

## Entwurfsprinzipien und Entwurfsmuster für sichere Systeme

Heutzutage ist unumstritten, dass die Sicherheit beim Entwurf eines Systems eine sehr wichtige Rolle spielen soll. Trotzdem wird ihre Bedeutung oft unterschätzt und viele Anwendungen werden entwickelt, ohne dass gleichzeitig die nötigen Sicherheitsmaßnahmen entworfen und im Programmcode implementiert werden.

Dabei ist es von großer Bedeutung, dass die Sicherheit gleich zu Beginn in den Entwicklungsprozess einbezogen wird. Nur so kann man erreichen, dass das entworfene System am Ende einen klaren und übersichtlichen Aufbau hat. Wird dagegen an die Sicherheit erst am Ende des Entwurfs gedacht, so kann das verspätete Hinzufügen von Schutzmechanismen die klare Struktur der Anwendung zerstören. Die später eventuell benötigten Korrekturen sind in diesem Fall auch schwieriger durchzuführen.

Da die meisten Systementwickler keine Sachkundigen auf dem Gebiet der Sicherheit sind, brauchen sie Leitlinien, um beim Entwurf ihrer Anwendung richtige Sicherheitsentscheidungen treffen zu können. Solche Hilfen sollten grundsätzlich die Methodik erklären, wie man ein sicheres System entwickeln kann, und keine starren Fertiglösungen liefern, die wenig flexibel und daher nicht auf jede Situation übertragbar sind.

In diesem Artikel werden zuerst acht Entwurfsprinzipien aufgeführt, die 1975 von J. Saltzer und M. Schröder erarbeitet wurden. Sie helfen, die Sicherheitsverletzungen in Form von unerlaubter Informationsfreigabe, -modifikation oder -verweigerung zu vermeiden. Die Design-Prinzipien bieten wichtige Grundsätze für den Entwurf sicherer Systeme, sind aber eher abstrakt.

Auf konkreten Lösungen basieren dagegen einige Entwurfsmuster (Design Patterns) für sichere Applikationen, die anschliessend vorgestellt werden.

### Über die Autoren

Jan Jürjens ([juerjens@in.tum.de](mailto:juerjens@in.tum.de)) ist wiss. Mitarbeiter in der Informatik an der TU München, Autor des Buches Secure Systems Development with UML (geplant für 2003, Springer Verlag) und von über 20 Veröffentlichungen in inter- nationalen Zeitschriften und Konferenzen über Computersicherheit und Software-technik. Er ist Mitglied im Beirat des Kompetenz-zentrums Sicherheit in Bayern (KoSiB) und des AK e-Security der Bayer. Staatskanzlei, sowie Vorsitzender der FG für Formale Methoden und Software Engineering für sichere Systeme (FoMSESS) in der Gesellschaft für Informatik (GI) und Mitglied des Leitungsgremiums des FB Sicherheit der GI.

Lidiya Buda (LMU München) ist Diplom-Linguistin und jetzt auf dem Gebiet der Computerlinguistik tätig.

## 1 Entwurfsprinzipien zum Schutz von Informationen in Computersystemen

In diesem Kapitel wollen wir kurz auf die acht Prinzipien zur Entwicklung sicherer Systeme eingehen, die Saltzer und Schröder in ihrem Paper "The Protection of Information in Computer Science" (1975) vorgestellt haben.

### 1.1 Economy of Mechanism (*Einfachheit der Schutzmechanismen*)

Hier geht es darum, dass das Design eines Systems möglichst einfach gehalten sein sollte. Dieses Prinzip gilt generell für die Entwicklung der Anwendungssysteme, ist aber für die Implementierung der Schutzmechanismen von besonderer Bedeutung.

Die Design- und Implementierungsfehler, die Sicherheitslücken im System verursachen könnten, müssen rechtzeitig entdeckt werden. Dies bedarf einer sorgfältigen Überprüfung des Programmcodes, die um so leichter durchzuführen ist, je einfacher und durchsichtiger das System aufgebaut ist.

## **1.2 Fail-safe Defaults (Minimale Berechtigungsvergabe)**

Dieses Prinzip, das 1965 von E. Glaser vorgeschlagen wurde, fordert:

„Anfangs ist der Zugang zum System grundsätzlich untersagt und der Schutzmechanismus implementiert Konditionen, unter welchen der Zugang erlaubt werden kann.“

Bei der alternativen Vorgehensweise (also wenn man versuchen würde, Bedingungen festzulegen, unter denen der Zugriff verboten bleibt) handelt es sich laut Saltzer und Schröder um die "falsche psychologische Basis" für den Entwurf sicherer Systeme. Im Falle eines Design- oder Implementierungsfehlers wird beim Einhalten des „Fail-safe Defaults“-Prinzips höchstens einem zulässigen Benutzer der Zugriff zum System verweigert, während der alternative Ansatz ermöglichen würde, dass ein Unbefugter den Zugang zum System erhält.

## **1.3 Complete Mediation (Vollständige Berechtigungsüberprüfung)**

Bei jedem Zugriff auf ein Objekt soll eine Berechtigungsüberprüfung durchgeführt werden. Die systematische Anwendung dieses Prinzips bildet eine verlässliche Grundlage eines Sicherheitssystems. Das Complete Mediation Prinzip weist auf die Problematik der Integrität von Daten und deren Authentizität hin:

- Wieviel Vertrauen kann in die Ergebnisse bereits getätigter Berechtigungschecks und damit in deren Wiederverwendung gesetzt werden?
- Gibt es eine Garantie für die Authentizität der gestellten Anfrage?

Diese Fragen untermauern die Bedeutung dieses Prinzips.

## **1.4 Open Design (Offener Entwurf)**

Bei der Implementierung der Schutzmaßnahmen für das System sollte man sich nicht darauf verlassen, dass sein Aufbau den Angreifern unbekannt ist. Es ist unmöglich, das Design eines weit verbreiteten Systems geheim zu halten. Stattdessen sollte das System mittels kryptographischer Schlüssel bzw. Passwörter geschützt werden.

## **1.5 Separation of Privilege (4-Augen-Prinzip)**

Der Grundgedanke hierbei ist, dass ein Schutzmechanismus, der zwei Schlüssel für den Zugang voraussetzt, robuster und flexibler ist als einer, der nur mit einem Schlüssel abgesichert ist.

Die Bedeutung dieser Beobachtung für die Computersysteme wurde von R. Needham 1973 unterstrichen. Nach erfolgreicher Blockierung durch den Schutzmechanismus können die Schlüssel getrennt und unterschiedlichen Verantwortlichen übergeben werden. Ein Beispiel für die Anwendung dieses Prinzips sind Tresorfächer in Banken. Es wird aber auch in Computersystemen angewandt, und zwar in allen Situationen, wo mehr als eine Bedingung erfüllt sein muss, damit der Zugang erlaubt wird.

## **1.6 Least Privilege (Minimum an Rechten)**

Jedem Programm und jedem Benutzer sollen nur die Rechte zur Verfügung stehen, die zur Erfüllung seiner Aufgabe notwendig sind. Dieses Prinzip dient in erster Linie dazu, die infolge eines Fehlers oder Zufalls entstehenden Schäden zu begrenzen. Es reduziert auch die potentiellen Interaktionen zwischen zwei Programmen auf diejenigen, die beide Programme durchzuführen "berechtigt" sind.

Liegt ein Missbrauch der Rechte vor, so wird die Anzahl der zu überprüfenden Programme ebenfalls minimiert.

### **1.7 Least Common Mechanism (Minimum an gemeinsamen Mechanismen)**

Dieses Prinzip fordert, die Anzahl der Mechanismen, die mehrere Benutzer-gemeinsam verwenden, zu minimieren. Solche "geteilten" Mechanismen können indirekten Informationsfluß zwischen den einzelnen Benutzern ermöglichen und sollten deswegen höchst sorgfältig entworfen werden, so dass sie keine Sicherheitsgefährdung darstellen. Zudem müssten solche gemeinsam genutzten Systembereiche die Sicherheitsansprüche eines jeden Users befriedigen, was leichter ist, wenn es sich nur um einen oder wenige Benutzer handelt.

So sollte man zum Beispiel eine neue Funktion am besten als Library-Funktion zum System hinzufügen, so dass jeder User für sich entscheiden kann, ob er sie verwenden will. Entspricht die Funktion den Vorstellungen eines oder mehrerer Nutzer nicht, so sind diese frei, eine Alternative zur Verfügung zu stellen oder die entsprechende Funktion gar nicht zu benutzen.

### **1.8 Psychological Acceptability (Psychologische Akzeptanz)**

Wichtig ist, dass das System leicht zu benutzen ist. Vor allem sollte die Benutzeroberfläche möglichst einfach aufgebaut sein, was dazu beiträgt, dass die Sicherheitsmechanismen vom Benutzer routinemäßig und automatisch richtig angewandt werden. Zum anderen ist es wichtig, dass die zu benutzenden Schutzmechanismen den Vorstellungen des Users über die Systemsicherheit entsprechen. Ist dies der Fall, so wird die Anzahl der Fehler, die der Benutzer machen kann, minimiert.

## **2 Von Prinzipien zu Patterns**

Die im ersten Kapitel vorgestellten Design Prinzipien bilden wichtige Grundsätze für den Entwurf sicherer Systeme. Sie sind aber eher abstrakt und bieten alleine keine konkreten Anweisungen, die der Entwickler in seinem System direkt verwenden kann. Solche ausgearbeiteten Lösungen liegen den Design Patterns (Entwurfsmustern) zu Grunde.

Design Patterns wurden von Christopher Alexander 1977 folgendermaßen definiert:

"Jedes Muster beschreibt ein in unserer Umwelt beständig wiederkehrendes Problem und erläutert den Kern der Lösung für dieses Problem, so dass diese Lösung beliebig oft anwendbar ist, ohne jemals ein zweites Mal gleich ausgeführt zu werden." [DP]

Im Kontext der sicheren Systementwicklung spricht man von Security Patterns:

„Ein Security Pattern ist eine wohlverstandene Lösung für ein beständig wiederkehrendes Informationssicherheitsproblem. Sie sind Muster im von C. Alexander ursprünglich definierten Sinn, angewandt auf den Bereich der Informationssicherheit." [SPR]

Security Patterns basieren auf praktischer Erfahrung mit früher entwickelten Systemen und sind somit erprobt. Sie sind konstruktive Vorlagen, mit deren Hilfe jeder Entwickler sein System sicher gestalten kann, ohne sich vorher ein umfangreiches Sicherheits-Fachwissen aneignen zu müssen.

## 3 Design Patterns für sichere Applikationen

In diesem Abschnitt werden die Entwurfsmuster für sichere Systeme vorgestellt. Es werden für jedes Muster die Problemstellung und der entsprechende Lösungsvorschlag beschrieben, ein Zusammenhang zu den im zweiten Kapitel aufgeführten Design Prinzipien hergestellt, und anschließend eine kurze Bewertung sowie einige Anwendungsbeispiele gegeben.

### 3.1 *Single Access Point (Einzigiger Zugriffspunkt)*

#### 3.1.1 Problemstellung

Es ist sehr schwierig eine Anwendung zu schützen, die mit Netzwerken, Betriebssystemen, Datenbanken und anderen Infrastrukturen kommuniziert und somit eine Vielzahl an "Eingangstüren" hat.

#### 3.1.2 Lösung

Es soll genau eine Zugangsmöglichkeit für das System geben, also nur einen Ort, an dem sich der Benutzer einloggen kann. Der Grundgedanke dabei ist, dass der Single Access Point eine Einkapselung des Initialisierungsprozesses ermöglicht. Dadurch ist es leichter, die Korrektheit der durchgeführten Sicherheitsschritte zu validieren.

#### 3.1.3 Bezug zu den Design-Prinzipien

Es lässt sich hier die Beziehung zu folgenden Design Prinzipien herstellen:

- Economy of Mechanism - entwirft man ein System mit einem Single Access Point, so wird sein Programmcode wesentlich einfacher, als wenn man sich für mehrere Zugangsmöglichkeiten zum System entscheiden würde.
- Psychological Acceptability - da bei einem System mit Single Access Point die Berechtigungsüberprüfung genau einmal durchgeführt wird und sich an keiner anderen Stelle wiederholt, ist so ein System für den Benutzer einfacher zu bedienen und wird leichter akzeptiert.

#### 3.1.4 Bewertung

Bei Single Access Point startet die Anwendung von einer Stelle aus, an der alle wichtigen Initialwerte des Systems auf ihre Richtigkeit überprüft werden. Somit kann ein korrekter Startzustand erreicht werden. Zu den Nachteilen vom betrachteten Entwurfsmuster zählt, dass das System dadurch weniger flexibel wird. Es ist beispielsweise wesentlich einfacher, eine Anwendung, die mehrere Zugangsmöglichkeiten hat, in verschiedene Umgebungen zu integrieren.

Ausserdem werden in einem Single-Access-Point System bei der einzigen Berechtigungsüberprüfung möglicherweise viele Informationen abgefragt, die später gar nicht benötigt werden. Bei den Anwendungen mit mehreren Eingängen wird dagegen jeweils nur die aktuell benötigte Informationsmenge gefordert, was für die Benutzer weniger Aufwand bedeutet.

#### 3.1.5 Anwendungsbeispiele

Single Access Point findet man in den Betriebssystemen wie Windows oder Unix, bei denen der Zugang zum System über ein Login-Fenster gewährt wird. Es findet sich generell in jeder Anwendung, die nur auf einem Wege gestartet werden kann, womit der korrekte Startzustand gesichert wird. Dabei sind nicht alle solche Anwendungen sicher (bei Windows 95 kann das Login-Fenster z.B. leicht umgangen werden).

## 3.2 Check Point (Überprüfungspunkt)

### 3.2.1 Problemstellung

Ein System soll vor möglichen "Eingriffsversuchen" durch unberechtigte Personen geschützt werden. Es soll auch ein Weg gefunden werden, wie die vielfältigsten Sicherheitspolice in das System integriert werden können, da diese bei verschiedenen Organisationen, die das System benutzen, oft wesentliche Unterschiede aufweisen.

### 3.2.2 Lösung

Es soll ein Objekt geschaffen werden, in dem der Algorithmus für die Sicherheitspolice der jeweiligen Firma eingekapselt wird. Dieses Objekt überprüft, wie viele Ausnahmen auftraten und beschließt, von deren Schwere ausgehend, welche Maßnahmen zu ergreifen sind. Zu Beginn des Systementwurfs, wenn noch keine konkrete Sicherheitspolice vorliegt, kann der Entwickler ein Platzhalter-Objekt erzeugen, das jedem Benutzer den Zugriff gewährt. Das erleichtert späteres Einfügen der eigentlichen Sicherheitsüberprüfung an dieser Stelle.

In manchen Systemen werden bei diesem Entwurfsmuster zwei wichtige Bereiche unterschieden: die Authentisierung, also Überprüfung der Zugangsberechtigung des Benutzers, und die Autorisierung, bei der die Rechte des Users festgestellt werden.

Es gibt drei wichtige Faktoren, die beim Entwurf eines Check Point Algorithmus beachtet werden sollten:

1. Je nach Schweregrad der vorhandenen Sicherheitsverletzung können verschiedene Schutzmaßnahmen unternommen werden: von der Ausgabe einer Warnung bzw. Fehlermeldung an den User bei nicht-kritischen Ausnahmen, über Aufforderung zum Neustart, Abbruch des Login-Prozesses oder Verlassen des Programms, zur Sperrung der Maschine oder des Benutzers auf der höchsten Sicherheitsstufe.
2. Häufig wird die Schwere einer Sicherheitsverletzung danach gemessen, wie oft diese wiederholt wurde. Dabei sollte berücksichtigt werden, dass sich berechtigte Benutzer bei der Passworteingabe manchmal vertippen können. Die Häufigkeit solcher Fehler sollte daher protokolliert werden, um dann als Parameter in den Schutzmechanismus einzugehen.
3. Manche Sicherheitschecks können nicht sofort beim Betreten des Systems durchgeführt werden und müssen daher später vorgenommen werden.

### 3.2.3 Bezug zu den Design-Prinzipien

Die Idee des Check-Point Musters beruht auf folgenden Prinzipien:

- Economy of Mechanism – obwohl der Check Point Algorithmus das gesamte Sicherheitskonzept realisiert und dadurch eine ziemlich komplexe Struktur haben kann, werden hier jedoch alle notwendigen Sicherheitsprüfungen an einer Stelle konzentriert, was spätere Programm-Änderungen und Korrekturen erheblich erleichtert.
- Fail-safe Defaults und Complete Mediation - im Check Point muss der Autorisierungsalgorithmus so implementiert werden, dass er die Idee „permission rather than exclusion" umsetzt und damit dem „fail-safe Defaults" Prinzip folgt, und für „complete mediation" ist es nötig den Check Point Algorithmus mit den entsprechenden Tests auszustatten.

### 3.2.4 Bewertung

Zu den Vorteilen des Check Point-Patterns gehört die schon erwähnte Konzentration aller Sicherheitschecks an einer Stelle, was die späteren Modifizierungen vom Sicherheitsalgorithmus erleichtert. Dadurch wird der Check Point aber gleichzeitig zu einer kritischen Stelle im System und

muss deswegen sehr sorgfältig und besonders sicher entworfen werden.

### **3.2.5 Anwendungsbeispiele**

Login-Prozesse bei ftp-Servern oder Telnet sind nach dem Check Point Entwurfsmuster gestaltet.

## **3.3 Rollen**

### **3.3.1 Problemstellung**

Systembenutzer haben in der Regel verschiedene Sicherheitsprofile, die bei manchen von ihnen aber ähnlich sein können. Ist die Anzahl der Benutzer sehr groß oder die Hierarchie der Sicherheitsprofile zu komplex, kann die Handhabung dieser Berechtigungen für den Administrator erschwert werden. Es soll daher eine einfache Möglichkeit gefunden werden, um diese Privilegien zu regeln.

### **3.3.2 Lösung**

Es sollen ein oder mehrere Rollenobjekte geschaffen werden, welche die Rechte der einzelnen Benutzergruppen definieren. Dieses Muster führt eine Zwischeninstanz (die Rolle) ein. Die Benutzer-Rechte-Beziehung wird somit in die Benutzer-Rolle- und die Rolle-Recht-Beziehungen aufgeteilt. Zur Gestaltung der Beziehung Rolle-Recht gibt es zwei Möglichkeiten: entweder kennt jede Rolle ihre Rechte (diese Variante wird meistens bevorzugt), oder jedes Recht weiß, zu welchen Rollen es gehört.

Welche Rechte einer Rolle zugeordnet werden lässt sich nach verschiedenen Anhaltspunkten entscheiden: nach Abteilungszugehörigkeit der Benutzer, ihrer Erfahrung im Umgang mit dem System oder nach den Aufgaben, die sie zu erfüllen haben. Die Rollen können aber auch nach den Operationen (wie Editieren, Schreiben usw.) festgelegt werden.

### **3.3.3 Bezug zu den Design-Prinzipien**

Es lassen sich bei dem Rollen-Pattern folgende Prinzipien von Saltzer und Schröder erkennen:

- Economy of Mechanism - bei vielen Benutzern und zahlreichen Rechten kann die Zuweisungsliste, welche Benutzer-Rechte-Beziehungen wiedergibt, sehr umfangreich und unübersichtlich werden. Durch eine optimale Rollenwahl wird diese Liste aber wesentlich reduziert. Es ist auch viel einfacher, die notwendigen Korrekturen an einer Rollengruppe vorzunehmen, weil man so nur die Rechte der entsprechenden Rolle abzuändern braucht und nicht die von jedem einzelnen Benutzer.
- Fail-safe Defaults - durch eine Rolle werden dem Benutzer bestimmte Rechte gewährt. Es wird also nicht umgekehrt versucht festzulegen, welche Rechte ihm in dieser Rolle verweigert bleiben.
- Least Privilege - jeder Benutzer soll eine "angemessene" Rolle zugeschrieben bekommen, also eine solche, die ihm nicht mehr Rechte gewährt, als er für die Erfüllung seiner Aufgaben benötigt.

### **3.3.4 Bewertung**

Rollen sind eine bequeme Organisationsform für Administratoren. Sie bieten eine gute Möglichkeit, oft gemeinsam auftretende Rechte in eine Gruppe zu binden. Ausserdem lassen sich die möglichen Änderungen an den Rollengruppen einfacher realisieren.

Zu den Nachteilen dieses Patterns zählt, dass die Einführung von Rollen in das System die Arbeit des Entwicklers komplizierter macht. Ausserdem wird jeder Benutzer trotz der Einführung der Rollen seine

"private" Rolle brauchen, die seine eigenen Rechte und Vorteile unterstützt.

### 3.3.5 Anwendungsbeispiele

Rollenmuster wird in Datenbanken oft verwendet, um Benutzerrechte festzulegen (z.B. bei Oracle: read-only usw.).

Es ist ein sehr häufig auftretendes Muster, das überall dort eingesetzt wird, wo es vertrauliche Informationen gibt, die nicht von jedem Benutzer des Systems eingesehen werden dürfen und daher geschützt werden müssen.

## 4 Zusammenfassung

In diesem Artikel wurden in Kürze einige Hilfen für den sicheren Systementwurf vermittelt. Es wurden acht Prinzipien vorgestellt, die als Leitlinie für die Entwicklung sicherer Anwendungen dienen können. Anschließend wurden drei Entwurfsmuster aufgeführt. Diese bieten Hilfe für die Lösung einiger Sicherheitsfragen, die sich immer wieder bei dem Systementwurf stellen. Es gibt auch weitere Entwurfsmuster, die in diesem Artikel aus Platzgründen nicht vorgestellt werden konnten.

Da sich die Softwaresysteme ständig weiterentwickeln und dadurch immer mehr neue Gestaltungsmöglichkeiten für die Anwendungen entstehen, werden ständig neue Design Patterns geschaffen.

## 5 Literaturverzeichnis

[PoI] J. H. Saltzer, M. D. Schroeder: „The Protection of Information in Computer Systems“, University of Virginia, ©1975

[AP] J. Yoder, J. Barcalow: „Architectural Patterns for Enabling Application Security“ PLoP'97 © 1998

[SPR] D.M. Kienzle, M.C. Elder, D. Tyree, J. Edwards-Hewitt: „Security Patterns Repository Vs. 1.0“

[DP] E. Gamma, R. Helm, R. Johnson, J. Vlissides: „Design Patterns“ Addison-Wesley © 1995  
[www.securitypatterns.de](http://www.securitypatterns.de) (Dort finden sich viele Dokumente zum Thema Security Patterns)