# Unleashing Java Security


Philipp Buchholz

# Inhalt

- **Basic Security Model in Java**
  - Grundlagen
  - PermissionCalc
  - Privileged Block API
  - Administration
  - Extension-Points
- **Extended Security Model**
  - JAAS
- Java EE und JASPIC

# Basic Security Model in Java

# Basic Security Model | Grundlagen

- Basiert auf Herkunft des Codes **(CodeSource based)**
  - Erteilen von Berechtigungen auf Basis der Herkunft
    - **URL** (CodeBase)
    - **Signer** (Alias im Keystore)
- Bekannt als „Java Sandbox“
  - Muss aktiviert werden **(SecurityManager)**
- Policy
  - Sicherheitseinstellung einer Anwendung

# Basic Security Model | CodeSource

- **CodeSource**
  - Herkunft von Code
    - Location (URL)
    - Signer (Alias eines Zertifikates im Keystore)

# Basic Security Model | ProtectionDomain

- **ProtectionDomain (PD)**
  - {Class A, Class B, Class C, … }
  - {Permission A, Permission B, Permission C, …

# Basic Security Model | ProtectionDomain

- Pro JVM immer mindestens zwei ProtectionDomains
  - **System-PD**
    - JDK
    - Implizit `AllPermission` (alle Berechtigungen)
  - **Application-PD**
    - Code der Applikation
    - Zugeteilte Berechtigungen

# Basic Security Model | Permission

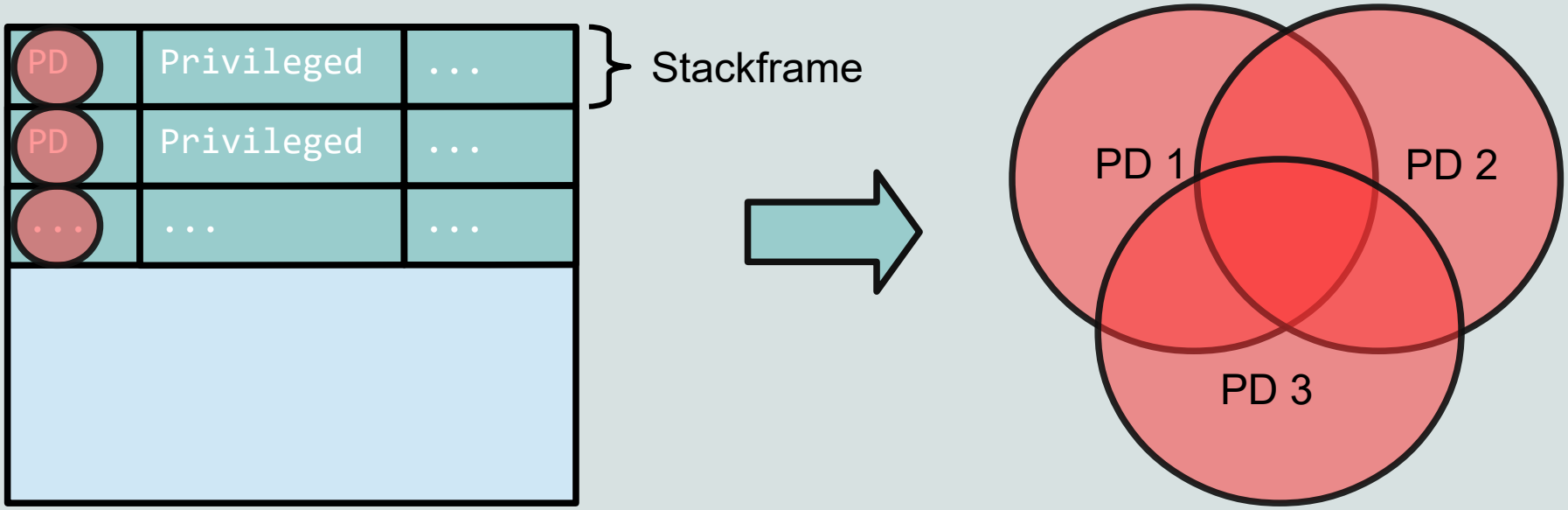
- Stellt Zugriff auf geschützte Ressourcen dar.
- Besteht aus drei Teilen
  - Type, Name (Optional), Action(s) (Optional)
- `java.security.Permission`
- **Beispiele**
  - Zugriff auf lokales Dateisystem
    - `java.io.FilePermission`
  - Zugriff auf das Netzwerk
    - `java.net.SocketPermission`

# Basic Security Model | Custom Permission

- `implements java.security.Permission`
  - Komplexe Permission mit Name und Action(s).
- `extends java.security.BasicPermission`
  - Nur Name
    - Wildcards möglich
      - Hierarchical Property Naming Convention
        - „*“, „propertyname.*“

# Basic Security Model | PermissionCalc

- Intersect der Permissions aller aktiven ProtectionDomains



# Basic Security Model | SecurityManager

- Bietet für bekannte Permissions Methoden zur Überprüfung
  - Rechte auf lokales Filesystem? `boolean checkWrite(...), boolean checkWrite(...), …`
- Delegiert `checkXXX(...)` Aufrufe an `AccessController.checkPermission(...)`
- `java.lang.SecurityManager`
- Legacy

# Basic Security Model | SecurityManager

- Für Berechtigungsprüfungen muss SecurityManager installiert sein.
  - Kommandozeile
    - `-Djava.security.manager`
  - Programmatisch
    - `System.setSecurityManager(new SecurityManager());`

# Basic Security Model | AccessController

- AccessController
  - Default SecurityManager
  - `checkPermission(xxx)`
    - Ermittelt AccessControlContext des aktuellen Aufrufkontexts (Stack)
      `native getStackAccessControlContext()`
    - Delegiert an
      `AccessControlContext.checkPermission(...)`
      - Prüft angeforderte Berechtigung gegen Policy

# Basic Security Model | Ablauf

# Basic Security Model | Privileged Block API

- `<T> AccessController.doPrivileged(PrivilegedAction<T> action) : T`
  - **Privileged Action** = Aktion die mit mehr Rechten als im aktuellen Kontext erteilt wird ausgeführt wird.
    - Umgehung PermissionDomain-Intersect
  - **Möglich wenn**
    - privilegierter Code berechtigt ist **und**
    - in Folge aufgerufener Code berechtigt ist.
  - **Implementierung von**
    - `java.security.PrivilegedAction` oder
    - `java.security.PrivilegedExceptionAction`

# Basic Security Model | Privileged Block API

- Erteilte Berechtigungen in Zielumgebung nicht klar.
  - Libraries und Frameworks

```
AccessController.doPrivileged(new PrivilegedAction<String>() {
          public String run() {
               Configuration configuration = new Configuration();
               return configuration.read("backendurl");
          }
     });
```

# Basic Security Model | PolicyFiles

- Administration von ProtectionDomains
  - Welcher Code (CodeSource) erhält welche Permissions
  - grant-Entries
- Default Policy-Files
  - `.java.security` configuration file
    - global Policy-File
    - user Policy-File
    - policy.url.1=file:...
- Application specific Policy-Files
  - `-Djava.security.policy=policyfile`

# Basic Security Model | PolicyFiles

```
grant codeBase "file:${{java.ext.dirs}}/*" {
        permission java.security.AllPermission;
};


grant codeBase „file:...jar“  {
        permission java.net.SocketPermission "192.168.2.1:8080", "listen";

        permission java.io.FilePermission "${user.home}${/}*", „read, write“
        permission java.io.FilePermission "file:C:/...", "read";

}

grant ... {
...
}
```

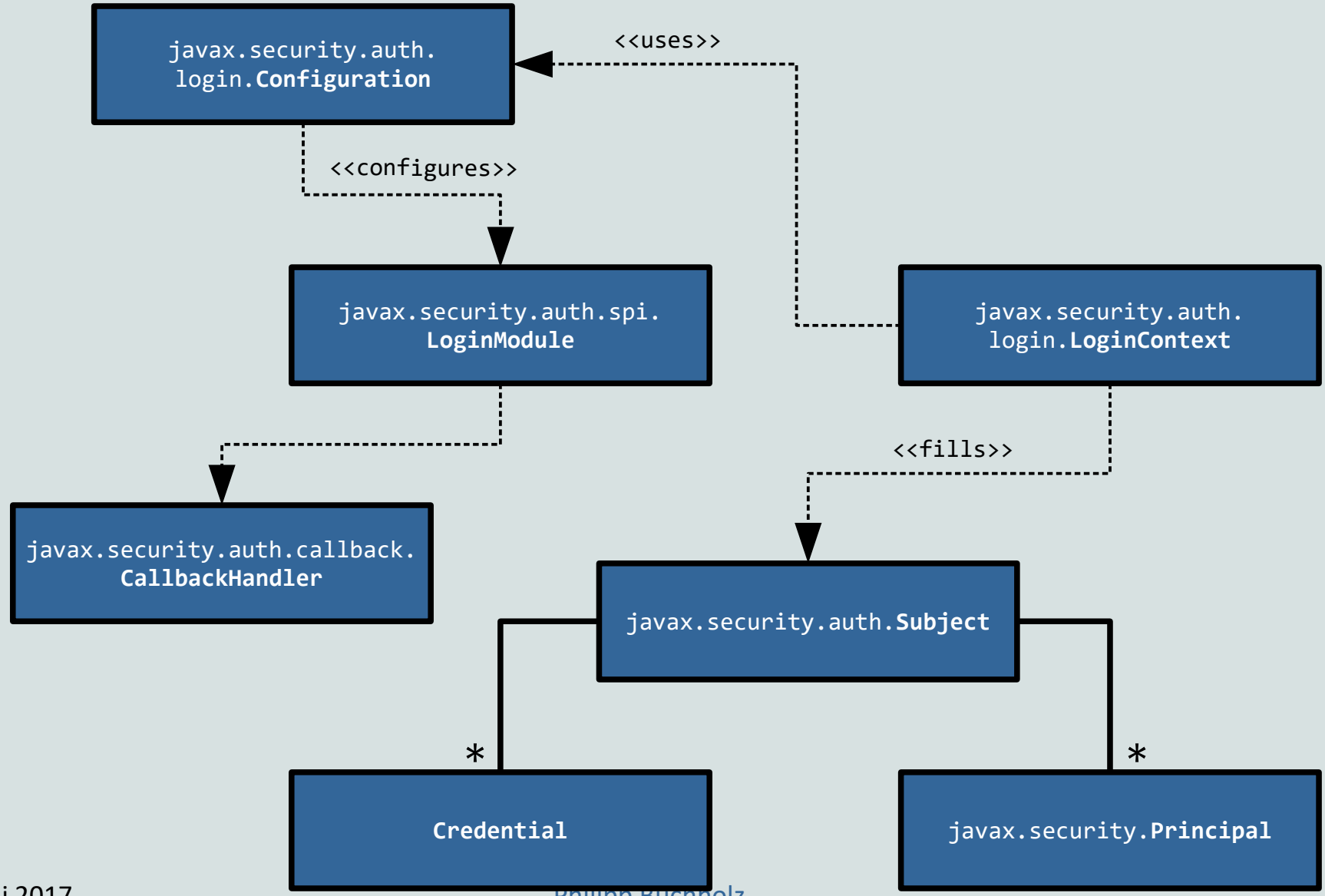
ProtectionDomain

# Extended Security Model

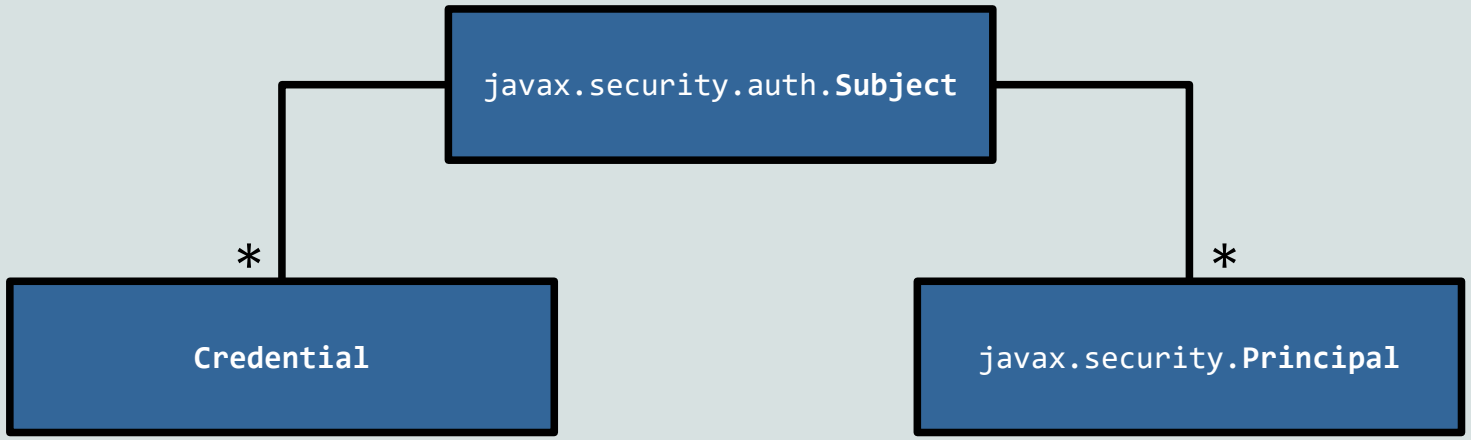
# Extended Security Model | JAAS

- Erweiterung des Basic Security Model (**CodeSource based**)
  - Integriert in JDK seit Version 1.4
  - Authentifizierung und Autorisierung von Benutzern oder anderen Subjects (**Subject based**)
  - **PAM** (**P**luggable **A**uthentication **M**odule)
    - Authentication Mechanismen ohne Anwendungsänderung hinzufügbar (**pluggable**)

# Extended Security Model | JAAS



# Extended Security Model | JAAS




- **Subject**
  - Authentifizierbare Entität bzw. Quelle eines Requests
  - Wird beim Login (Authentifizierung) erstellt.
- **Principal**
  - Einem Subject zugeordnete Identitäten
    - Name, Ausweisnummer, Zertifikate, …
  - Wird nach erfolgreicher Authentifizierung durch LoginModule erstellt

# JAAS | Credential

- **Credential**
  - Sicherheitsrelevante Attribute
  - Öffentlich
    - z.b. public keys
  - Private
    - z.b. private keys

# JAAS | LoginModule

- Implementiert Authentifizierungsvorgang
  - Abfrage Authentifizierungsinformationen
    - **CallbackHandler** und **Callbacks**
      - `javax.security.auth.callback.NameCallback`
        - Abfrage Username
      - `javax.security.auth.callback.PasswordCallback`
        - Abfrage Passwort
      - Benutzerdefinierte Callbacks
- LoginContext **1 : N** LoginModule
- JAAS LoginModule Developers Guide

# JAAS | LoginContext

- Kapselt N LoginModules und deren Konfiguration
  - **Flags**
    - Welche LoginModules müssen erfolgreich ausgeführt werden?
  - **Optionen** pro LoginModule
- Identifiziert durch einen **Namen**
  - Name als Einstiegspunkt in Konfiguration
- Instanziierung durch die Applikation
  - Applikation ruft **login()** auf

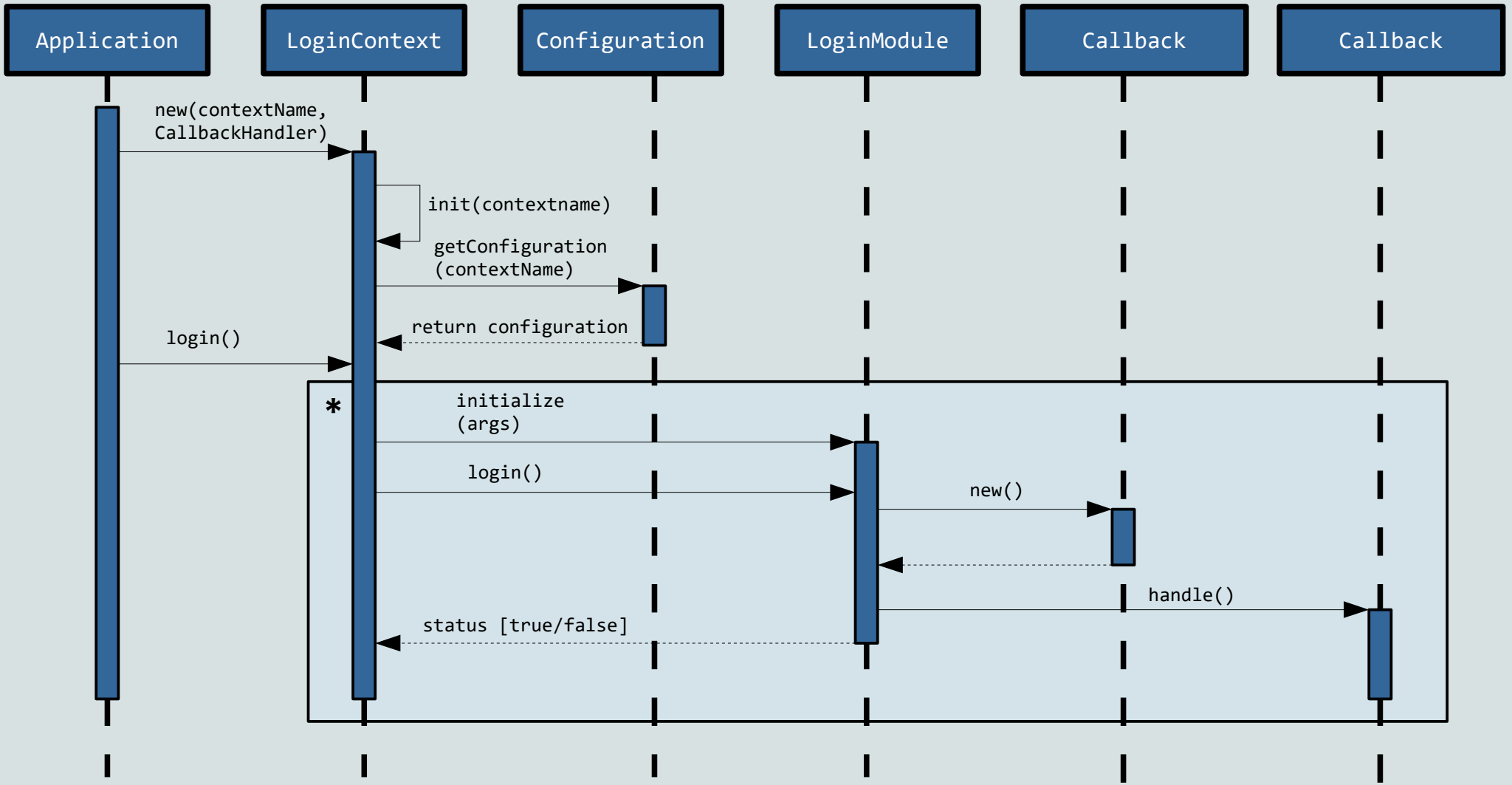
# JAAS | Konfiguration

- Konfiguration der LoginModules (Authentifizierungstechnologien)
  - Implementierung von `javax.security.auth.login.Configuration`
  - Default-Implementierung liest externe Konfigurationsdatei
  - `-Djava.security.auth.login.config=authentication.conf`

# JAAS | Konfiguration

```
AuthenticationExample {
    com.sun.security.auth.module.Krb5LoginModule sufficient;
    com.sun.security.auth.module.UnixLoginModule required;
    <Klassenname des LoginModules> <flags> <options>
    ...
}

/* First step for authentication is to create a LoginContext. */
LoginContext loginContext =
    new LoginContext("AuthenticationExample", new CallbackHandler() {
    ...
}
```

# JAAS | Ablauf


Application
LoginContext
Configuration
LoginModule
Callback
Callback
new(contextName, CallbackHandler)
init(contextname)
getConfiguration (contextName)
return configuration
login()
*
initialize (args)
login()
new()
handle()
status [true/false]

# JAAS | Beispielcode

```
/* First step for authentication is to create a LoginContext. */
LoginContext loginContext = new LoginContext("AuthenticationExample",
    new TextCallbackHandler());

/* Authenticate the Subject using LoginModules. */
loginContext.login();

/* Delivers the authenticated Subject. */
Subject subject = loginContext.getSubject();

/* Associate with current AccessControlContext... */
Subject.doAs(subject, new PrivilegedAction<String>() {
    public String run() {
        /* ... and execute Action as Subject. */

        return …;
    }
});
```

# JAAS | Autorisierung

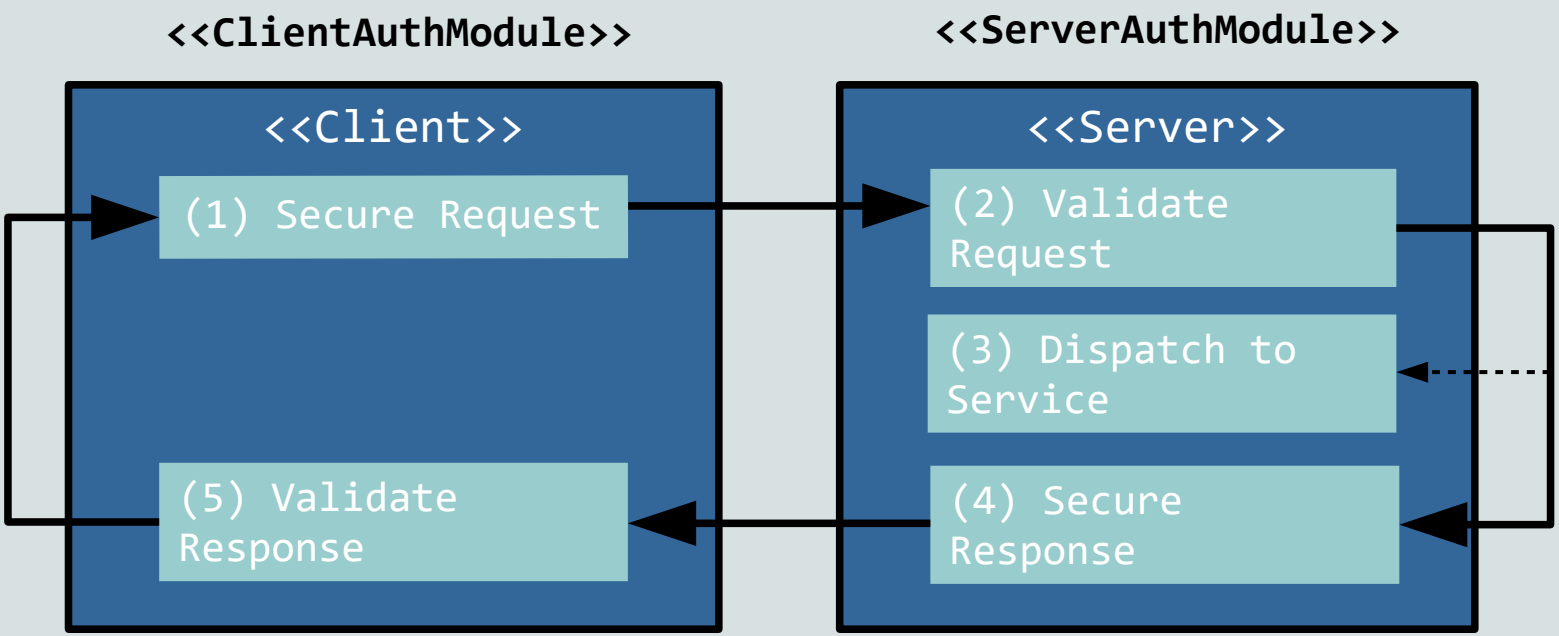
```
grant principal com.sun.security.auth.UnixPrincipal "UnixUser"  {

    permission <full qualified classname> <name> <actions>;
    permission <full qualified classname> <name> <actions>;
    permission <full qualified classname> <name> <actions>;
    ...

};
```

# JASPIC

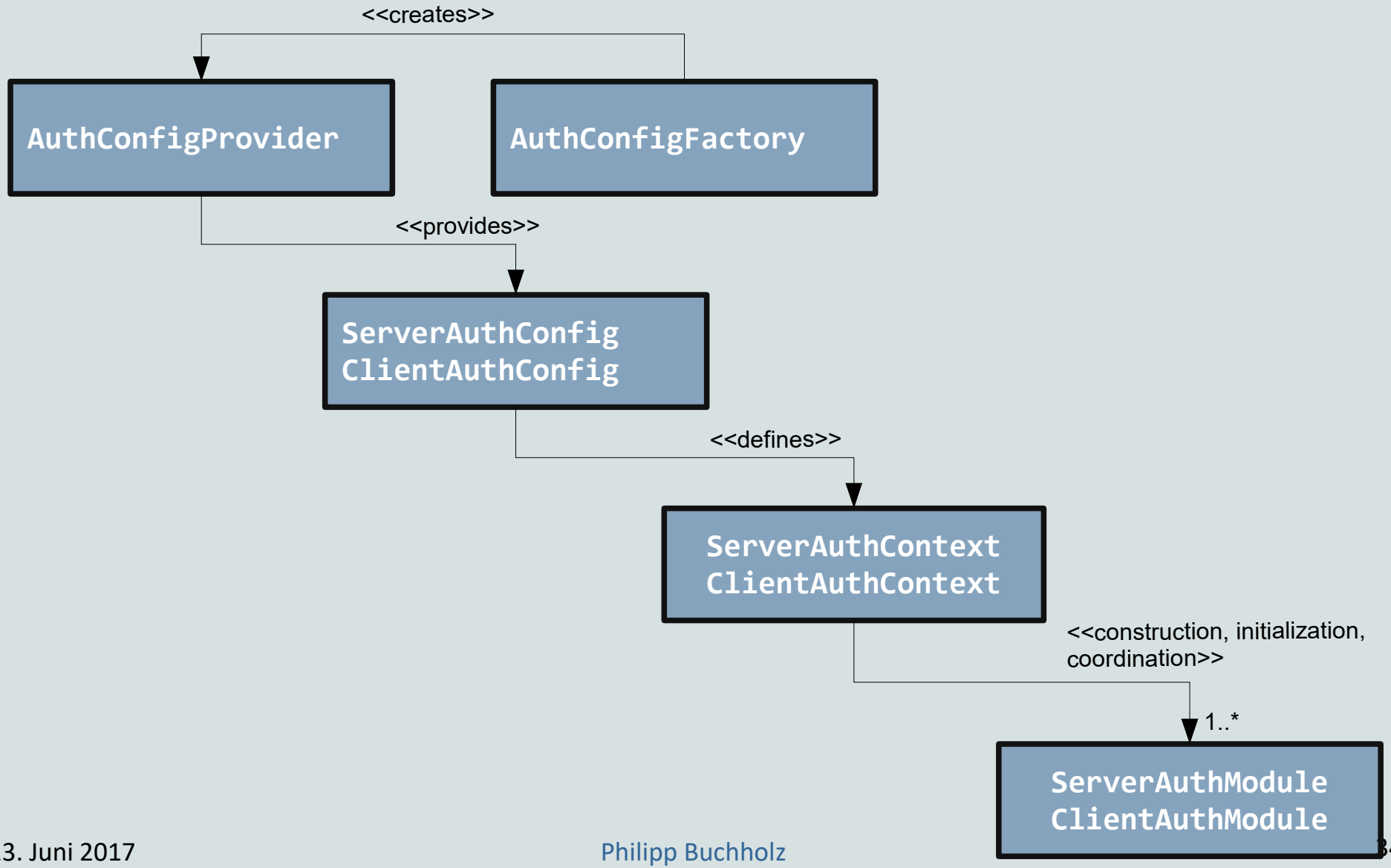
# JASPIC | Allgemein

- **JASPIC** (**J**ava**A**uthentication**S**ervice**P**rovider**I**nterfaceFor**C**ontainers)
  - Basiert auf JAAS
    - Verwendung **CallbackHandler** und **Callbacks**
    - Ähnliches Modell
      - AuthenticationContext
      - ServerAuthenticationModule
      - ClientAuthenticationModule
  - Integration in Java EE Declarative Security
- Standardisierte (**portable**) Authentifizierung
  - JSR-196

# JASPIC | MessagingProcessingModel



# JASPIC | Überblick



# JASPIC | ServerAuthModule (SAM)

- Führt serverseitige Authentifizierung durch.
- `ValidateRequest(MessageInfo, clientSubject, serviceSubject) : AuthStatus`
  - Implementiert die Authentifizierung
  - MessageInfo
    - Zugriff auf Request (`HttpServletRequest`) und Response (`HttpServletResponse`)
  - AuthStatus
    - Authentifizierung erfolgreich oder gescheitert

# JASPIC | Callbacks

- Interaktion mit Container
  - ServerAuthModule erhält CallbackHandler Instanz
    - `javax.security.auth.callback.CallbackHandler`
    - `void CallbackHandler.handle(Callback[] callbacks)`
  - `javax.security.auth.message.callback.CallerPrincipalCallback`
    - Übergabe eines authentifizierten Subjects
  - `javax.security.auth.message.callback.GroupPrincipcalCallback`
    - Übergabe von Gruppen eines authentifizierten Subjects

# JASPIC | ServerAuthContext

- Enthält und koordiniert einen oder mehrere ServerAuthModules (SAMs)
  - `javax.security.auth.message.config.ServerAuthContext`
  - Priorisiert zwischen ServerAuthModules (sufficient alternatives)

# JASPIC | ServerAuthConfig

- Konfiguration eines ServerAuthContext und dessen ServerAuthModules.
  - `javax.security.auth.message.config.ServerAuthConfig`
  - Konfiguration von ServerAuthContext
    - Pro Applikationscontext und MessagingLayer
    - Unter Beachtung der Security-Policies

# JASPIC | AuthConfigProvider

- Stellt die verfügbaren Konfigurationen (**ServerAuthConfig**) zur Verfügung.
  - `javax.security.auth.message.config.AuthConfigProvider`
  - Abfrage einer bestimmten Konfiguration
    - **Applikationskontext** und **MessagingLayer**
  - Unterschiedliche Implementierungen denkbar.
    - XML, JSON, ...

# JASPIC | AuthConfigFactory

- Registry an verfügbaren AuthConfigProvidern.
  - `javax.security.auth.message.config.AuthConfigFactory`
- Programmatisches Registrieren von AuthConfigProvidern
  - Beispielsweise innerhalb ServletContextListener
  - `registerConfigProvider(...)`

# JASPIC | Java EE Security

- Basiert auf Rollen
- Sicherheitseinschränkungen
  - **HttpServlets:**
    - Path und Method
  - **EJBs / CDI-Beans**
    - Absicherung auf Bean- bzw. Methoden-Basis
  - Definition
    - Deployment-Descriptoren bzw. Annotations
    - Programmatisch

# JASPIC | Java EE Security

```
<!-- Sicherheitseinschränkungen -->
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Backend API</web-resource-name>
        <url-pattern>/api/resource/*</url-pattern>
        <http-method>GET</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>restrictedrole</role-name>
    </auth-constraint>
</security-constraint>

 <!-- Deklarierte Rollen -->
<security-role>
    <role-name>serviceuser</role-name>
</security-role>
<security-role>
    <role-name>restrictedrole</role-name>
</security-role>
```

# JASPIC | Java EE Security

```
@WebServlet(urlPatterns = "/api/service1")
@ServletSecurity(httpMethodConstraints = {
        @HttpMethodConstraint(value = "GET", rolesAllowed =
{ "restrictedRole" }) })
public class SecuredServlet extends HttpServlet {

    private static final long serialVersionUID = 85494674315909227l7L;

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        ...
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        ...
    }

}
```

# JASPIC | Java EE Security

```
@RolesAllowed({"Administrator"})
@Stateless
public class AdminService {

    @Resource
    private EJBContext ejbContext;

    public void startSecuredOperation() {

    }

    @RolesAllowed({"Supervisor"})
    public void startSupervision() {

    }

    public void createPayroll() {
        ejbContext.getCallerPrincipal().getName();
        ...
    }

}
```

# JASPIC | Java EE Security

- JASPIC ServerAuthModule
  - Teilt Container CallerPrincipal mit
    - `CallerPrincipalCallback`
  - Teilt Container Rollen mit
    - `GroupPrincipalCallback`
- Custom Authentication
  - Integriert in declarative / programmatic security

# Live Demo