COIS3040 Lecture 11

Security

What is Security?

- Security is a measure of the system's ability to protect data and information from unauthorized access while still providing access to people and systems that are authorized.
- An action taken against a computer system with the intention of doing harm is called an *attack* and can take a number of forms.
- It may be an unauthorized attempt to access data or services or to modify data, or it may be intended to deny services to legitimate users.

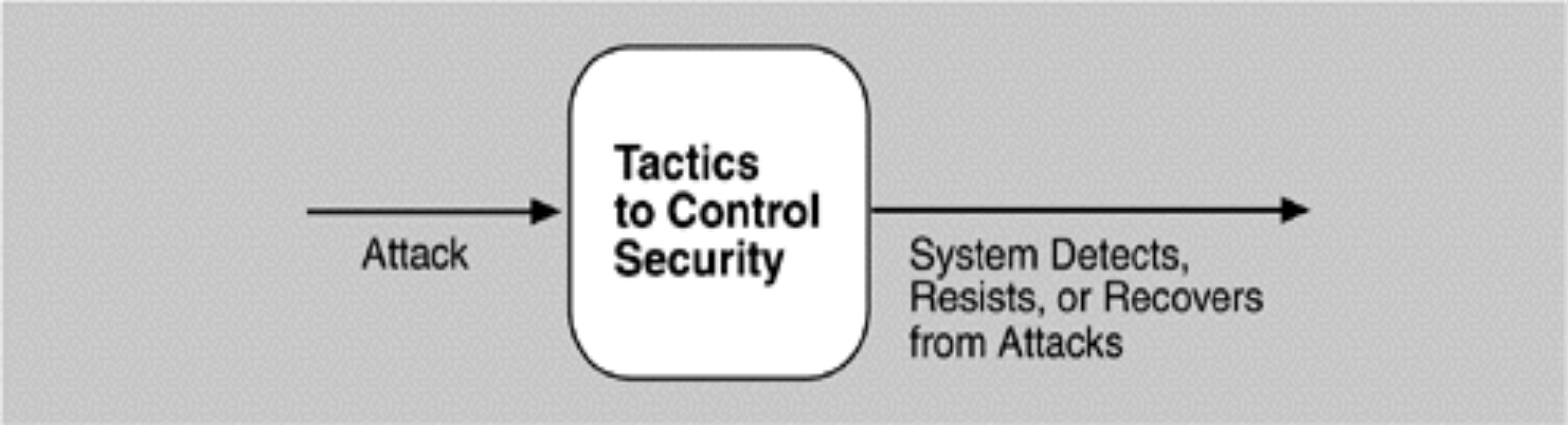
What is Security?

- Security has three main characteristics, called CIA:
  - **Confidentiality** is the property that data or services are protected from unauthorized access. For example, a hacker cannot access your income tax returns on a government computer.
  - **Integrity** is the property that data or services are not subject to unauthorized manipulation. For example, your grade has not been changed since your instructor assigned it.
  - **Availability** is the property that the system will be available for legitimate use. For example, a denial-of-service attack won't prevent you from ordering a book from an online bookstore.

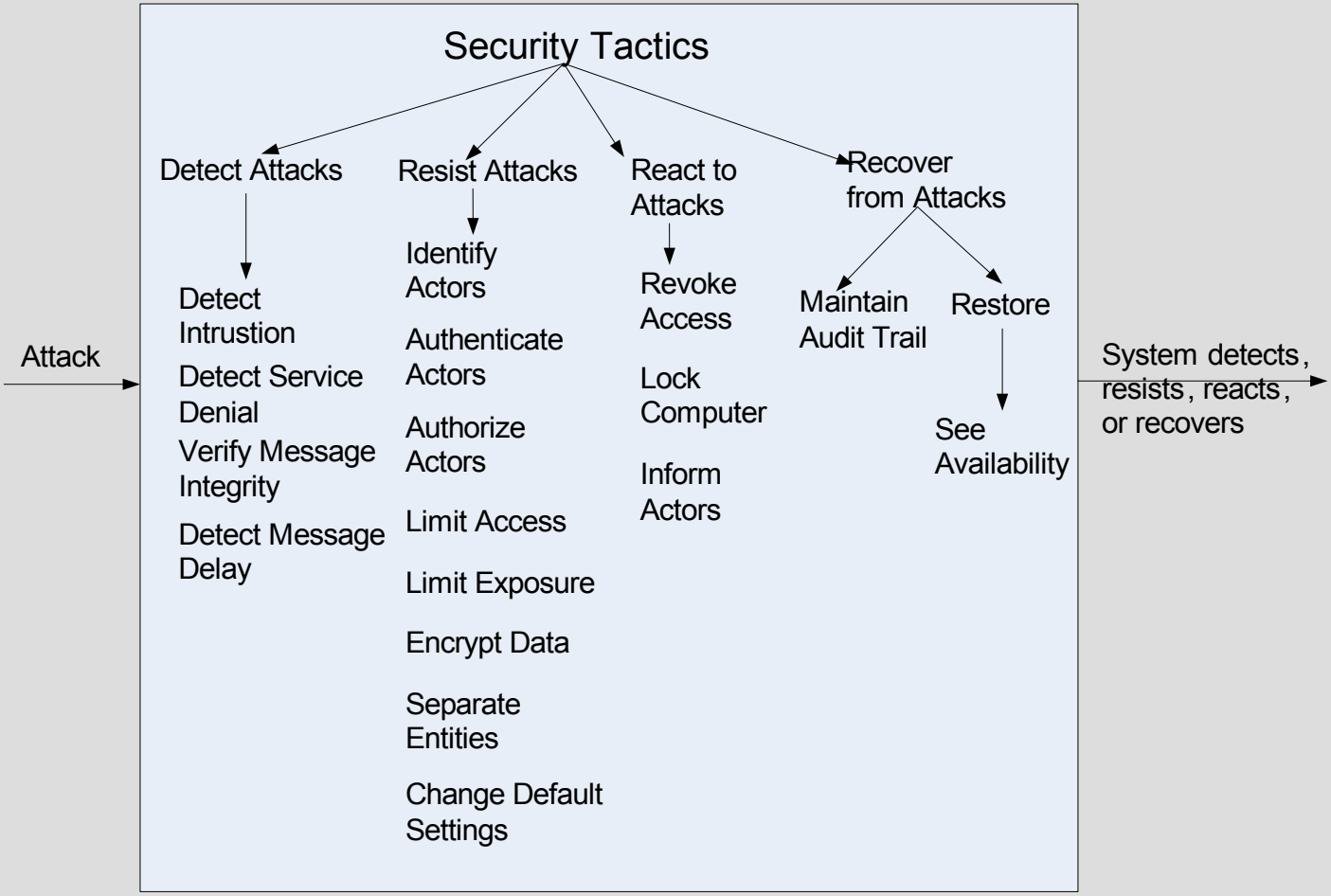
Sample Concrete Security Scenario

- A disgruntled employee from a remote location attempts to modify the pay rate table during normal operations. The system maintains an audit trail and the correct data is restored within a day.

Goal of Security Tactics



Security Tactics



Detect Attacks

- Detect Intrusion: compare network traffic or service request patterns *within* a system to a set of signatures or known patterns of malicious behaviour stored in a database.
- Detect Service Denial: comparison of the pattern or signature of network traffic *coming into* a system to historic profiles of known Denial of Service (DoS) attacks.
- Verify Message Integrity: use techniques such as checksums or hash values to verify the integrity of messages, resource files, deployment files, and configuration files.
- Detect Message Delay: checking the time that it takes to deliver a message, it is possible to detect suspicious timing behavior.

Resist Attacks

- Identify Actors: identify the source of any external input to the system.
- Authenticate Actors: ensure that an actor (user or a remote computer) is actually who or what it purports to be.
- Authorize Actors: ensuring that an authenticated actor has the rights to access and modify either data or services.
- Limit Access: limiting access to resources such as memory, network connections, or access points.

Resist Attacks

- Limit Exposure: minimize the attack surface of a system by having the fewest possible number of access points.
- Encrypt Data: apply some form of encryption to data and to communication.
- Separate Entities: can be done through physical separation on different servers attached to different networks, the use of virtual machines, or an “air gap”.
- Change Default Settings: Force the user to change settings assigned by default.

React to Attacks

- Revoke Access: limit access to sensitive resources, even for normally legitimate users and uses, if an attack is suspected.
- Lock Computer: limit access to a resource if there are repeated failed attempts to access it.
- Inform Actors: notify operators, other personnel, or cooperating systems when an attack is suspected or detected.

Recover From Attacks

- In addition to the Availability tactics for recovery of failed resources there is Audit.
- Audit: keep a record of user and system actions and their effects, to help trace the actions of, and to identify, an attacker.

Testability

What is Testability?

- Software testability refers to the ease with which software can be made to demonstrate its faults through (typically execution-based) testing.
- Specifically, testability refers to the probability, assuming that the software has at least one fault, that it will fail on its *next* test execution.
- If a fault is present in a system, then we want it to fail during testing as quickly as possible.

What is Testability?

- For a system to be properly testable, it must be possible to *control* each component's inputs (and possibly manipulate its internal state) and then to *observe* its outputs (and possibly its internal state).

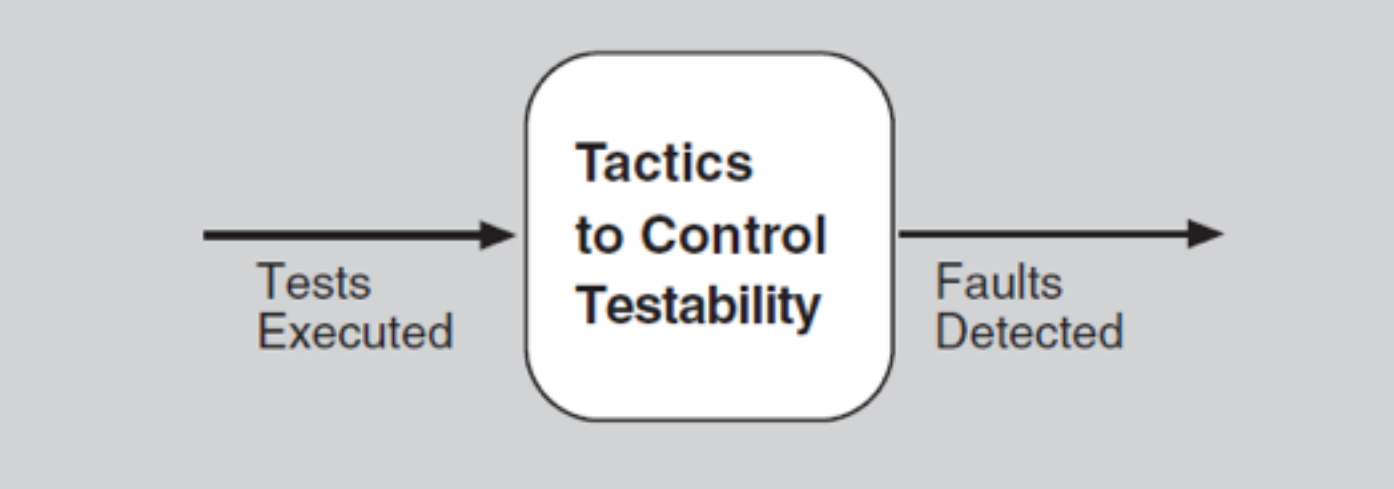
Sample Concrete Testability Scenario

- The unit tester completes a code unit during development and performs a test sequence whose results are captured and that gives 85% path coverage within 3 hours of testing.

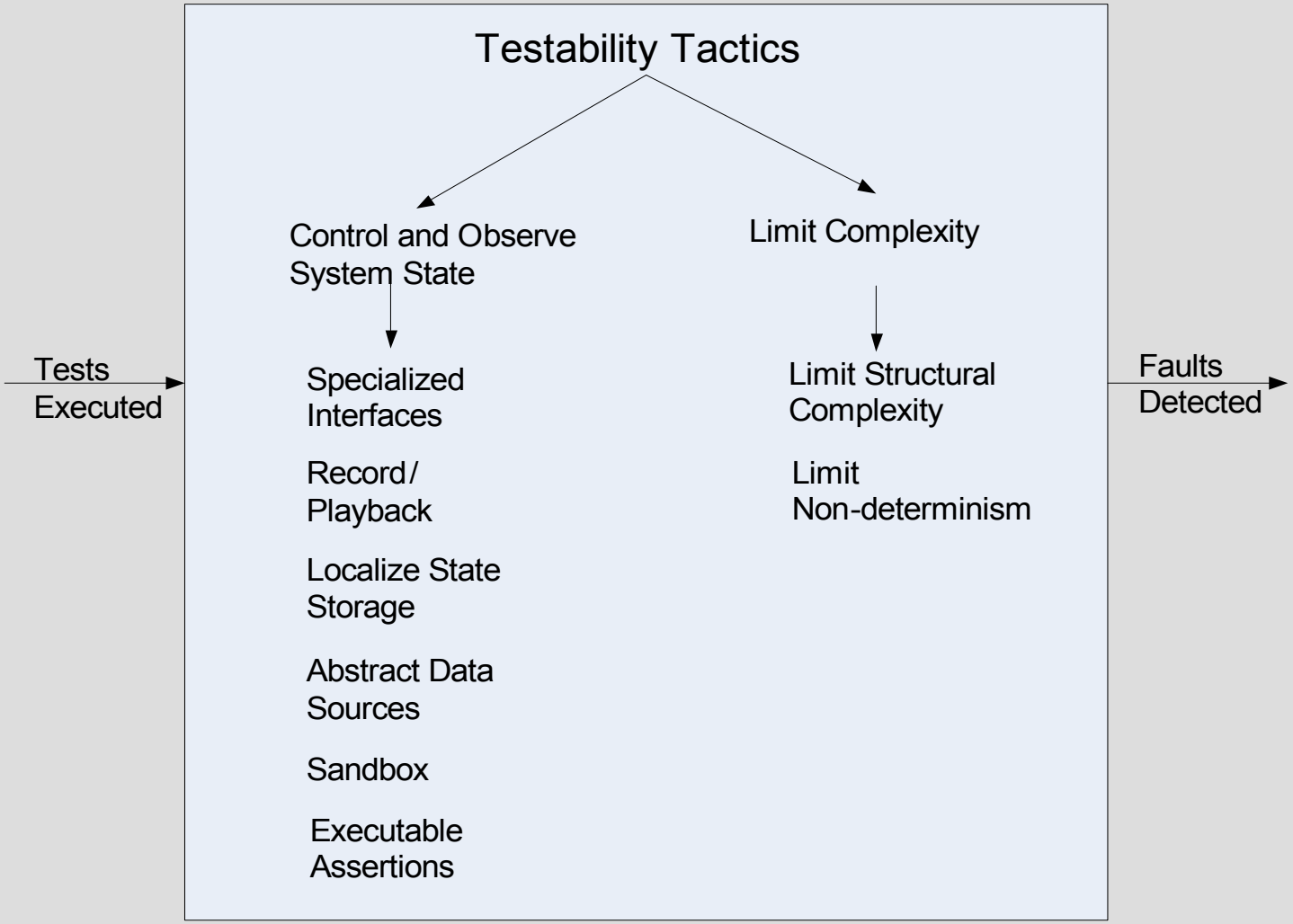
Goal of Testability Tactics

- The goal of tactics for testability is to allow for easier testing when an increment of software development has completed.
- Anything the architect can do to reduce the high cost of testing will yield a significant benefit.
- There are two categories of tactics for testability:
  - The first category deals with adding controllability and observability to the system.
  - The second deals with limiting complexity in the system's design.

Goal of Testability Tactics



Testability Tactics



Control and Observe System State

- Specialized Interfaces: to control or capture variable values for a component either through a test harness or through normal execution. Example: get/set, reset, report.
- Record/Playback: capturing information crossing an interface and using it as input for further testing.
- Localize State Storage: To start a system, subsystem, or module in an arbitrary state for a test, it is most convenient if that state is stored in a single place.

Control and Observe System State

- Abstract Data Sources: Controlling input data makes it easier to test. Abstracting the interfaces lets you substitute test data more easily.
- Sandbox: isolate the system from the real world to enable experimentation that is unconstrained by the worry about having to undo the consequences of the experiment.
- Executable Assertions: assertions are (usually) hand coded and placed at desired locations to indicate when and where a program is in a faulty state.

Limit Complexity

- Limit Structural Complexity: avoiding or resolving cyclic dependencies between components, isolating and encapsulating dependencies on the external environment, and reducing dependencies between components in general.
- Limit Non-determinism: finding all the sources of non-determinism, such as unconstrained parallelism, and weeding them out as far as possible.