



integrata
cegos

Java und Spring Security

Javacream

Einführung

- Java-Anwendungen sind wie alle anderen Software-Programme anfällig gegen typische Hacker-Angriffe:
 - Verarbeitung unvalidierter und damit potenziell gefährdender Benutzereingaben,
 - Ausführen von Code aus nicht vertrauenswürdigen Quellen,
 - Ausspionieren von Informationen durch Abhören unsicherer Kommunikationskanäle,
 - Unzulässige Ausführung von Programmteilen durch eine fehlerhafte Implementierung einer Authentifizierungs- und Autorisierungs-Routine.
- Die dadurch hervorgerufenen Schäden können selbst bei trivial scheinenden Lücken immens sein.
 - Es ist deshalb obligatorisch, zumindest die typischen Security-Lücken von vornherein auszuschließen und alle anderen durch ständige Qualitätssicherung zu erkennen und zu beheben.

- Die OWASP-Gruppe stellt jedes Jahr unter anderem eine Liste der „Top-10“ der Security-Probleme dar.



- Die Platzierung innerhalb der Liste wird durch eine Kombination aus Häufigkeit und Gefährlichkeit getroffen.

- SQL Injection ist ein Begriff, der wohl den meisten Entwicklern, die zumindest rudimentär auf das Thema Security sensibilisiert sind, bekannt sein dürfte.
- Die Security-Lücke besteht darin, dass unvalidierte Benutzereingaben zu einem SQL-Skript aufbereitet werden, so dass der Angreifer potenziell die vollständige Kontrolle über den gesamten Datenbestand erhält.
- ```
ResultSet resultSet = statement.executeQuery("select message from
messages where user = '"+ userCriterion + "'");
```
- Der rot dargestellte Teil zeigt die Security-Lücke durch das Hinzufügen des Kriterien-Ausdrucks zum SQL-Statement.

- Einige Attacken:
  - `String userCriterion = "User-0' or '1' = '1";`
  - `String userCriterion = "User-0' INSERT INTO MESSAGES VALUES ('A fake message', 'User-0') SELECT * from MESSAGES where user='User-0";`
  - `String userCriterion = "User-0' SELECT ADMINNAME from ADMINNS where 'h'='h";`

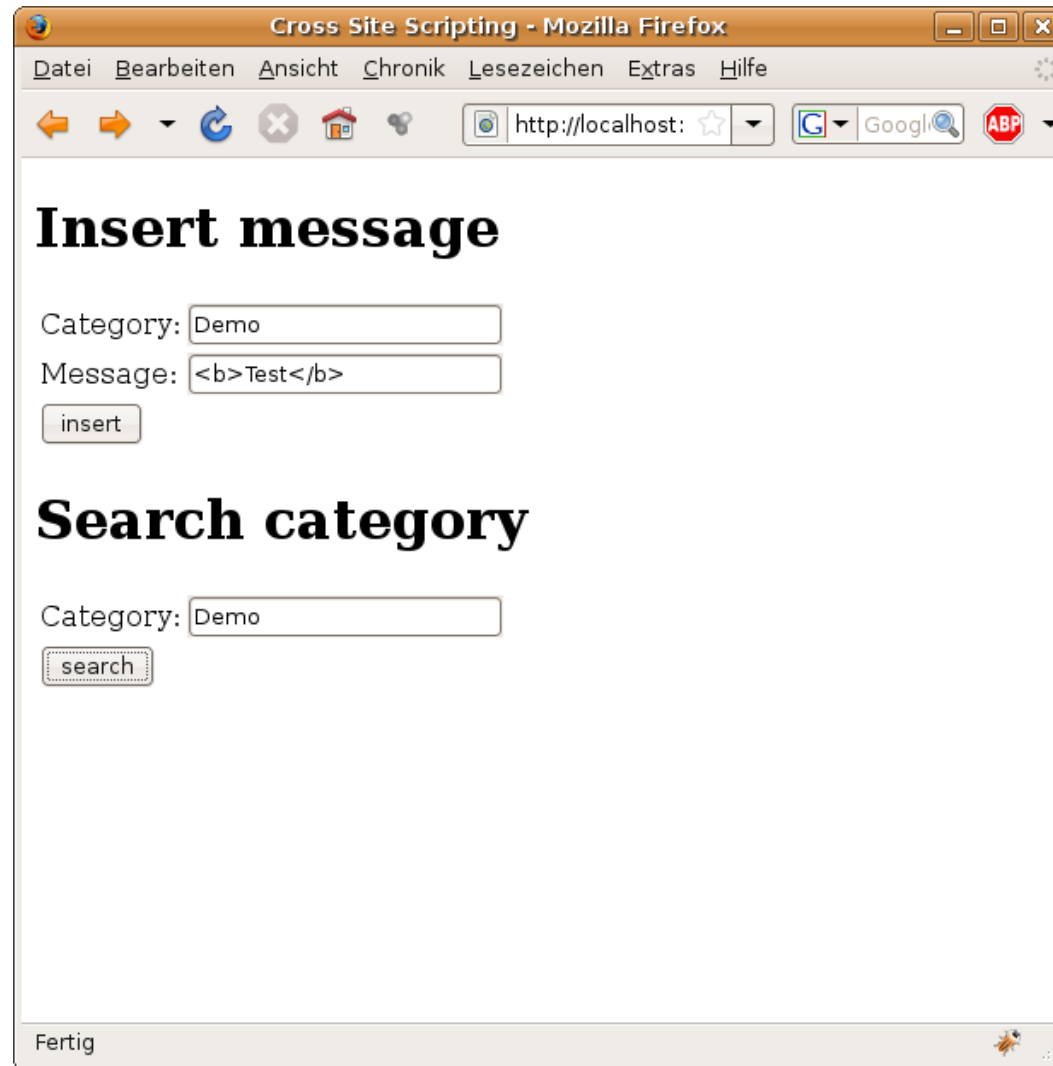
- Häufig wird unterschätzt, dass der Einsatz eines O/R-Mappers wie beispielsweise Hibernate oder dem Java Persistence API vor der SQL-Injection schützt.
- Dies ist aber nur beschränkt der Fall
- `Query query = entityManager.createQuery("from MessageHolder as message where message.user='" + userCriterion + "'");`
- Auch hier wird wie im Beispiel vorher im rot dargestellten Teil eine Benutzereingabe direkt zu einer Abfrage, diesmal in der JPA Query-Sprache, umgesetzt.
- Potenzielle Attacken:
  - `String userCriterion = "User-0' or 'h'='h";`
  - `String userCriterion = "User-0' AND (select count(*) from Admin) > 0 AND 'h' = 'h";`
  - `String userCriterion = "User-0' AND (select count(*) from Admin as admin where admin.adminName LIKE 'A%') > 0 AND 'h' = 'h";`

- Wird die Abfragelogik in Stored Procedures abgelegt, so wird gerne vergessen, dass damit eine Abfrage- oder Darstellungs-Logik nur in die Datenbank verschoben wird.
- Damit muss selbstverständlich auch die Parameter-Prüfung innerhalb der Stored Procedure erfolgen.
- Ist dies nicht der Fall sind auch Stored Procedures für SQL Injection-Attacken anfällig.



- Die aktuell laut der OWASP-Organisation empfindlichste Security-Lücke ist das Cross Site Scripting. Das Grund-Szenarium dafür lautet:
  - Eine Anwendung erlaubt es einem beliebigen (oder auch durchaus authentifiziertem!) Benutzer, Informationen zu hinterlegen.
  - Diese Informationen können von anderen Benutzern betrachtet bzw. allgemein verwendet werden.
  - Bei der Benutzung dieser Informationen werden Vorgänge ausgelöst, die für den Benutzer schädlich sein können.
- Das Paradebeispiel hierfür sind Web 2.0-Anwendungen, bei denen der Austausch beliebiger Informationen über Benutzer hinweg ja gerade das kennzeichnende Feature darstellt:
  - Chat-Rooms, Online-Foren oder Gästebücher sind typische Anwendungen.
- Es ist an dieser Stelle jedoch definitiv zu beachten, dass eine Beschränkung auf Web-Anwendungen das wahre Bedrohungspotenzial verschleiert.
  - Auch eine Administrationsoberfläche für eine Datenbank, die Datensätze darstellt, ist dafür anfällig.
  - Selbst ein simpler Editor für Log-Dateien kann theoretisch für Attacken benutzt werden.

# Cross Site Scripting: Eine anfällige Seite



Cross Site Scripting - Mozilla Firefox

Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe

http://localhost: Google ABP

## Insert message

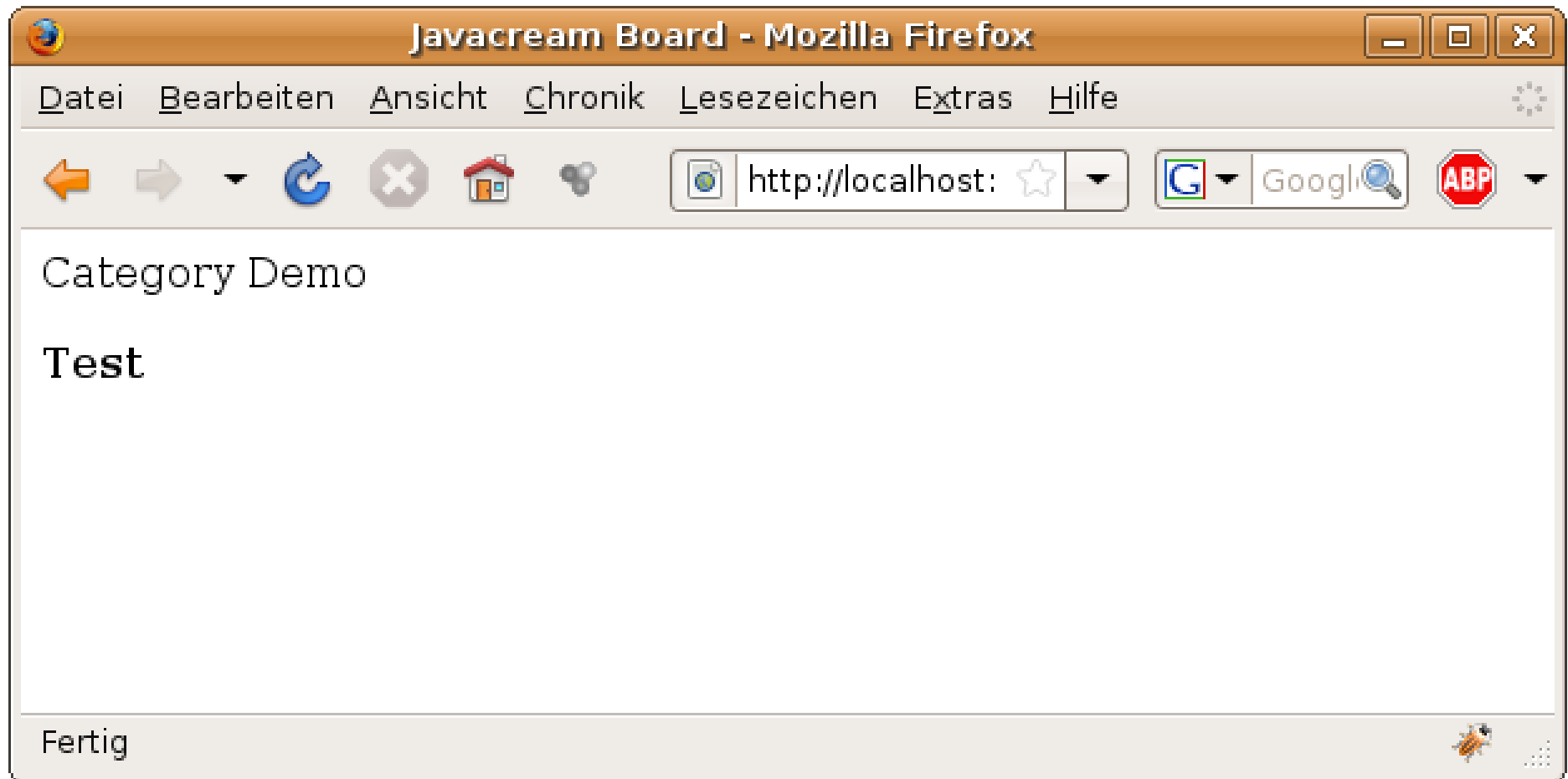
Category:

Message:

## Search category

Category:

Fertig



- Links auf unsichere Seiten
- Integration von Javascript
- Verlinken von Ressourcen (z.B. JPG-Bilder), die Fehler in Plug-Ins (z.B. das JPG-PlugIn im Internet Explorer) ausnutzen
- ...

- Diese Sicherheitslücke nutzt in seiner klassischen Form aus, dass in der URL eines Aufrufs eine Dateiangabe einer Ressource angegeben ist.
  - Der Angreifer kann diese Angabe manipulieren und auf diese Art und Weise den Zugriff auf kritische Informationen erhalten.
- Typische Beispiele sind:
  - Auslesen von Konfigurations- oder Log-Dateien.
  - Auslesen von uncompilierten jsp-Seiten.
  - ...
- In der modernen Welt der Java-Applikationen kann „Insecure Direct Object Reference“ anderes interpretiert werden, nämlich als „echte“ Referenz auf ein Business Objekt.
  - Frameworks wie Apache Struts oder auch Web Services Frameworks zeichnen sich unter anderem dadurch aus, dass rein konfiguratativ durch eine Deskriptor-Datei praktische jede Klasse ohne zusätzlichen Programmieraufwand für einen entfernten Methodenaufwurf freigegeben werden kann.

- Wo liegt das Problem?
- Ganz einfach:
  - Der von der Web Seite gesendete Request (hier ein GET-Request) ist für jeden einfach zu interpretieren.
  - So kann beispielsweise sofort statt des Methodennamens get ein remove ausprobiert werden.
  - Und schon kann der Client einen Eintrag löschen.
- Noch gefährlicher wird es dann, wenn auf Grund von Namenskonventionen der Angreifer andere Services erraten kann.
  - Wird diese Lücke nicht behoben ist damit der Zugriff auf die gesamte Domänenlogik möglich.
- Als Abhilfe dürfen nur die Methoden erreichbar sein, die wirklich nach Außen gegeben werden sollen.
  - Dies kann durch einen Servlet-Filter realisiert werden.

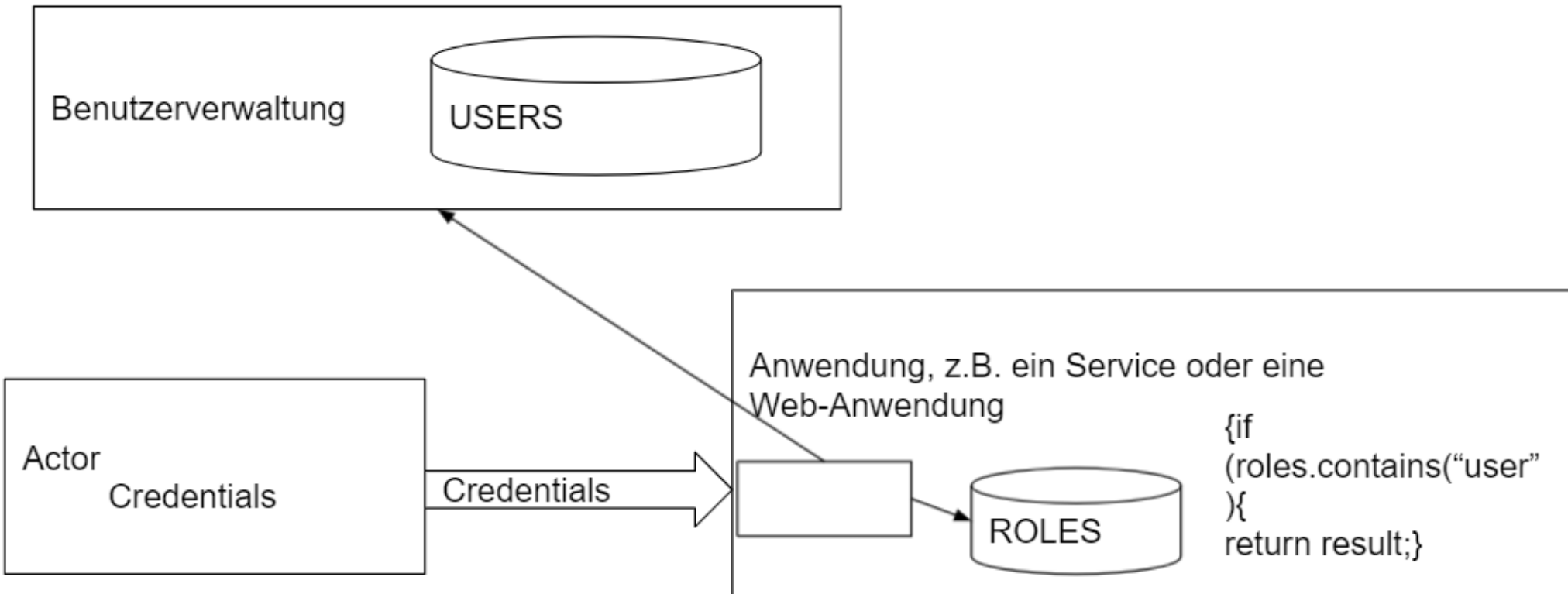
# AUSGANGSSITUATION

- Verschlüsselung der Kommunikation muss “anders” realisiert sein
  - https
  - (VPN)
- Klassische Firewall-Funktionen
  - z.B. Denial of Service
  - Port Scans
  - Injection
- Benutzerverwaltung mit Benutzername/Password, Credentials
  - Directory Server (LDAP)
  - Datenbank

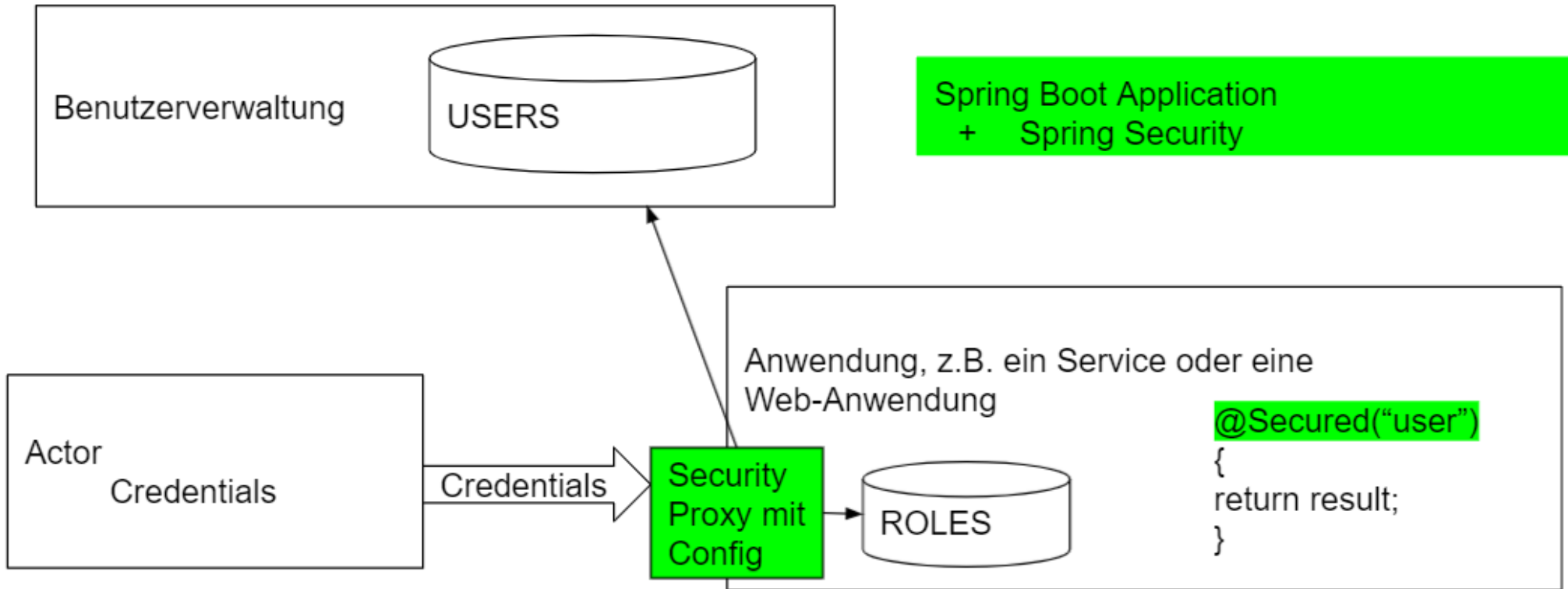


- Anwendung definiert Rollen, denen bestimmte Berechtigungen erteilt werden
  - Keine Querschnittsfunktion mehr!

- Definition der Anwender-Rollen
- Rollen müssen entweder statisch gespeichert oder dynamisch berechnet werden
  - Tabelle ROLES in einer Datenbank
  - (Berechnung durch eine Rules-Engine)
- Programmierlogik muss die konkreten Einschränkungen realisieren

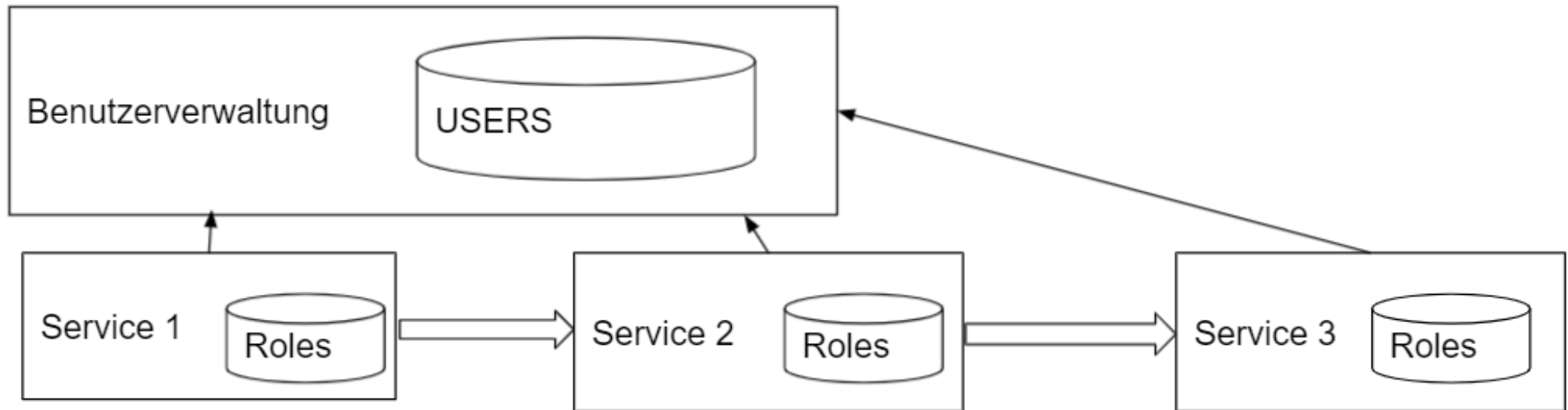


# Bestandteile einer aktuellen, etablierten Lösung



In der Realität:  
Service-Mesh



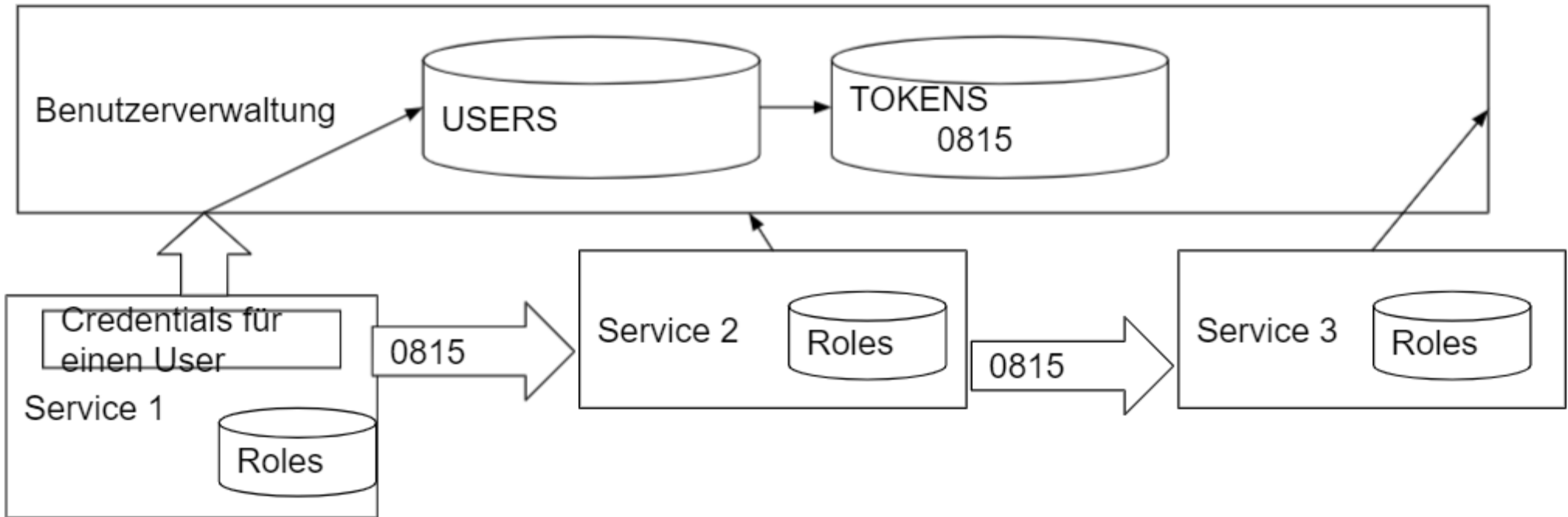


- Wer muss sich bei wem Authentifizieren?
- Eventuell Duplizierung von Rollen
- Anti-Pattern: "Credentials"

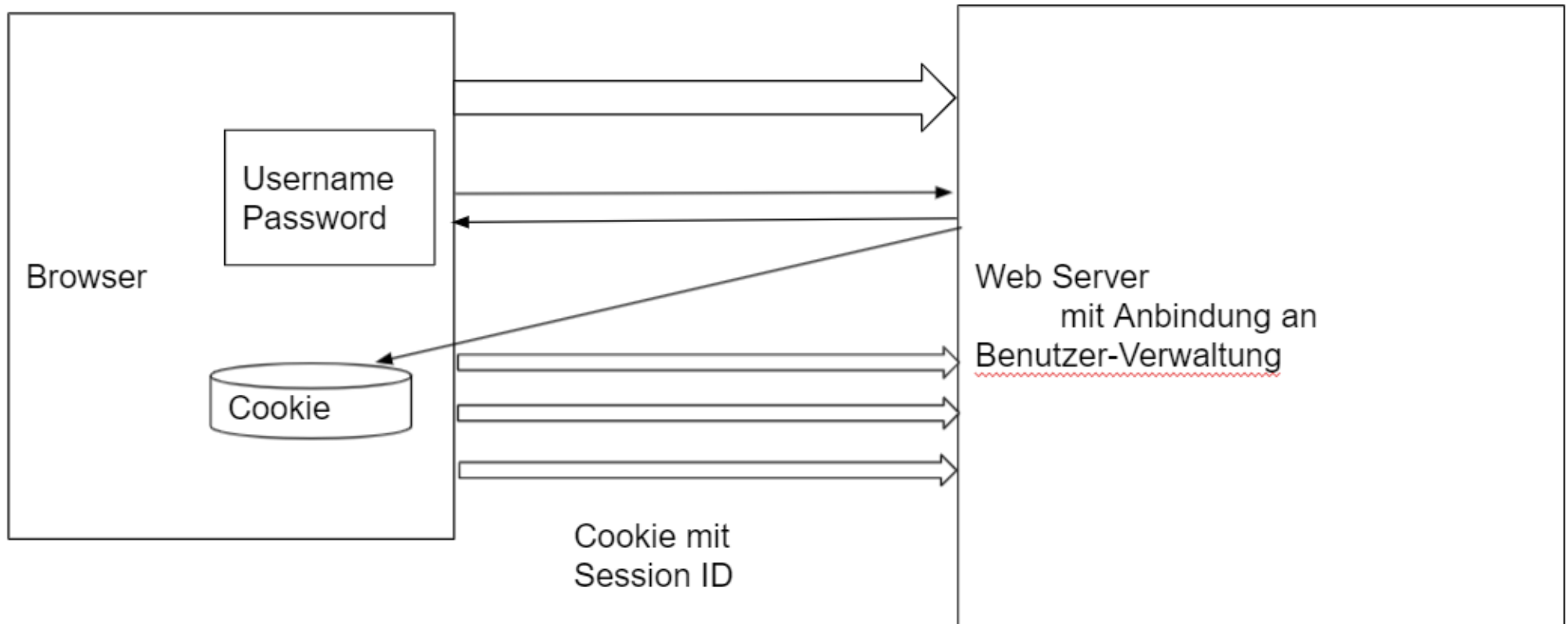
- Anti-Pattern “Credentials”
  - “Es ist zu vermeiden, User-Credentials auf verschiedene Services zu verteilen und permanent über Netzwerk zu übertragen”

- Tokens
  - Können risikolos ausgetauscht werden
  - Im Vergleich zu Credentials sind Tokens wesentlich kürzer gültig
  - Kategorien
    - Referenz-Tokens
      - Zufallswert
    - Value Tokens
      - Enthaltene Inhalte, die durch Standard-Verfahren garantiert unveränderbar sind
        - Mischung Hash und Signaturen

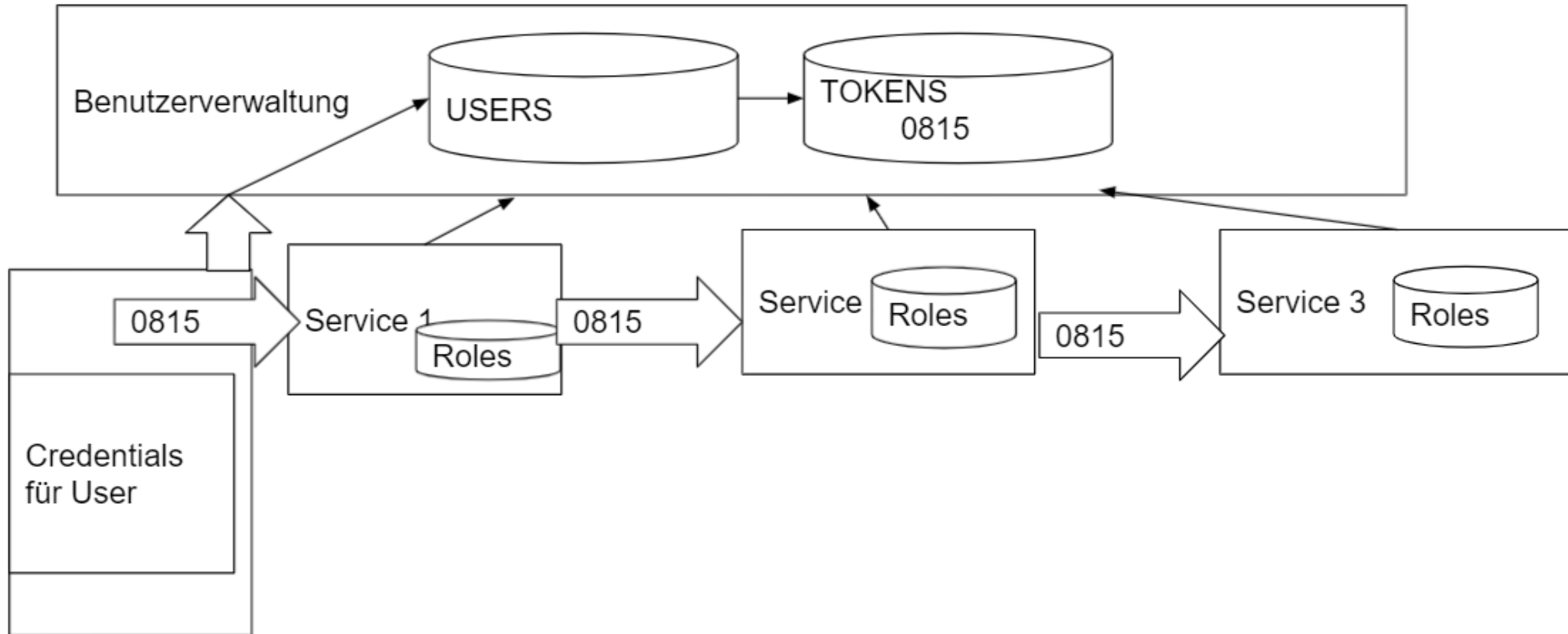




# Analogie: Tokens als Verallgemeinerung von Session-IDs einer Web Anwendung



# Service-orientierte Systemlandschaft mit API Gateway



# KOMMUNIKATION IM INTERNET

- Verfügbarkeit (availability)
- Integrität (integrity)
- Vertraulichkeit (confidentiality)
- Verbindlichkeit (liability)

- Generierung
- Zertifizierung
- Verwaltung
- Verifizierung



**Bundesamt für Sicherheit in der Informationstechnik**

**IT-Grundschutz-Kataloge**

Startseite IT-Grundschutz  
**Inhalt**  
■ Allgemeines  
■ Bausteine  
■ Gefährdungskataloge  
■ **Maßnahmenkataloge**  
  ▶ M 1 Infrastruktur  
  ▶ **M 2 Organisation**  
  ▶ M 3 Personal  
  ▶ M 4 Hardware und Software  
  ▶ M 5 Kommunikation  
  ▶ M 6 Notfallvorsorge  
■ Rollendefinitionen  
■ Glossar  
■ Index A-Z  
Baustein Datenschutz  
Hilfsmittel  
Überblickspapiere  
Bezugsquellen  
FAQ  
Registrierung / Newsletter  
Download  
Kontakt

Suche  Suchbegriff eingeben

**M 2.46 Geeignetes Schlüsselmanagement**

Verantwortlich für Initiierung: IT-Sicherheitsbeauftragter  
Verantwortlich für Umsetzung: Fachverantwortliche, IT-Sicherheitsbeauftragter

Die Verwendung kryptographischer Sicherheitsmechanismen (z. B. Verschlüsselung, digitale Signatur) setzt die vertrauliche, integre und authentische Erzeugung, Verteilung und Installation von geeigneten Schlüsseln voraus. Schlüssel, die Unbefugten zur Kenntnis gelangt sind, bei der Verteilung verfälscht worden sind oder gar aus unkontrollierter Quelle stammen (dies gilt auch für die Schlüsselvereinbarung zwischen Kommunikationspartnern), können den kryptographischen Sicherheitsmechanismus genauso kompromittieren wie qualitativ schlechte Schlüssel, die auf ungeeignete Weise erzeugt worden sind. Qualitativ gute Schlüssel werden in der Regel unter Verwendung geeigneter Schlüsselgeneratoren erzeugt (siehe unten). Für das Schlüsselmanagement sind folgende Punkte zu beachten:

### Schlüsselerzeugung

Die Schlüsselerzeugung sollte in sicherer Umgebung und unter Einsatz geeigneter Schlüsselgeneratoren erfolgen. Kryptographische Schlüssel können zum einen direkt am Einsatzort (und dann meistens durch den Benutzer initiiert) oder zum anderen zentral erzeugt werden. Bei der Erzeugung vor Ort müssen meistens Abstriche an die Sicherheit der Umgebung gemacht werden, bei einer zentralen Schlüsselgenerierung muss sichergestellt sein, dass sie ihre Besitzer authentisch und kompromittierungsfrei erreichen.

Geeignete Schlüsselgeneratoren müssen kontrollierte, statistisch gleichverteilte Zufallsfolgen unter Ausnutzung des gesamten möglichen Schlüsselraums produzieren. Dazu erzeugt z. B. eine Rauschquelle zufällige Bitfolgen, die mit Hilfe einer Logik nachbereitet werden. Anschließend wird unter Verwendung verschiedener Testverfahren die Güte der so gewonnenen Schlüssel überprüft.

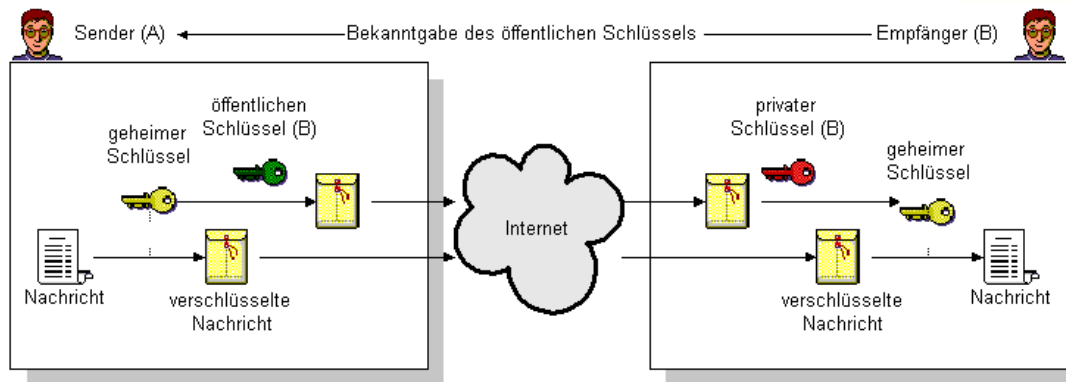
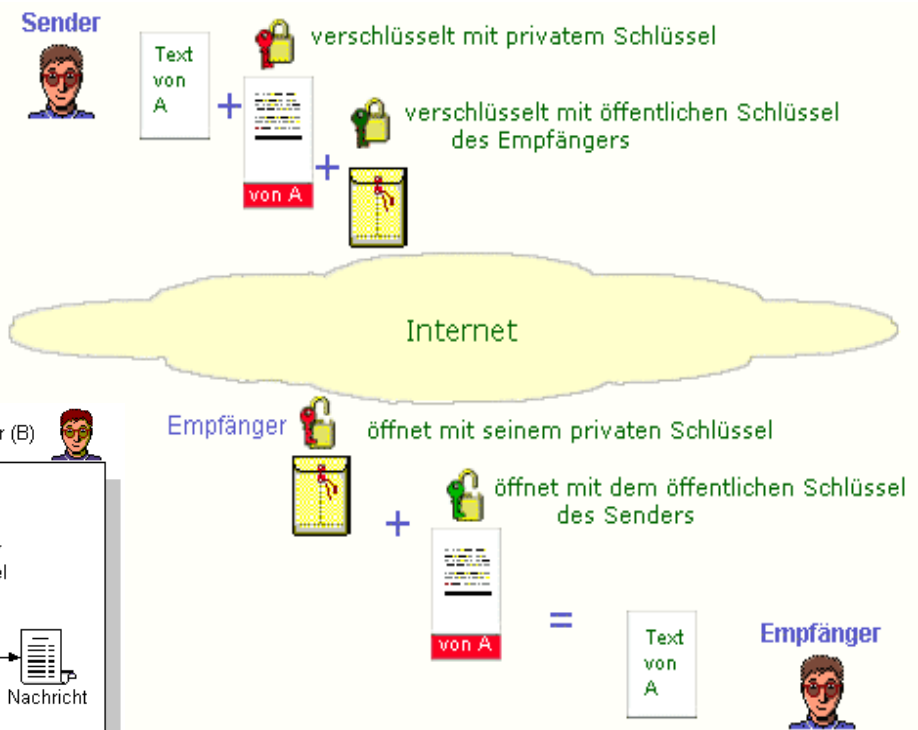
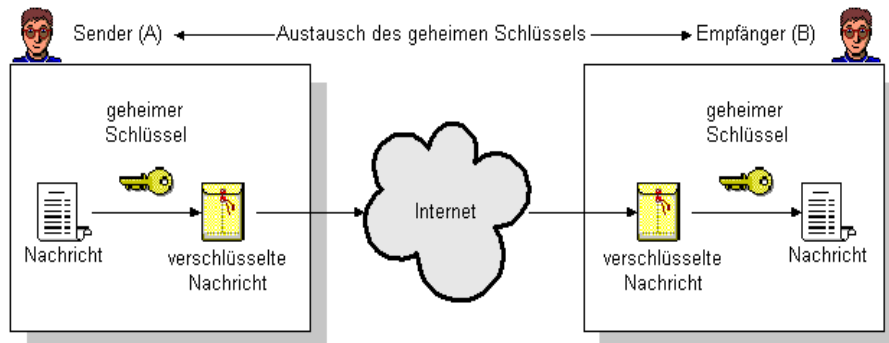
Einige Kryptomodule, insbesondere solche, die keinen integrierten Zufallszahlengenerator besitzen, greifen auf Benutzereingaben zur Schlüsselerzeugung zurück. Beispielsweise werden hier Passwörter abgefragt, aus denen dann ein Schlüssel abgeleitet wird, oder der Benutzer wird gebeten, beliebigen Text einzutippen, um zufällige Startwerte für die Schlüsselgenerierung zu erhalten. Solche Passwörter sollten dabei gut gewählt sein und möglichst lang sein. Wenn möglichst "zufällige" Benutzereingaben angefordert werden, sollten diese auch zufällig, also schlecht vorhersagbar, sein.

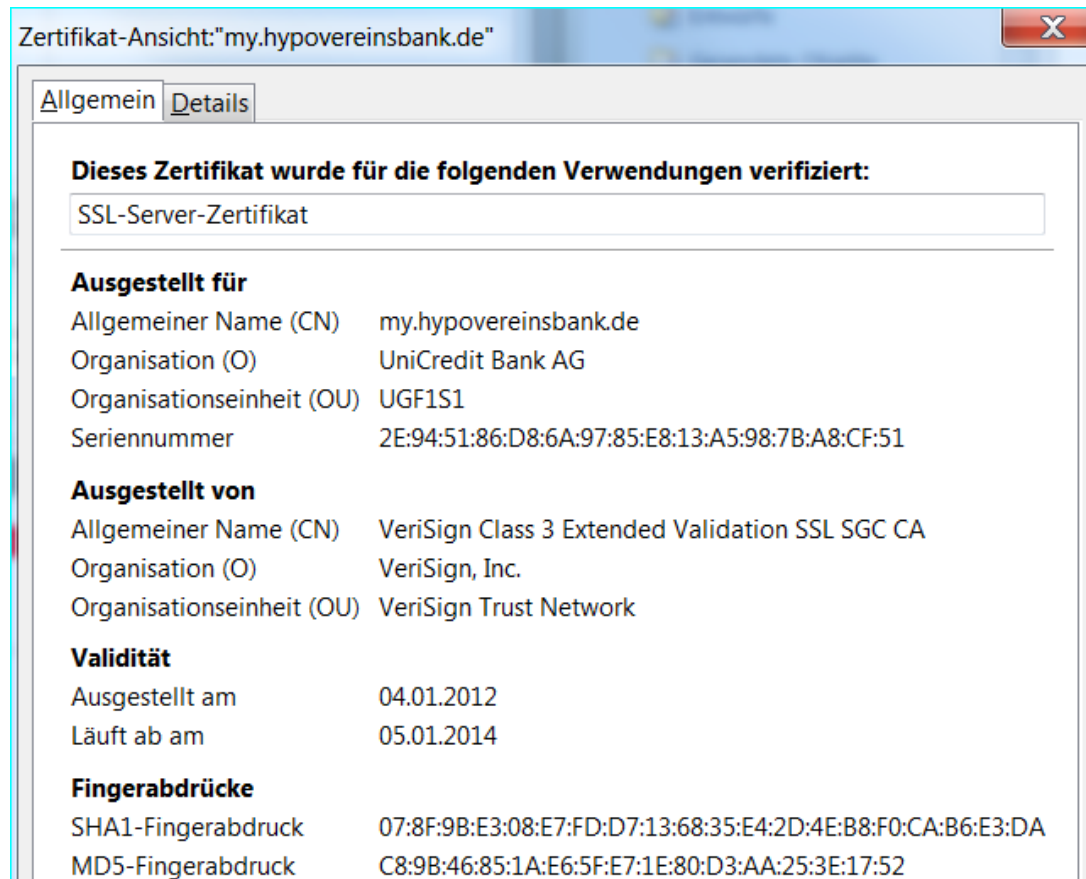
### Schlüsseltrennung

Kryptographische Schlüssel sollten möglichst nur für einen Einsatzzweck dienen. Insbesondere sollten für die Verschlüsselung immer andere Schlüssel als für die Signaturbildung benutzt werden. Dies ist sinnvoll,

- damit bei der Offenlegung eines Schlüssels nicht alle Verfahren betroffen sind,
- da es manchmal erforderlich sein kann, Verschlüsselungsschlüssel weiterzugeben (Vertretungsfall),
- da es unterschiedliche Zyklen für den Schlüsselwechsel geben kann.

### Schlüsselverteilung / Schlüsselaustausch










Jetzt von  
**Symantec**

VeriSign  
Authentifizierungsdienste

Deutschland [ändern] | Kontaktieren Sie uns



powered by VeriSign

Produkte & Services | Partner | Support | Mein Konto



powered by VeriSign

## Dasselbe Häkchen. Ein neuer Name. Derselbe leistungsstarke Schutz.

Dieselbe Qualität bei Sicherheit, Services und Support, der Sie bei VeriSign vertrauen, wird Ihnen nun von Symantec geboten.

Die Konsequenzen für Sie ➤

- 1
- 2
- 3
- 4

**KAUFEN** SSL-Zertifikate


**KAUFEN** Symantec™ Safe Site

**KAUFEN** Code-Signing

**TESTEN** Kostenlose Demoversion

**ERNEUERN** SSL-Zertifikate erneuern

**ANMELDEN** Symantec™ Trust Center



powered by VeriSign

Norton™ Secured-Siegel

### Verwalten Sie mehrere SSL-Zertifikate?

Eine einfache Management-Lösung.



Weitere Informationen ➤

### Schützen Sie Ihre Website. Steigern Sie Ihren Umsatz.

Neue Funktionen in Symantec SSL machen es einfach, Ihrer Website **zu vertrauen**.



Weitere Informationen ➤



**VERISIGN™**

Für Online-Sicherheit und Verfügbarkeit verlässt sich Ihr Unternehmen auf:

- Managed DNS
- DDoS Protection
- iDefense™
- Domain Name Services

Diese Services sind bei VeriSign Inc unter [VeriSignInc.com](http://VeriSignInc.com) erhältlich.

Kontaktieren Sie uns | Über Symantec | Neuigkeiten | Blogs | Rechtliche Hinweise | Datenschutz | Repositorium | Globale Sites | Sitemap

# JAAS

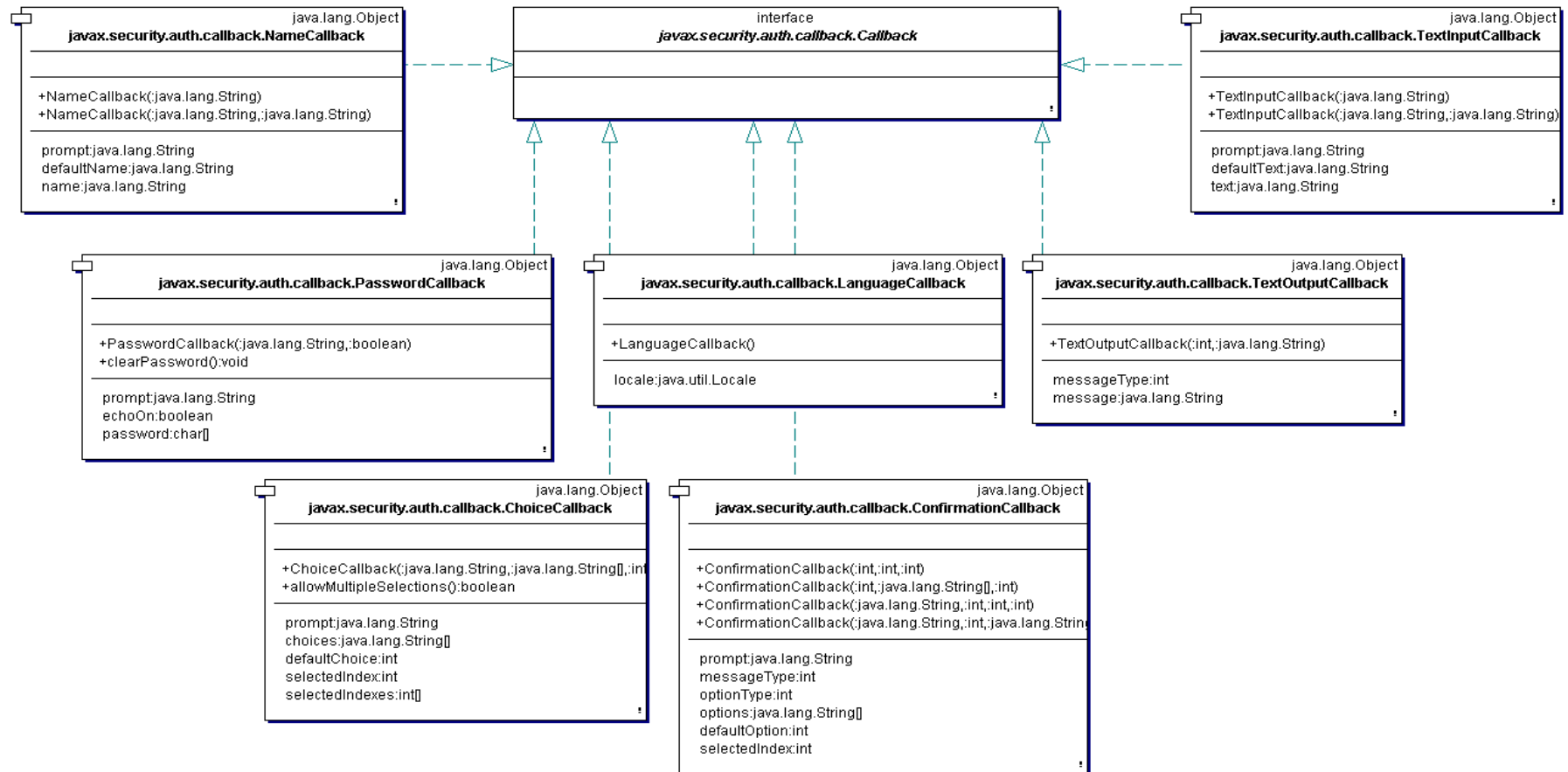
- Das Realm definiert einen eigenen Namensraum für Benutzer und Gruppen. Ein Realm wird persistent verwaltet.
- Ein einzelner Benutzer kann keiner, einer oder mehrerer Gruppen zugeordnet sein.
- Benutzer und Gruppen können z.B. in Tabellen abgelegt werden. Jede Gruppe besteht aus keinem, einem oder mehreren Benutzern. Eine Rolle ist eine Menge aus Gruppen und Benutzern. Rollen sind anwendungsabhängig.

- Prinzipal
  - Ein Name, eine Personalnummer, ein Benutzername
  - Interface `java.security.Principal`
- Credential
  - Ein Credential ist kein Java-Interface sondern irgendeine Klasse, die zur Authorisierung benutzt werden kann
  - Je nach dem verwendeten Authentifizierungsmechanismus kann dies ein einfaches Byte-Array als Password oder ein Zertifikat sein.

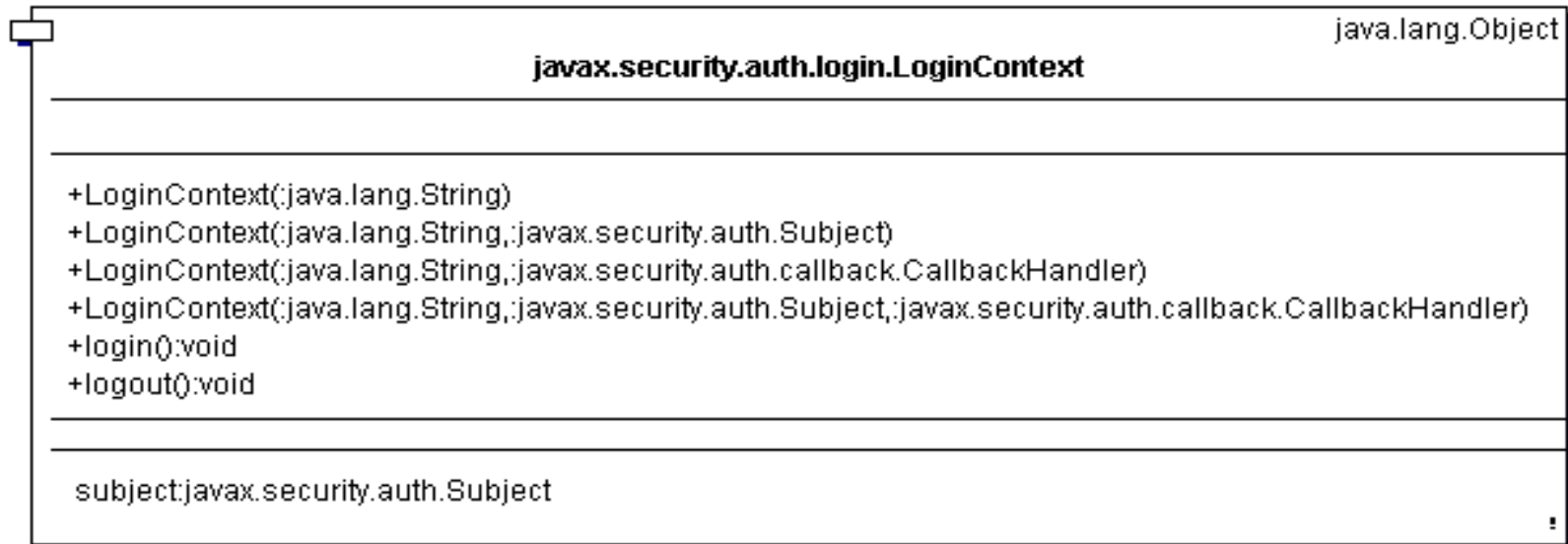
- Subject erfüllt drei Aufgaben:
- Halten von einem oder mehreren Principal-Referenzen.
- Halten von Credentials.
- Ablegen eines konkreten Subjects innerhalb des aufrufenden Threads. Dies erfolgt durch den Aufruf einer der statischen Methoden `doAs()` oder `doAsPrivileged()`.
  - Nach diesem Aufruf kann das Subject als Bestandteil des `AccessContext` ausgelesen werden.

|                                                                           |
|---------------------------------------------------------------------------|
| interface<br><i><b>javax.security.auth.callback.CallbackHandler</b></i>   |
|                                                                           |
| <i>+ handle(<b>javax.security.auth.callback.Callback[]</b>):void</i><br>! |

- `javax.security.callback.Callback` ist ein reines Marker-Interface
- Callback-Implementierungen dienen zum Sammeln und Transport von Informationen, die zur Authentifizierung benötigt werden.
- Der Authentifizierungs-Mechanismus ruft für den `CallbackHandler` die Methode `handleCallback(Callback)` mit geeigneten Callbacks auf.
- Der `CallbackHandler` prüft den Typ des übergebenen Callbacks und setzt die geforderten Parameter







|                                                                                                                                                                                                                                                |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| interface<br><b><i>javax.security.auth.spi.LoginModule</i></b>                                                                                                                                                                                 |
|                                                                                                                                                                                                                                                |
| <i>+initialize(javax.security.auth.Subject, javax.security.auth.callback.CallbackHandler, java.util.Map, java.util.Map):void</i><br><i>+login():boolean</i><br><i>+commit():boolean</i><br><i>+abort():boolean</i><br><i>+logout():boolean</i> |
| !                                                                                                                                                                                                                                              |

- Der initialize-Methode werden alle benötigten Referenzen übergeben:
  - Das zu ergänzende Subject.
  - Der CallbackHandler des Aufrufenden.
  - Eine Map mit dem „Shared-State“, der allen an der Authentifizierung beteiligten LoginModules gemeinsam ist.
  - Eine Map mit den Konfigurationsparametern.

- Der Authentifizierungs-Teil des JAAS wird durch einen Provider zur Verfügung gestellt:
- LoginModule,
- Principal,
- Credential-Klassen,
- Spezielle Callback-Klassen,
- Callback-Handler

- Für den Client beschränkt sich der JAAS-Provider auf eine Hilfsklasse, die eine Konfigurations-Datei ausliest
  - `javax.security.auth.login.Configuration`
- Eintrag in der Konfigurationsdatei der Java-Laufzeitumgebung für den Default-Provider Sun (
  - `<JAVA_HOME>\jre\lib\security\java.security`
  - `login.configuration.provider=com.sun.security.auth.login.ConfigFile`

- Suchen einer Konfigurationsdatei an Hand eines Eintrags in der JRE-Konfigurationsdatei:
  - `login.config.url.1=file:${user.home}/.java.login.config`
- Auslesen einer Umgebungsvariable
  - `java.security.auth.login.config`

# Syntax der Konfigurationsdatei: Authentifizierungs-Namen

```
SimpleLogin{

};
SSLLogin{

};
default{

};
```

```
SimpleLogin{
 com.hotspots.javax.security.login.spi.SimpleLoginModule;
};
SSLLogin{
 com.hotspots.javax.security.login.spi.SSLLoginModule;
 com.hotspots.javax.security.login.spi.CertificateLoginModule;
};
default{
 com.hotspots.javax.security.login.spi.SimpleLoginModule;
};
```



```
SimpleLogin{
 com.hotspots.javax.security.login.spi.SimpleLoginModule debug=true;
};
SSLLogin{
 com.hotspots.javax.security.login.spi.SSLLoginModule debug=true;
 com.hotspots.javax.security.login.spi.CertificateLoginModule debug=true
 url=ldap://hotspots.com;
};
default{
 com.hotspots.javax.security.login.spi.SimpleLoginModule;
};
```

Diese Parameter werden als Map der initialize-Methode des LoginModule übergeben.

```
SimpleLogin{
 com.hotspots.javax.security.login.spi.SimpleLoginModule debug=true;
};
SSLLogin{
 com.hotspots.javax.security.login.spi.SSLLoginModule requisite debug=true;
 com.hotspots.javax.security.login.spi.CertificateLoginModule required debug=true
 url=ldap://hotspots.com;
};
default{
 com.hotspots.javax.security.login.spi.SimpleLoginModule;
};
```

|            |                                                                                       |
|------------|---------------------------------------------------------------------------------------|
| required   | Erfolgreicher Login notwendig, die weiteren Module werden in jedem Fall abgearbeitet. |
| requisite  | Falls der Login nicht erfolgreich ist, wird sofort abgebrochen.                       |
| sufficient | Falls der Login erfolgreich ist, werden keine weiteren Module mehr abgearbeitet.      |
| optional   | Das LoginModule kann erfolgreich sein oder auch nicht.                                |

# Der LoginContext ist eine Fassade zur Erzeugung eines Subjects

