Security Patterns

Acknowledgements: Ronald Wassermann

Motivation

- Today’s systems have various communication features
- Many security-critical dependencies exist
- Security is a non-functional requirement that is difficult to evaluate (e.g., metrics, etc.)
- Which security features are necessary in certain domains?

  It is difficult to design secure systems [Bis02]

  Expert knowledge is needed
Approach

- In order to overcome the knowledge gap among developers we use patterns to
  - provide relevant information in a structured way
  - convey experience
- We use a variation of the well-known design pattern template [Gam94] to present more security-specific information

Our goal is to enable the reuse of security knowledge

Security Principles (1)

- References ten guiding security principles [Viega and McGraw 2002]

Ten Security Principles

- Secure the weakest link.
- Practice defense in depth.
- Fail securely.
- Follow the principle of least privilege.
- Compartmentalize.
- Keep it simple.
- Promote privacy.
- Remember that hiding secrets is hard.
- Be reluctant to trust.
- Use your community resources.
Security Principles (2)

1. Secure the weakest link
   • Intruders will attack parts that are most likely to break
   • Identify and strengthen weak parts to improve overall security

2. Practice defense in depth
   • Implement overlapping security mechanisms
   • Every protection layer adds to overall security

3. Fail securely
   • Failures are not avoidable
   • Security flaws are often inherent to system failures
   • Plan failure modes that assure that the system’s security is not compromised by exceptional behavior

Security Principles (3)

4. Follow the principle of least privilege
   • Grant only the minimum set of permissions
   • Thereby reducing the risk of privilege-abuse

5. Compartmentalize
   • Structure your system in a way that protects different parts independently.
   • Reduces amount of damage that is caused by a security breach in one unit.

6. Keep it simple
   • Avoid unnecessary complexity
   • Usability is an important part of simple design
Security Principles (4)

7. Promote privacy
   • Minimize the information that can be gathered about a system and its users
   • Use misinformation to deter attackers

8. Remember that hiding secrets is hard
   • A system’s security depends on certain secrets being kept
   • Be aware of critical information that could compromise security

9. Be reluctant to trust
   • Do not extend trust unnecessarily
   • Design systems that mistrust information of other parts

Security Principles (5)

10. Use your community resources
    • Public scrutiny improves code as it exploits weaknesses and errors
    • Code written by individuals is usually less secure

Tradeoffs:
- Compartmentalize (P5) ↔ Keep it simple (P6)
- Usability (P6) ↔ Promote privacy (P7)
- Practice defense in depth (P2) ↔ Keep it simple (P6)
Patterns

“Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.” Christopher Alexander [Ale77]

• Essential elements of a pattern [GHJV94]
  ➔ Name
  ➔ Problem
  ➔ Consequences
  ➔ Solution

• Benefits
  ➔ Improves communication and establish terminology
  ➔ Provides structured information and captures knowledge
  ➔ Unifies design and improves comprehensibility

Design Patterns

• “Design patterns are patterns that express solutions to recurring software design problems in terms of objects and interfaces” [GHJV94].

• Gamma et al. propose template structure
Previous work

- Other pattern-based approaches to security problems
  - Fernandez [Fer02]
    - Collection of security patterns (using few UML diagrams)
  - Kienzle et al. [KETE02]
    - Tutorial for writing security patterns
  - Schumacher and Roedig [SR01]
    - Propose the use of patterns in security engineering (no specific template)
  - Yoder and Barcalow [YB97]
    - Collection of security patterns (no diagrams)

Security Pattern Definition

- How can the information be structured to reflect the needs of the security domain?
  
  "A Security Pattern describes a particular recurring security problem that arises in a specific context and presents a well-proven generic scheme for its solution."

  Schumacher and Roedig [SR01]

  New template that is customized for use in the development of secure software
Security Pattern Template

- We use the design pattern template of Gamma et al. [GHJV94] and extend and modify it to fit our needs:

<table>
<thead>
<tr>
<th>Design Pattern Template</th>
<th>Security Pattern Template</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern name and Classification</td>
<td>Pattern name and Classification</td>
</tr>
<tr>
<td>Intent</td>
<td>Intent</td>
</tr>
<tr>
<td>Also known as</td>
<td>Also known as</td>
</tr>
<tr>
<td>Motivation</td>
<td>Motivation</td>
</tr>
<tr>
<td>Applicability</td>
<td>Applicability</td>
</tr>
<tr>
<td>Structure</td>
<td>Structure</td>
</tr>
<tr>
<td>Participants</td>
<td>Participants</td>
</tr>
<tr>
<td>Collaborations</td>
<td>Collaborations</td>
</tr>
<tr>
<td>Consequences</td>
<td>Behavior</td>
</tr>
<tr>
<td>Implementation</td>
<td>Constraints</td>
</tr>
<tr>
<td>Sample Code</td>
<td>Consequences</td>
</tr>
<tr>
<td>Known Uses</td>
<td>Known Uses</td>
</tr>
<tr>
<td>Related Patterns</td>
<td>Related Security Patterns</td>
</tr>
<tr>
<td>Related Design Patterns</td>
<td>Related Security Principles</td>
</tr>
<tr>
<td>Related Security Principles</td>
<td></td>
</tr>
</tbody>
</table>

Classification

- Similar to design patterns [GHJV94] we organize security patterns by purpose in:
  - structural,
  - behavioral, and
  - creational patterns

- Furthermore, we denote the following abstraction levels for patterns:
  - Application level (objects are deployed at a client)
  - Host level (objects are running on a server)
  - Network level (objects are distributed over a network)
Overview

• Several patterns were identified by the security pattern community [Fer01][YB97]

• We present the **Single Access Point, Check Point** and the **Role-Based Access Control** pattern

---

Single Access Point (SAP) (1)

• SAP was introduced by Yoder and Barcalow [YB97]

• **Name and Classification**
  – Single Access Point, structural pattern

• **Intent**
  – Proposes single interface to the system to improve control

• **Also known as**
  – Guard Door, Login Window, One Way In, or Validation Screen

• **Motivation**
  – Various access points and hidden back doors make protection difficult
  – Monitoring of external communication should be possible

• **Applicability**
  – For self-contained systems that communicate with external entities
  – Several entry points for greater flexibility cannot be realized with this pattern
Single Access Point (SAP) (2)

- **Structure**

![Diagram of Single Access Point (SAP) (2)]

Single Access Point (SAP) (3)

- **Participants**
  - External Entities
  - Internal Entities
  - Single Access Point

- **Collaborations**
  - SingleAccessPoint is contacted by external entities
  - It works as mediator

- **Behavior**

![Diagram of Single Access Point (SAP) (3)]
Single Access Point (SAP) (4)

• **Constraints**
  - **Authenticity**: A message that is directly sent to an internal component originates either from a system internal component or the SAP.
  - **Confidentiality**: Communication between internal components is not disclosed to outside entities.
  - **Integrity**: Messages inside the system cannot be modified by external entities.

• **Consequences**
  - **Accountability**: The SAP could perform logging tasks and thereby improve accountability.
  - **Confidentiality, Integrity**: SAP provides a place for monitoring of communication.
  - **Availability**: If the SAP cannot handle all accesses, availability might be reduced; information for the detection of DoS attacks can be gathered.
  - **Performance**: Substantial logging operations at the SAP can affect the performance of a system.

Consequences (cont’d)

• **Cost**: Depending on the extend of communication with external parties, development can be more difficult and expensive.
• **Manageability**: Security-code not be scattered over the entire system.
• **Usability**: Access to system might be more inconvenient for a user.

• **Known uses**
  - Linux telnet application
  - Windows NT login application

• **Related Security Patterns**
  - Check Point (monitors communication that passes the SAP)
  - Role-Based Access Control (is initialized upon login)
  - Session (is created upon login)

• **Related Design Patterns**
  - Singleton (to implement the SAP)
Single Access Point (SAP) (5)

- Related Security Principles
  - **Principle 1**: Secure the weakest link (the pattern considers the interface to external entities to be a possible weakness)
  - **Principle 6**: Keep it simple (the SAP pattern propagates a simplification of the systems access)
  - **Principle 9**: Reluctance to trust

Check Point (1)

- The Check Point pattern was first presented in a framework by Yoder and Barcalow [YB97]
- **Name and Classification**
  - Check Point, structural pattern
- **Intent**
  - A structure for checking incoming requests and handling violations
- **Also known as**
  - Access Verification, Authentication and Authorization, Holding off hackers, Validation and Penalization, or Make the punishment fit the crime
- **Motivation**
  - Systems that communicate with external entities have to take into account illegal requests and attacks
  - Monitoring and access validation is necessary
Check Point (2)

- **Structure**

Check Point (3)

- **Applicability**
  - Check Point can be applied in any system that needs to monitor communication
  - In order to perform checks the system needs a security policy

- **Participants**
  - Check Point
  - Countermeasure
  - Security Policy

- **Collaborations**
  - The Check Point monitors if messages are consistent with the Security Policy
  - Countermeasures are triggered if necessary
Check Point (4)

**Behavior**

- **Constraints**
  - **Authenticity:** Check Point’s policy requests may only be answered by the Security Policy object.
  - **Integrity:** Messages that are sent between Check Point and Security Policy cannot be modified.
  - **Confidentiality, Integrity:** External requests are not forwarded until the Security Policy approves it.
Check Point (6)

- **Consequences**
  - **Confidentiality**: unauthorized access can be prevented
  - **Integrity**: malicious modification can be filtered
  - **Availability**: Check Point can trigger countermeasures to prevent DoS attacks (e.g. delays, blacklists)
  - **Performance**: Complex checks slow down the system
  - **Cost**: Development of an effective check algorithm is difficult and expensive
  - **Manageability**: combining security code in one place simplifies maintenance
  - **Usability**: depending on the check algorithm, harmless requests may be blocked if they match a certain pattern

Check Point (7)

- **Known Uses**
  - During the login to an ftp server a Check Point is usually used to control the access of users

- **Related Security Patterns**
  - Single Access Point
  - Session
  - Role-Based Access Control

- **Related Design Patterns**
  - Strategy (decouple Check Point from actual implementation of the security policy)
Check Point (8)

- **Related Principles**
  - Principle 1: *Secure the weakest link* (by applying the Check Point)
  - Principle 2: *Practice defense in depth* (further security measures should be considered in addition to the Check Point)
  - Principle 4: *Principle of least privilege* (should be reflected in the Security Policy)
  - Principle 9: *Reluctance to trust* (that all requests are harmless)

Role-Based Access Control (1)

- The Role-Based Access Control pattern was first presented in [YB97] and later in [Fer01]

- **Name and Classification**
  - Role-Based Access Control, structural pattern

- **Intent**
  - Facilitates the representation and maintenance of access structures

- **Also known as**
  - Roles, Actors, Groups, Projects, Profiles, Jobs, or User Types

- **Motivation**
  - The use of resources usually underlies certain restrictions
  - In order to facilitate enforcement restrictions need to be represented in some structure inside the system

- **Applicability**
  - Applicable in any system that restricts subjects’ access on resources
Role-Based Access Control (2)

- **Structure**

![Diagram](image)

Role-Based Access Control (3)

- **Participants**
  - ProtectionObject
  - Right
  - Role
  - Roles
  - User

- **Collaborations**
  - Roles are associated to a set of Objects; a Right object defines the properties of each relationship (type, constraint, transferable)
  - Each user can be associated to Roles that determine his/her privileges
  - This information about access privileges can be queried by other system components
Role-Based Access Control (4)

• Behavior

Role-Based Access Control (5)

• Constraints
  – To be determined

• Consequences
  – Confidentiality, Integrity: An right structure enables definition of access privileges that protect confidentiality and integrity
  – Availability: Restriction of access to resources enhances availability
  – Performance: can be improved by reducing the overall amount of relationships that reflect the access structure
  – Cost: higher development cost, reduced maintenance
  – Manageability: Maintenance is simplified as subjects can be managed in groups

• Known Uses
  – Several applications, including various Database Management Systems (DBMS) and Windows 2000
Role-Based Access Control (6)

- **Related Security Patterns**
  - Check Point
  - Session
  - Limited View

- **Related Design Patterns**
  - Strategy (implement different behavior depending on users role)
  - Observer (keep structure consistent)

- **Related Security Principles**
  - Principle 4: *Principle of least privilege* (should be reflected in the right structure)
  - Principle 6: *Keep it simple* (by reducing relationships)
  - Principle 7: *Promote privacy* (by restricting access)

Conclusions

- Security patterns help to keep track of non-functional security requirements from the beginning of design
- A well-structured template can enhance the effectiveness of the pattern approach
- Avoiding errors is extremely important in security critical applications
Open problems

- Facilitate formal verification during the application of security patterns by providing formalized constraints that can be checked against the system model
- Continue to scan for security patterns
- Domain-specific security patterns
  - Medical applications?
  - Automotive?
- Explore how extending modeling languages (such as UML) can/should be extended for security.

References

- [GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
Appendix

Overview of Gamma et al. Design Patterns for Reference

Creational Patterns

- **Factory Method:**
  - method in a derived class creates associations
- **Abstract Factory:**
  - Factory for building related objects
- **Builder:**
  - Factory for building complex objects incrementally
- **Prototype:**
  - Factory for cloning new instances from a prototype
- **Singleton:**
  - Factory for a singular (sole) instance
Structural Patterns:

- **Adapter:**
  - Translator adapts a server interface for a client
- **Bridge:**
  - Abstraction for binding one of many implementations
- **Composite:**
  - Structure for building recursive aggregations
- **Decorator:**
  - Decorator extends an object transparently
- **Facade:**
  - Simplifies the interface for a subsystem
- **Flyweight:**
  - Many fine-grained objects shared efficiently.
- **Proxy:**
  - One object approximates another

Behavioral Patterns

- **Chain of Responsibility**
  - Request delegated to the responsible service provider
- **Command:**
  - Request is first-class object
- **Iterator:**
  - Aggregate elements are accessed sequentially
- **Interpreter:**
  - Language interpreter for a small grammar
- **Mediator:**
  - Coordinates interactions between its associates
- **Memento:**
  - Snapshot captures and restores object states privately
- **Observer:**
  - Dependents update automatically when subject changes
- **State:**
  - Object whose behavior depends on its state
Behavior Patterns (more)

- **Strategy:**
  - Abstraction for selecting one of many algorithms
- **Template Method:**
  - Algorithm with some steps supplied by a derived class
- **Visitor:**
  - Operations applied to elements of a *heterogeneous* object structure