that they need rather than all system information.
- Design for deployment
  - Design the system to avoid deployment problems
- Design for recoverability
  - Design the system to simplify recoverability after a successful attack.

Aspects of Secure Systems Programming

- Vulnerabilities are often language-specific.
  - Array bound checking is automatic in languages like Java so this is not a vulnerability that can be exploited in Java programs.
  - However, millions of programs are written in C and C++ as these allow for the development of more efficient software so simply avoiding the use of these languages is not a realistic option.
- Security vulnerabilities are closely related to program reliability.
  - Programs without array bound checking can crash so actions taken to improve program reliability can also improve system security.

Dependable Programming Guidelines

Dependable programming guidelines

1. **Limit the visibility of information in a program**
2. **Check all inputs for validity**
3. **Provide a handler for all exceptions**
4. **Minimize the use of error-prone constructs**
5. **Provide restart capabilities**
6. **Check array bounds**
7. **Include timeouts when calling external components**
8. **Name all constants that represent real-world values**

Security Testing and Assurance

Security Testing

- Testing the extent to which the system can protect itself from external attacks.
- Problems with security testing:
  - Security requirements are ‘shall not’ requirements i.e. they specify what should not happen. It is not usually possible to define security requirements as simple constraints that can be checked by the system.
  - The people attacking a system are intelligent and look for vulnerabilities. They can experiment to discover weaknesses and loopholes in the system.

Security Validation

- Experience-based testing
  - The system is reviewed and analysed against the types of attack that are known to the validation team.
- Penetration testing
  - A team is established whose goal is to breach the security of the system by simulating attacks on the system.
- Tool-based analysis
  - Various security tools such as password checkers are used to analyse the system in operation.
- Formal verification
  - The system is verified against a formal security specification.

Examples of Entries in a Security Checklist

Security checklist

1. Do all files that are created in the application have appropriate access permissions? The wrong access permissions may lead to these files being accessed by unauthorized users.

2. Does the system automatically terminate user sessions after a period of inactivity? Sessions that are left active may allow unauthorized access through an unattended computer.

3. If the system is written in a programming language without array bound checking, are there situations where buffer overflow may be exploited? Buffer overflow may allow attackers to send code strings to the system and then execute them.

4. If passwords are set, does the system check that passwords are ‘strong’? Strong passwords consist of mixed letters, numbers, and punctuation, and are not normal dictionary entries. They are more difficult to break than simple passwords.

5. Are inputs from the system’s environment always checked against an input specification? Incorrect processing of badly formed inputs is a common cause of security vulnerabilities.

つづく