SSE-LAB 2: Secure Software Design

Riccardo Scandariato
Institute of Software Security, TUHH, Germany
ric***do . scanda***to @ tuhh.de

Lecturer: Nicolás Díaz Ferreyra

Agenda

- Brief conceptual review
  - Architectural Patterns
  - Architectural Tactics
- Practical case
- Security requirements
- Identification of security tactics
- Identification of security patterns

Architectural Patterns

An architectural pattern is a package of design decisions that:

- Is found **repeatedly** in practice.
- Has known properties that permit **reuse** (can be named, formalized)
- Describes a **class** of architectures.

Patterns are found in practice, not invented, they are discovered!

- There will never be a complete list of patterns.
- Patterns spontaneously emerge in reaction of **environmental conditions** ⇒ when these conditions change new patterns emerge!

Architectural design seldom starts from scratch. It is often a process process where patterns are **selected**, **tailored** and **combined**.

Model-View-Controller (MVC) Pattern

Context:

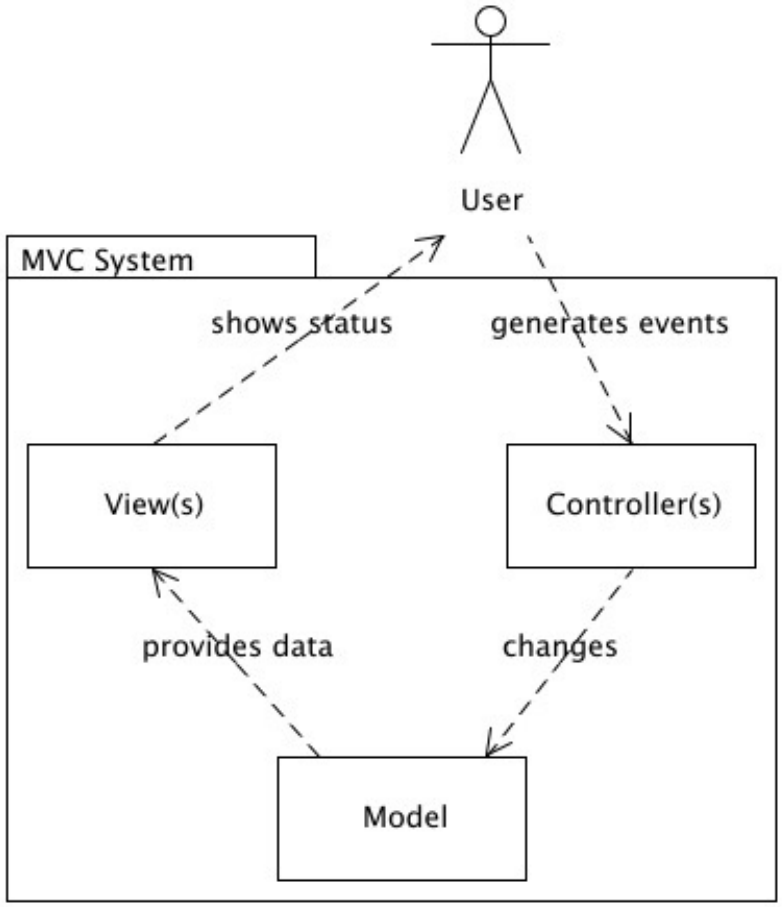
For interactive GUI based systems, to keep modifications to the user interface separate from the rest of the system, and support multiple or alternative views of computing results or current state of data.

Problem:

- How can user interface functionality be kept **separate** from application functionality and still be responsive to user input, or changes in the underlying application's data?
- How can **multiple views** of user interface be created, maintained, and coordinated when the underlying application data changes?

Model-View-Controller (MVC) Pattern

Solution:



http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/mvc.html

Model-View-Controller (MVC) Pattern

TABLE 13.3 Model-View-Controller Pattern Solution

Overview	The MVC pattern breaks system functionality into three components: a model, a view, and a controller that mediates between the model and the view.
Elements	The *model* is a representation of the application data or state, and it contains (or provides an interface to) application logic. The *view* is a user interface component that either produces a representation of the model for the user or allows for some form of user input, or both. The *controller* manages the interaction between the model and the view, translating user actions into changes to the model or changes to the view.
Relations	The *notifies* relation connects instances of model, view, and controller, notifying elements of relevant state changes.
Constraints	There must be at least one instance each of model, view, and controller. The model component should not interact directly with the controller.
Weaknesses	The complexity may not be worth it for simple user interfaces. The model, view, and controller abstractions may not be good fits for some user interface toolkits.

Bass, L., Clemens, P., Kazman, R.: Software Architecture in Practice.
SEI Series in Software Engineering, Addison-Wesley, 2 edn. (2003)

Architectural Tactics

The architecture of a complex system can contain multiple patterns.

- **Tactics** are building blocks (of design) from which architectural patterns are created.
- Patterns package tactics ⇒ most of patterns consists of (are constructed from) several different tactics.

Tactics are rather simple ideas. They are **fine grained** but **abstract** and thus (as opposed to patterns) expressible in just a few sentences.

- A **simple idea** explained in one sentence and a few examples!

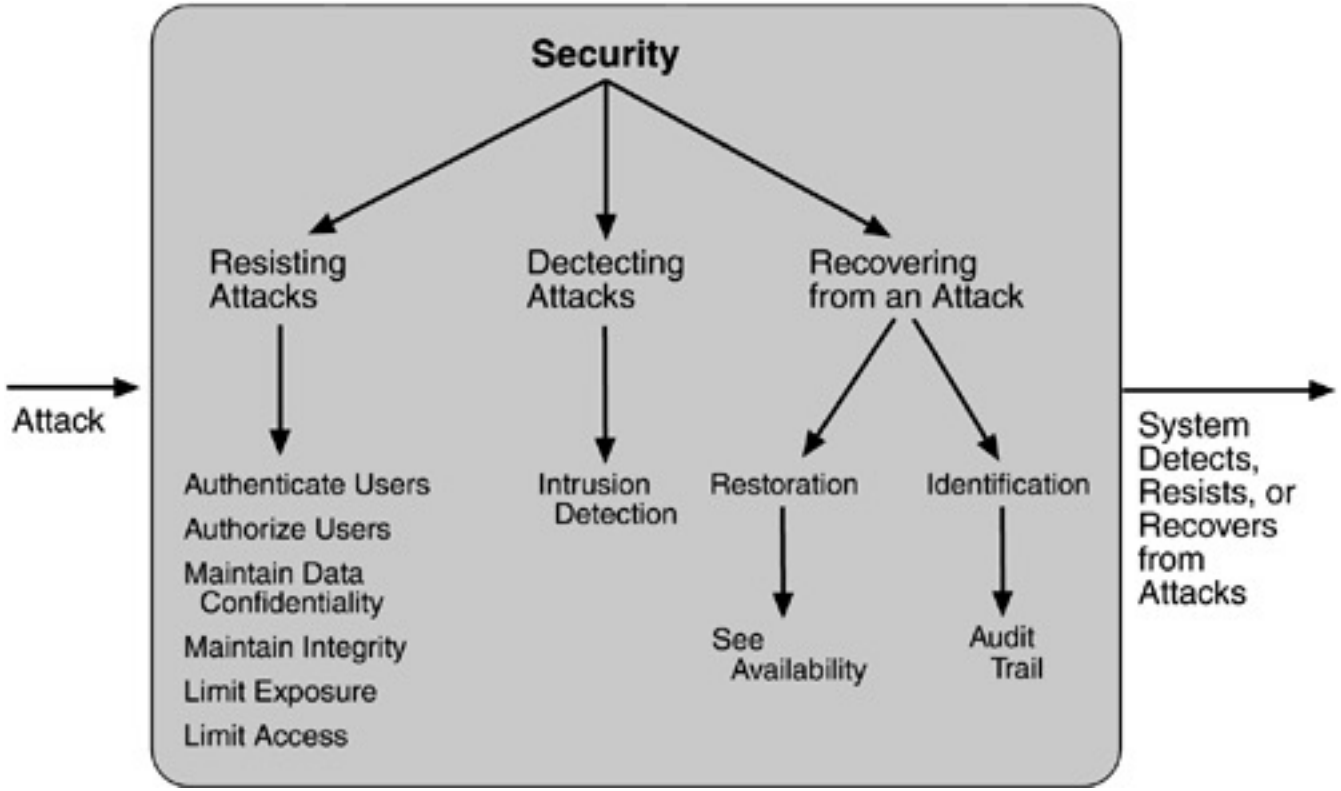
> *"Authentication is ensuring that a user or remote computer is actually who she purports to be. Passwords, one-time passwords, digital certificates, and biometric identifications provide authentication"*

Tactics and Patterns: Relationship

	Modifiability										
	Increase Cohesion		Reduce Coupling					Defer Binding Time			
Pattern	Increase Semantic Coherence	Abstract Common Services	Encapsulate	Use a Wrapper	Restrict Comm. Paths	Use an Intermediary	Raise the Abstraction Level	Use Runtime Registration	Use Startup-Time Binding	Use Runtime Binding
Layered	X	X	X		X	X	X			
Pipes and Filters	X		X		X	X			X	
Blackboard	X	X			X	X	X	X		X
Broker	X	X	X		X	X	X	X		
Model View Controller	X		X			X				X
Presentation Abstraction Control	X		X			X	X			
Microkernel	X	X	X		X	X				
Reflection	X		X							

Bass, L., Clemens, P., Kazman, R.: Software Architecture in Practice. SEI Series in Software Engineering, Addison-Wesley, 2 edn. (2003) -> Section 13.3

Security Tactics



Bass, L., Clemens, P., Kazman, R.: Software Architecture in Practice. SEI Series in Software Engineering, Addison-Wesley, 2 edn. (2003) -> Section 9.2

Security Tactics

- Maintain integrity: Data should be delivered as intended. It can have redundant information encoded in it, such as checksums or hash results, which can be encrypted either along with or independently from the original data.
- Limit exposure: Attacks typically depend on exploiting a single weakness to attack all data and services on a host. The architect can design the allocation of services to hosts so that limited services are available on each host.
- Authorize users: Authorization is ensuring that an authenticated user has the rights to access and modify either data or services. This is usually managed by providing some access control patterns within a system. Access control can be by user or by user class. Classes of users can be defined by user groups, by user roles, or by lists of individuals.

Security tactics can be used in combination to **achieve security goals** like *Confidentiality*, *Integrity*, *Availability*,… **CIA+**

Lab Tasks

We have outlined a **7 security requirements** for the Metaverse:

- Contextual information.
- Additional constraints.
- Security pattern catalogue.

You must select adequate patterns and tactics for each requirement.

1. Identify security tactics.
2. Identify security patterns.
3. Discuss you your solution in pairs.

Questions ?