ECE 651 – Software Engineering
Week Ten: Reliability and Security
March 23rd, 2016

Ric Telford
Adjunct Associate Professor

Founder, Telford Ventures

Duke | PRATT SCHOOL OF ENGINEERING

03/23/16 – Week 10 Overview

- Recap / Announcements
- Guest Speaker
- Lecture - Reliability
- Lecture - Security

Announcements

- Have you met with your TA for meeting #1? This needs to occur in March to get credit.
- I have revised the schedule again to pull in the Final Exam:
  - March 23rd - Week 10 – Reliability, Security, Guest Speaker
  - March 30th - Week 11 - Risk Mgmt and Project Planning/Mgmt
  - April 6th - Week 12 – Web and Mobile, Guest Speaker
  - April 13th - Week 13 – SOA, Distributed and Real Time
  - April 20th - Week 14 – Team Presentations (Documentation and Code)
  - April 27th Week 15 – Team Presentations continued, followed by Final Exam
- Please **keep laptops closed** when we have a guest and **feel free to ask questions** of the speaker throughout his presentation

Recap

- Midterm Review

Guest Speaker – Don Boulia

- Vice President, Cloud Strategy and Portfolio Management, IBM

Chapter 11 – Reliability Engineering

- 11.0, 11.1 - Availability and reliability
- 11.2 - Reliability requirements
- Fault-tolerant architectures
- 11.4 - Programming for reliability
- Reliability measurement

Software reliability

- In general, software customers expect all software to be dependable. However, for non-critical applications, they may be willing to accept some system failures.
- Some applications (critical systems) have very high reliability requirements and special software engineering techniques may be used to achieve this.
  - Medical systems
  - Telecommunications and power systems
  - Aerospace systems

Faults, errors and failures

Term	Description
Human error or mistake	Human behavior that results in the introduction of faults into a system. For example, in the wilderness weather system, a programmer might decide that the way to compute the time for the next transmission is to add 1 hour to the current time. This works except when the transmission time is between 23.00 and midnight (midnight is 00.00 in the 24-hour clock).
System fault	A characteristic of a software system that can lead to a system error. The fault is the inclusion of the code to add 1 hour to the time of the last transmission, without a check if the time is greater than or equal to 23.00.
System error	An erroneous system state that can lead to system behavior that is unexpected by system users. The value of transmission time is set incorrectly (to 24.XX rather than 00.XX) when the faulty code is executed.
System failure	An event that occurs at some point in time when the system does not deliver a service as expected by its users. No weather data is transmitted because the time is invalid.

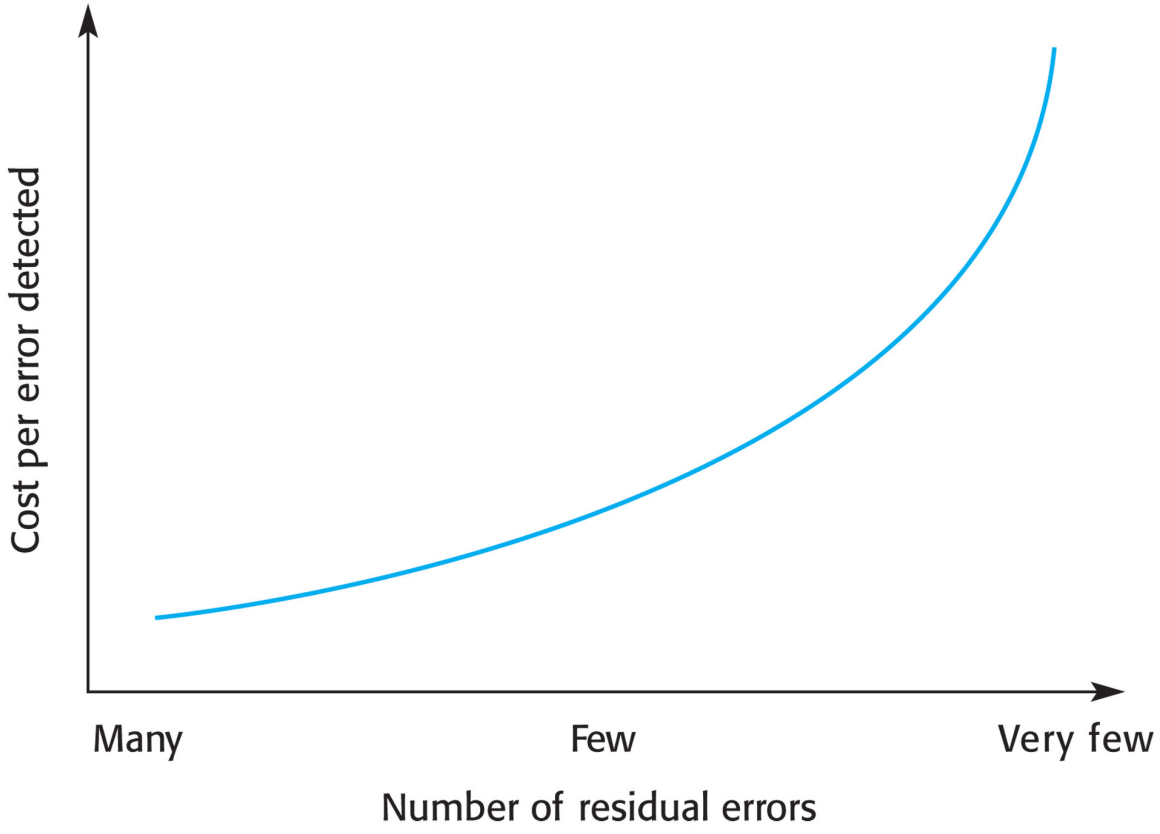
Faults and failures

- Failures are a usually a result of system errors that are derived from faults in the system
- However, faults do not necessarily result in system errors
  - The erroneous system state resulting from the fault may be transient and 'corrected' before an error arises.
  - The faulty code may never be executed.
- Errors do not necessarily lead to system failures
  - The error can be corrected by built-in error detection and recovery
  - The failure can be protected against by built-in protection facilities. These may, for example, protect system resources from system errors

Fault management

- *Fault avoidance* (Development Phase)
  - The system is developed in such a way that human error is avoided and thus system faults are minimized.
  - The development process is organized so that faults in the system are detected and repaired before delivery to the customer.
- *Fault detection* (Test Phase)
  - Verification and validation techniques are used to discover and remove faults in a system before it is deployed.
- *Fault tolerance* (Delivered Code)
  - The system is designed so that faults in the delivered software do not result in system failure.

The increasing costs of residual fault removal



11.1 Availability and reliability

Availability and reliability

- *Reliability*
  - The probability of failure-free system operation over a specified time in a given environment for a given purpose
- *Availability*
  - The probability that a system, at a point in time, will be operational and able to deliver the requested services
- Both of these attributes can be expressed quantitatively e.g. availability of 0.999 means that the system is up and running for 99.9% of the time.

How many hours a year is 0.1% downtime?

Perceptions of reliability

- The formal definition of reliability does not always reflect the user's perception of a system's reliability
  - The assumptions that are made about the environment where a system will be used may be incorrect
    - Usage of a system in an office environment is likely to be quite different from usage of the same system in a university environment
  - The consequences of system failures affects the perception of reliability
    - Unreliable windscreen wipers in a car may be irrelevant in a dry climate
    - Failures that have serious consequences (such as an engine breakdown in a car) are given greater weight by users than failures that are inconvenient

Availability perception

- Availability is usually expressed as a percentage of the time that the system is available to deliver services e.g. 99.95%.
- However, this does not take into account two factors:
  - The number of users affected by the service outage. Loss of service in the middle of the night is less important for many systems than loss of service during peak usage periods.
  - The length of the outage. The longer the outage, the more the disruption. Several short outages are less likely to be disruptive than 1 long outage. Long repair times are a particular problem.

11.2 Reliability requirements

System reliability requirements

- *Functional reliability requirements define system and software functions that avoid, detect or tolerate faults in the software and so ensure that these faults do not lead to system failure.*
- Software reliability requirements may also be included to cope with hardware failure or operator error.
  - Reliability is a measurable system attribute so non-functional reliability requirements may be specified quantitatively. These define the number of failures that are acceptable during normal use of the system or the time in which the system must be available.

Reliability metrics

- *Reliability metrics are units of measurement of system reliability.*
- System reliability is measured by counting the number of operational failures and, where appropriate, relating these to the demands made on the system and the time that the system has been operational.
- A long-term measurement program is required to assess the reliability of critical systems.
- Metrics
  - Probability of failure on demand
  - Rate of occurrence of failures/Mean time to failure
  - Availability

Probability of failure on demand (POFOD)

- This is the probability that the system will fail when a service request is made. Useful when demands for service are intermittent and relatively infrequent.
- Appropriate for protection systems where services are demanded occasionally and where there are serious consequence if the service is not delivered.
- Relevant for many safety-critical systems with exception management components
  - Emergency shutdown system in a chemical plant.

Rate of fault occurrence (ROCOF)

- Reflects the rate of occurrence of failure in the system.
- ROCOF of 0.002 means 2 failures are likely in each 1000 operational time units e.g. 2 failures per 1000 hours of operation.
- Relevant for systems where the system has to process a large number of similar requests in a short time
  - Credit card processing system, airline booking system.
- Reciprocal of ROCOF is *Mean time to Failure (MTTF)*
  - Relevant for systems with long transactions i.e. where system processing takes a long time (e.g. CAD systems). MTTF should be longer than expected transaction length.

Availability

- *Measure of the fraction of the time that the system is available for use.*
- Takes repair and restart time into account
- Availability of 0.998 means software is available for 998 out of 1000 time units.
- Relevant for non-stop, continuously running systems
  - telephone switching systems, railway signalling systems.

Availability specification

Availability	Explanation
0.9	The system is available for 90% of the time. This means that, in a 24-hour period (1,440 minutes), the system will be unavailable for 144 minutes.
0.99	In a 24-hour period, the system is unavailable for 14.4 minutes.
0.999	The system is unavailable for 84 seconds in a 24-hour period.
0.9999	The system is unavailable for 8.4 seconds in a 24-hour period. Roughly, one minute per week.

Non-functional reliability requirements

- *Non-functional reliability requirements are specifications of the required reliability and availability of a system using one of the reliability metrics (POFOD, ROCOF or AVAIL).*
- Quantitative reliability and availability specification has been used for many years in safety-critical systems but is uncommon for business critical systems.
- However, as more and more companies demand 24/7 service from their systems, it makes sense for them to be precise about their reliability and availability expectations.

Functional reliability requirements

- *Checking* requirements that identify checks to ensure that incorrect data is detected before it leads to a failure.
- *Recovery* requirements that are geared to help the system recover after a failure has occurred.
- *Redundancy* requirements that specify redundant features of the system to be included.
- *Process* requirements for reliability which specify the development process to be used may also be included.

Examples of functional reliability requirements

RR1: A pre-defined range for all operator inputs shall be defined and the system shall check that all operator inputs fall within this pre-defined range. (Checking)

RR2: Copies of the patient database shall be maintained on two separate servers that are not housed in the same building. (Recovery, redundancy)

RR3: N-version programming shall be used to implement the braking control system. (Redundancy)

RR4: The system must be implemented in a safe subset of Ada and checked using static analysis. (Process)

11.4 Programming for reliability

Dependable programming

- Good programming practices can be adopted that help reduce the incidence of program faults.
- These programming practices support
  - Fault avoidance
  - Fault detection
  - Fault tolerance

Good practice guidelines for dependable programming

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*

(1) Limit the visibility of information in a program

- Program components should only be allowed access to data that they need for their implementation.
- This means that accidental corruption of parts of the program state by these components is impossible.
- You can control visibility by using abstract data types where the data representation is private and you only allow access to the data through predefined operations such as get () and put ().

(2) Check all inputs for validity

- All program take inputs from their environment and make assumptions about these inputs.
- However, program specifications rarely define what to do if an input is not consistent with these assumptions.
- Consequently, many programs behave unpredictably when presented with unusual inputs and, sometimes, these are threats to the security of the system.
- Consequently, you should always check inputs before processing against the assumptions made about these inputs.

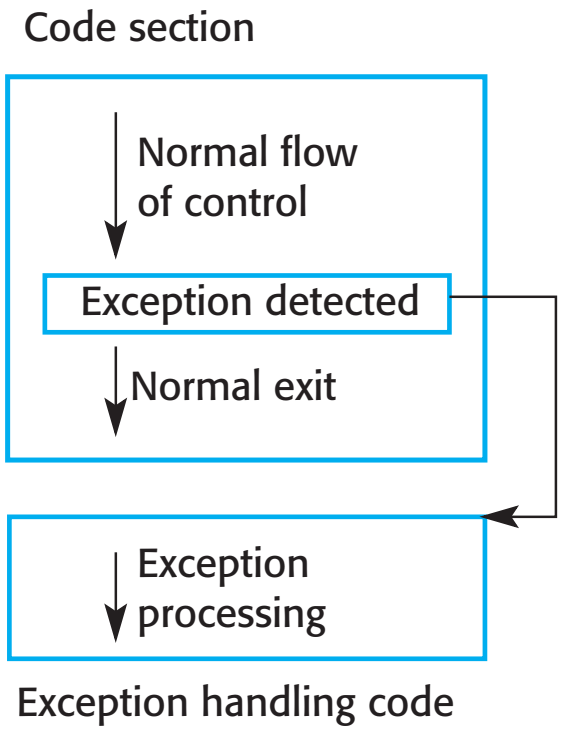
Validity checks

- Range checks
  - Check that the input falls within a known range.
- Size checks
  - Check that the input does not exceed some maximum size e.g. 40 characters for a name.
- Representation checks
  - Check that the input does not include characters that should not be part of its representation e.g. names do not include numerals.
- Reasonableness checks
  - Use information about the input to check if it is reasonable rather than an extreme value.

(3) Provide a handler for all exceptions

- A program exception is an error or some unexpected event such as a power failure.
- Exception handling constructs allow for such events to be handled without the need for continual status checking to detect exceptions.
- Using normal control constructs to detect exceptions needs many additional statements to be added to the program. This adds a significant overhead and is potentially error-prone.

Exception handling



Exception handling

- Three possible exception handling strategies
  - Signal to a calling component that an exception has occurred and provide information about the type of exception.
  - Carry out some alternative processing to the processing where the exception occurred. This is only possible where the exception handler has enough information to recover from the problem that has arisen.
  - Pass control to a run-time support system to handle the exception.
- Exception handling is a mechanism to provide some fault tolerance

(4) Minimize the use of error-prone constructs

- Program faults are usually a consequence of human error because programmers lose track of the relationships between the different parts of the system
- This is exacerbated by error-prone constructs in programming languages that are inherently complex or that don't check for mistakes when they could do so.
- Therefore, when programming, you should try to avoid or at least minimize the use of these error-prone constructs.

Error-prone constructs

- Unconditional branch (goto) statements
- Floating-point numbers
  - Inherently imprecise. The imprecision may lead to invalid comparisons.
- Pointers
  - Pointers referring to the wrong memory areas can corrupt data. Aliasing can make programs difficult to understand and change.
- Dynamic memory allocation
  - Run-time allocation can cause memory overflow.

Error-prone constructs

- Parallelism
  - Can result in subtle timing errors because of unforeseen interaction between parallel processes.
- Recursion
  - Errors in recursion can cause memory overflow as the program stack fills up.
- Interrupts
  - Interrupts can cause a critical operation to be terminated and make a program difficult to understand.
- Inheritance
  - Code is not localised. This can result in unexpected behaviour when changes are made and problems of understanding the code.

Error-prone constructs

- Aliasing
  - Using more than 1 name to refer to the same state variable.
- Unbounded arrays
  - Buffer overflow failures can occur if no bound checking on arrays.
- Default input processing
  - An input action that occurs irrespective of the input.
  - This can cause problems if the default action is to transfer control elsewhere in the program. In incorrect or deliberately malicious input can then trigger a program failure.

(5) Provide restart capabilities

- For systems that involve long transactions or user interactions, you should always provide a restart capability that allows the system to restart after failure without users having to redo everything that they have done.
- Restart depends on the type of system
  - Keep copies of forms so that users don't have to fill them in again if there is a problem
  - Save state periodically and restart from the saved state

(6) Check array bounds

- In some programming languages, such as C, it is possible to address a memory location outside of the range allowed for in an array declaration.
- This leads to the well-known 'bounded buffer' vulnerability where attackers write executable code into memory by deliberately writing beyond the top element in an array.
- If your language does not include bound checking, you should therefore always check that an array access is within the bounds of the array.

(7) Include timeouts when calling external components

- In a distributed system, failure of a remote computer can be ‘silent’ so that programs expecting a service from that computer may never receive that service or any indication that there has been a failure.
- To avoid this, you should always include timeouts on all calls to external components.
- After a defined time period has elapsed without a response, your system should then assume failure and take whatever actions are required to recover from this.

(8) Name all constants that represent real-world values

- Always give constants that reflect real-world values (such as tax rates) names rather than using their numeric values and always refer to them by name
- You are less likely to make mistakes and type the wrong value when you are using a name rather than a value.
- This means that when these ‘constants’ change (for sure, they are not really constant), then you only have to make the change in one place in your program.

Key points

- Software reliability can be achieved by avoiding the introduction of faults, by detecting and removing faults before system deployment and by including fault tolerance facilities that allow the system to remain operational after a fault has caused a system failure.
- Reliability requirements can be defined quantitatively in the system requirements specification.
- Reliability metrics include probability of failure on demand (POFOD), rate of occurrence of failure (ROCOF) and availability (AVAIL).

Key points

- Functional reliability requirements are requirements for system functionality, such as checking and redundancy requirements, which help the system meet its non-functional reliability requirements.
- Dependable programming relies on including redundancy in a program as checks on the validity of inputs and the values of program variables.

Chapter 13 - Security Engineering

- **13.0, 13.1 - Security and dependability**
- Security and organizations
- **13.3 - 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.

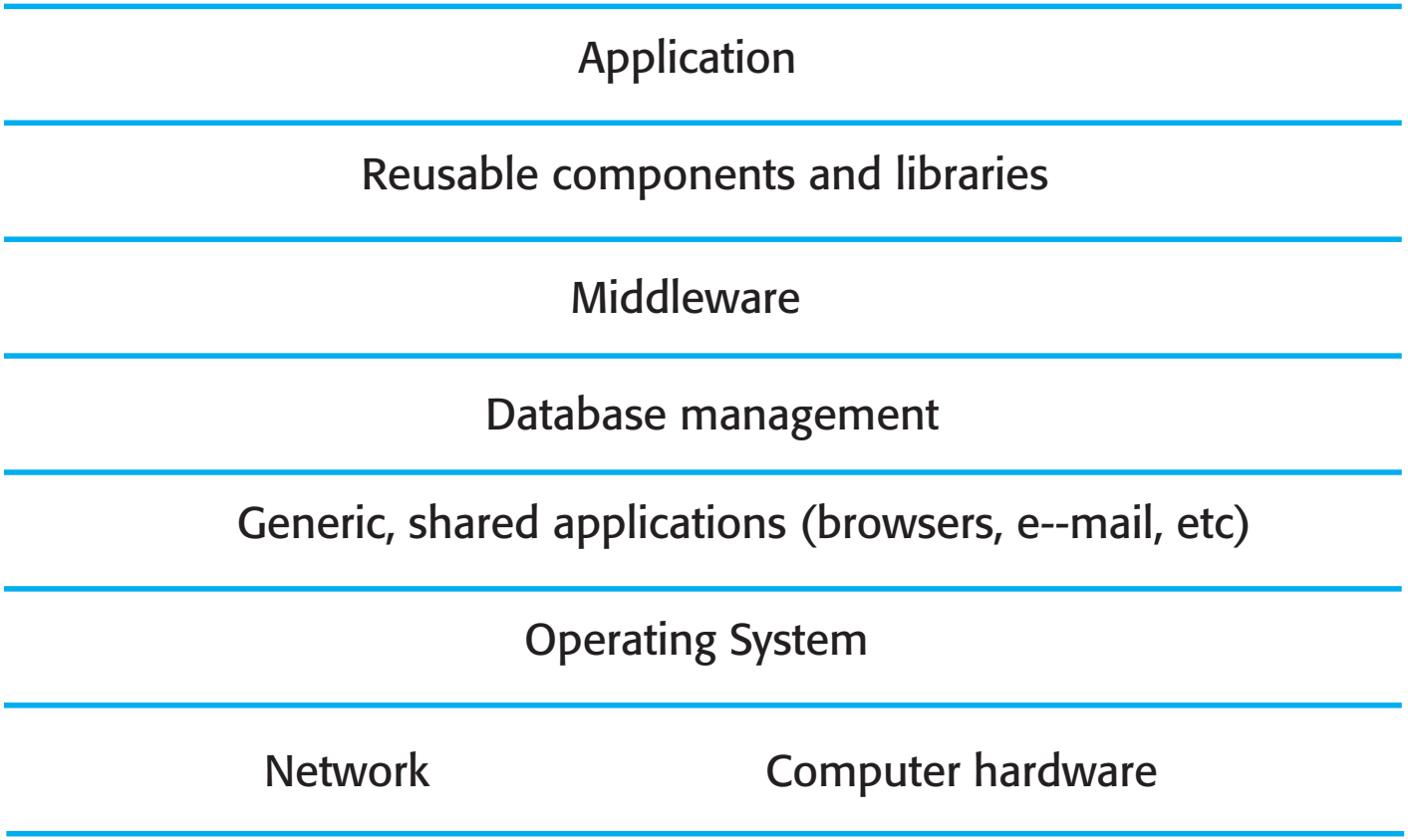
Security dimensions

- *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.

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.

13.1 - Security and dependability

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.

Threat types

- *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

13.3 Security requirements

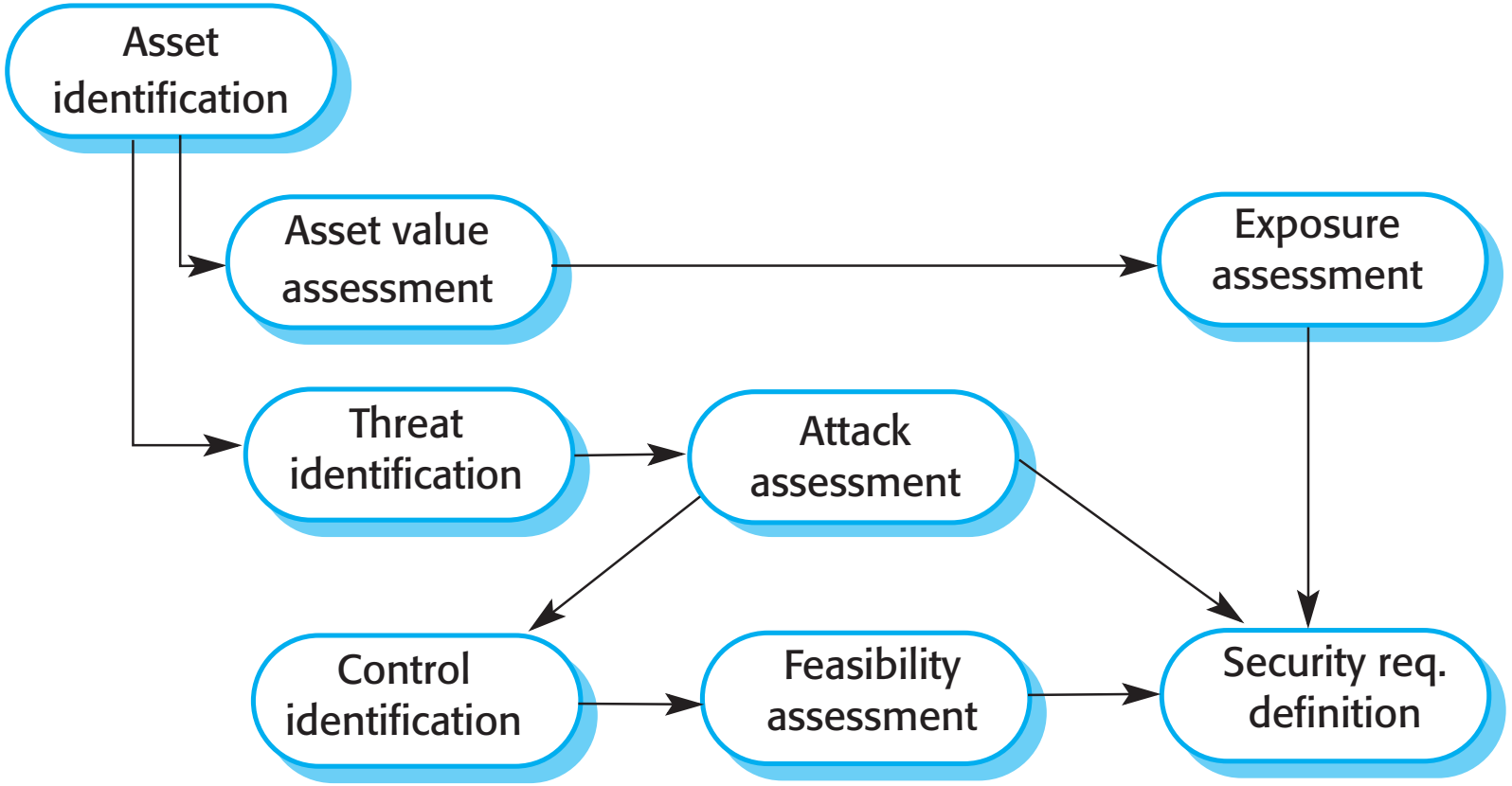
Types of security requirement

- *Identification requirements* – do we need to identify users?
- *Authentication requirements* – how users are identified
- *Authorization requirements* – privileges / access control
- *Immunity requirements* – protect against worms/viruses
- *Integrity requirements* – how to avoid data corruption
- *Intrusion detection requirements* – detect attacks
- *Non-repudiation requirements* – ensure a party in a tranaction cannot deny involvement in transaction
- *Privacy requirements* – how data privacy is maintained
- *Security auditing requirements* – how audit trail created
- *System maintenance security requirements* – how the application prevents authorized changes from defeating security.

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.

The preliminary risk assessment process for security requirements



Security risk assessment

- Asset identification
  - Identify the key system assets (or services) that have to be protected.
- Asset value assessment
  - Estimate the value of the identified assets.
- Exposure assessment
  - Assess the potential losses associated with each asset.
- Threat identification
  - Identify the most probable threats to the system assets

Security risk assessment

- Attack assessment
  - Decompose threats into possible attacks on the system and the ways that these may occur.
- Control identification
  - Propose the controls that may be put in place to protect an asset.
- Feasibility assessment
  - Assess the technical feasibility and cost of the controls.
- Security requirements definition
  - Define system security requirements. These can be infrastructure or application system requirements.

Asset analysis in a preliminary risk 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.

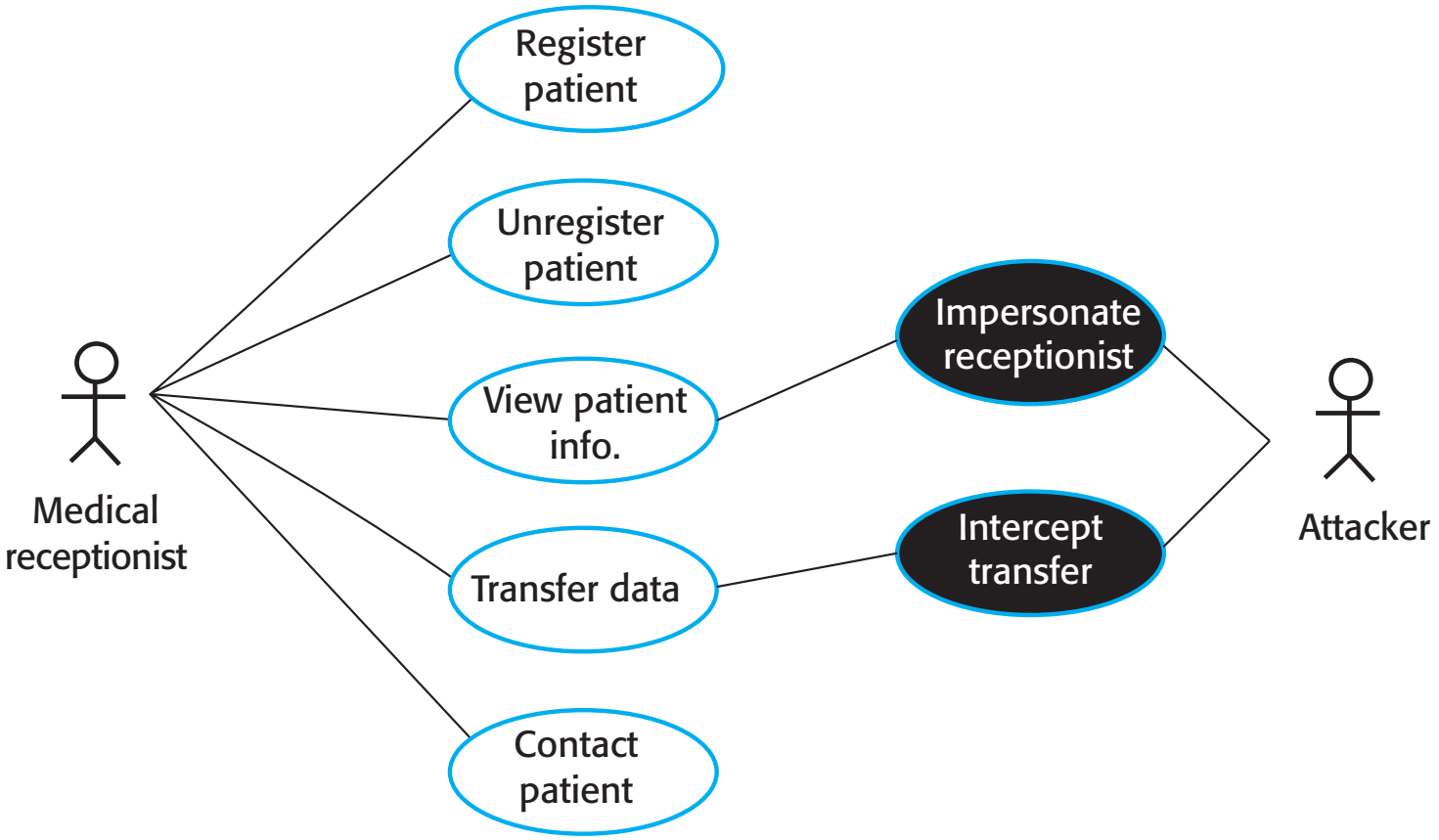
Security requirements for the Mentcare system

- 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



Mentcare use case – Transfer data

Mentcare system: Transfer data	
Actors	Medical receptionist, Patient records system (PRS)
Description	A receptionist may transfer data from the Mentcare system to a general patient record database that is maintained by a health authority. The information transferred may either be updated personal information (address, phone number, etc.) or a summary of the patient's diagnosis and treatment.
Data	Patient's personal information, treatment summary.
Stimulus	User command issued by medical receptionist.
Response	Confirmation that PRS has been updated.
Comments	The receptionist must have appropriate security permissions to access the patient information and the PRS.

Mentcare misuse case: Intercept transfer

Mentcare system: Intercept transfer (Misuse case)	
Actors	Medical receptionist, Patient records system (PRS), Attacker
Description	A receptionist transfers data from his or her PC to the Mentcare system on the server. An attacker intercepts the data transfer and takes a copy of that data.
Data (assets)	Patient's personal information, treatment summary
Attacks	A network monitor is added to the system and packets from the receptionist to the server are intercepted. A spoof server is set up between the receptionist and the database server so that receptionist believes they are interacting with the real system.

Misuse case: Intercept transfer

Mentcare system: Intercept transfer (Misuse case)	
Mitigations	All networking equipment must be maintained in a locked room. Engineers accessing the equipment must be accredited. All data transfers between the client and server must be encrypted. Certificate-based client-server communication must be used
Requirements	All communications between the client and the server must use the Secure Socket Layer (SSL). The https protocol uses certificate based authentication and encryption.

Key points

- Security engineering is concerned with how to develop systems that can resist malicious attacks
- Security threats can be threats to confidentiality, integrity or availability of a system or its data
- Security risk management is concerned with assessing possible losses from attacks and deriving security requirements to minimise losses
- To specify security requirements, you should identify the assets that are to be protected and define how security techniques and technology should be used to protect these assets.

About Week 11 – March 30th

- Reading Assignment
  - Chapters 22 (“Project Management”) and 23 (“Project Planning”)
- TEAM Homework Assignments
  - HW 8 – Checkpoint Meetings. It is up to you to schedule. Up to ten points earned for each meeting held on time. 3 meetings for total of 30 points.
    - Second checkpoint with TA – DUE April 15th
    - No need to respond in Sakai – we will track and grade.