Topics

1. Design Patterns
2. Security Patterns
3. MTA Architecture Example
4. OpenSSH Architecture Example
5. PatternShare Patterns
Design Patterns

- A solution to a problem in context that can be re-used in different ways in software.
  - Standard name
  - Concise summary of problem
  - Description of solution, usually with UML
  - Examples of pattern in use
- Pattern variations
  - Architectural patterns
  - Antipatterns
  - Pattern languages
Security Patterns

- A solution to a recurring security problem.
  - Often are not design patterns with UML.

- Scope
  - Design principle
  - Architectural
  - Procedural
  - Design
Mail Transport Agent

MTA Requirements

- Receive mail from local users
- Receive mail from remote users
- Deliver mail to local users
- Send mail to remote users

Figure 2: An MTA with a mail queue
Security Requirements of a MTA

Administrative access required for

- Opening port 25 for SMTP.
- Delivery of mail to local users.

Administrative or group account needed to

- Create files in mail queue.
- Delete files from mail queue.
Architecture of Sendmail

Monolithic program performs all tasks

- Always runs as root.
- Drops privilege when unnecessary.
- Large code base.
- Complex configuration files.

Security history

- Debug command exploited by Morris worm.
- Many remote root exploits.
- Last root exploit in 2006.
Pattern: Compartmentalization

- **Problem**: A security failure in one part of a system allows another part of the system to be exploited.
- **Solution**: Put each part in a separate security domain. Even when the security of one part is compromised, the other parts remain secure.
Pattern: Distributed Responsibility

- **Problem**: A security failure in a compartment can change any data in that compartment. A compartment has both an interface that is at risk of a security failure and data that needs to be secure.

- **Solution**: Partition responsibility across compartments such that compartments that are likely to fail do not have data that needs to be secure. Assign responsibilities in such a way that several of them need to fail in order for the whole system to fail.

- **AKA**: Distributed Delegation
qmail queue

- All messages stored in queue first.
- Queue is accessible only by qmailq user.
- Subdirectories
  - pid: initial location of messages; files named after process ID of qmail-queue process
  - mess: second location of messages; files named after inode to ensure uniqueness
  - intd: contains envelopes for each message
  - todo: links to envelopes to be delivered
Pattern: Unique Atomic Chunks

- **Problem**: Many processes need to add information to a database concurrently. How do we ensure that multiple write operations are handled correctly and even if there is a crash no trace is left of the failure?

- **Solution**: Ensure that every write request is to a different location. Thus, different processes are never writing to the same file at the same time.
Pattern: Checkpointed System

- **Problem**: A component failure can result in loss or corruption of state information maintained by the failed component. How can we design a system so that its state can be recovered and restored to a known valid state in case a component fails?

- **Solution**: Design the system as a finite state machine. Make the state information persistent. Use configurations that provide ability to restart system from known valid state.
Architecture of Postfix

Uses many of same patterns as qmail.
- Compartmentalization
- Distributed responsibility

Adds focus on compatibility
- Uses mbox as well as maildir format.
- Violates Unique Atomic Chunks in mbox mode.

Adds focus on performance
- Qmail uses processes for single delivery.
- Postfix processes last longer.
- Uses Secure Preforking pattern.
Pattern: Secure Preforking

- **Problem**: The consequences of security compromise are worse in the case of daemon processes because they have a long lifetime. How can the vulnerability associated with daemon processes be reduced?

- **Solution**: Limit the lifetime of daemon processes and fork them again after a configurable, short lifetime. Run the daemons in a contained environment to minimize the exploits.
Architecture of OpenSSH

SSH login requires privileges to
- Open port 22.
- Access host key and password information.
- Open new pseudo-terminal devices.
- Change UID to that of user logging in.

Privilege separation
- Divide SSH into multiple processes.
- Operate at root, user, and unprivileged levels.
- Limit scope of exploit.
Architecture of OpenSSH

Monitor
- Privileged
- FSM that accepts requests from child for privileged actions.

Unprivileged
- Handles auth requests.

User-privileged
- Runs as user once user authenticated.
Patterns Followed
- Compartmentalization
- Least Privilege
- Reducing the TCB

Comparisons
- Much more interaction btw priv/unpriv than qmail or postfix.
- Very Secure FTPd (vsftpd) follows a privilege separation approach too.
Problem

The problem statement of the pattern.

Solution

The solution provided by the pattern.

Known Uses. Example implementation of the pattern.

Related Patterns. Patterns that are related.

Source: Source Repository

Tags: Tags for identifying the pattern

CSC 666: Secure Software Engineering

Munawar Hafiz, Security Patterns and Secure Architecture, OOPSLA ‘06
Viewpoints

- Business Arch. – CEO
- Integration Arch. – Ent. Arch.
- Application Arch. – Arch.
- Application Arch. Design
- Application Arch. – Developer
- Operational Arch. – Sys. Arch.

Interrogatives

- Function
- Data
- Network
- Test
- Everything
Audit Interceptor

Problem

A security audit allows auditors to reconcile actions or events that have taken place in the application with the policies that govern those actions. The audit logs have to be checked periodically to ensure that the actions that users have taken are in accordance with the actions allowed by the user’s privileges. The most important part is to record an audit trail and making sure that the audit trail helps proper auditing of appropriate events and user actions associated. Deviations must be identified from the audit reports and corrective actions have to be taken so that the deviations do not recur, either through code fixes or policy changes.

How can you make an auditing framework to easily support additions or changes to the auditing events?

Solution

Intercept business tier requests and responses. Create audit events based on the information in the request response pair using declarative mechanisms defined externally to the application. The declarative approach is crucial to maintainability of the application. This makes it easy to keep up with the changed corporate policies.

Known Uses. Audit interceptor with a JMS store to store audit information.
Problem

A malicious attacker might try to impersonate a legitimate user to have access to the user's resources. This could be even more serious if the impersonated user has a high level of privilege.

How to prevent agents who are not allowed from entering the system?

Solution

Create a single point of access to receive the interactions of a subject and apply a protocol to verify the identity of the subject. Create a proof of identity if subject is successfully authenticated.

Known Uses. Centralized authentication service in RADIUS.

Related Patterns. Single Access Point, Policy Enforcement Point.
Problem

In a client server system, there may be necessity to store data on the client. This data storage is necessitated for load-balancing, session management, single sign on etc. The client should not have access to view the data.

How can the data be protected from unauthorized access of the client?

Solution

Use encryption to protect the data that is stored on the client. Keep a hash value of the data to detect that the content is not tampered with. Use lightweight symmetric key to protect the data.
Change the session key often to protect against guessing attacks.

Known Uses. Amazon.com, Buy.com etc store encrypted cookie in the client.

Related Patterns. Encrypted Storage.
Application Architecture Pattern

Single Access Point

Problem
A security model is difficult to validate when there are multiple ways for entering the application.

How can we secure a system from outside intrusion?

Solution
Set up only one way to get into the system and if necessary, create a mechanism to decide which sub-application to launch. Typically most applications use a log in screen to accomplish the single access point.

Known Uses. Login window in Windows, Unix, MacOS.

Related Patterns. Policy Enforcement Point, Security Session.

Source: Wiley Book  
Tags: Access Control, Policy
Problem

A program memory processes can be limited by the memory used by the processes. If the processes grow unbounded, then there is a potential Denial of Service scenario. How can a program with many processes be made safe from resource exhaustion?

Solution

Make the processes small. Each process should perform one task. This will ensure that processes allocate limited memory.

Known Uses. qmail processes are very small and they only perform the required task.

Related Patterns. DoS Safety.

Source: Hafiz et. al.  Tags: Resource Limit, DoS.
References


