SaferApps

Sichere Integration von Unternehmensanwendungen in bestehende IT-Infrastrukturen

Helmar Hutschenreuter

Deutsches Forschungszentrum für Künstliche Intelligenz
Forschungsbereich „Cyber-Physical Systems“

FoMSESS-Jahrestreffen 2016

Entstehung

UCS – „Univention Corporate Server“

- Debian-basiertes Serverbetriebssystem für kleine und mittlere Unternehmen
- SaferApps ist ein gemeinsames Projekt mit dem Unternehmen Univention[1]

UCS App Store

- Bereitstellung von Unternehmensanwendungen / „Apps“
- App-Installation per „Mausklick“ auf UCS-Instanzen
- *Vorlage:* Smartphone-Betriebssysteme
- Beitrag von SaferApps:
  - Apps sollen abgesichert installiert werden
  - Automatische Absicherung während der Installation und im Betrieb

[1] univention.de

Problematik

Große Bandbreite von Apps

- ownCloud, OpenVPN, Bacula, Zarafa, OX App Suite ...

Apps sind nicht vertrauenswürdig

- Erhalten oft Zugriff auf sensible Unternehmensdaten
  - Unkontrollierte Weitergabe oder Änderung von Daten
  - Gefährdung des Betriebssystems und dritter Anwendungen
  - Apps werden durch potenzielle Sicherheitslücken zur Gefahr
- Anwendungsspezifische Absicherung erforderlich
  - Einschränkung des Informationsfluss
  - (Über)fordert meist IT-Administrator

Situation

Auf einer UCS-Instanz stehen Apps Ressourcen zur Verfügung:

- Dateisystem
- Netzwerk
- LDAP-Verzeichnis

Apps nutzen diese Ressourcen . . .

- zur Kommunikation – z.B.
  - Adressbuch-App schreibt Daten ins LDAP-Verzeichis, welche E-Mail-App liest
  - App-Instanzen verbinden sich gegenseitig per Netzwerkverbindung
  - . . .
- zur Konfiguration – z.B.
  - Konfigurationsdateien
  - Systembenutzerverzeichnis im LDAP-Verzeichnis
  - Laden von Updates und Konfigurationsdaten via Netzwerk
  - . . .

Ziele

- Automatische Absicherung
- IT-Administrator soll Informationen zur Einschätzung von Apps erhalten, um Installationsentscheidung zu treffen

Nachvollziehbare Information

- Weshalb verlangt die App Zugriff?
  (ggf. für welche Funktionen ist das nötig)
- Mit welchen anderen Apps kommuniziert die App?
  - Was bedeutet es, wenn Zugriff erteilt wird?
    (ggf. für sensible Daten)

Automatische Abssicherung

- App wird an die Zugriffe gebunden,
  über die der Administrator informiert wurde.
- Weitere Zugriffe sind der App nicht möglich.

Lösungsansatz

App wird in einen Container gekapselt

- Container enhält die App und alle ihre Abhängigkeiten, . . .
- erlaubt „klare Trennung“ von App und UCS-Instanz

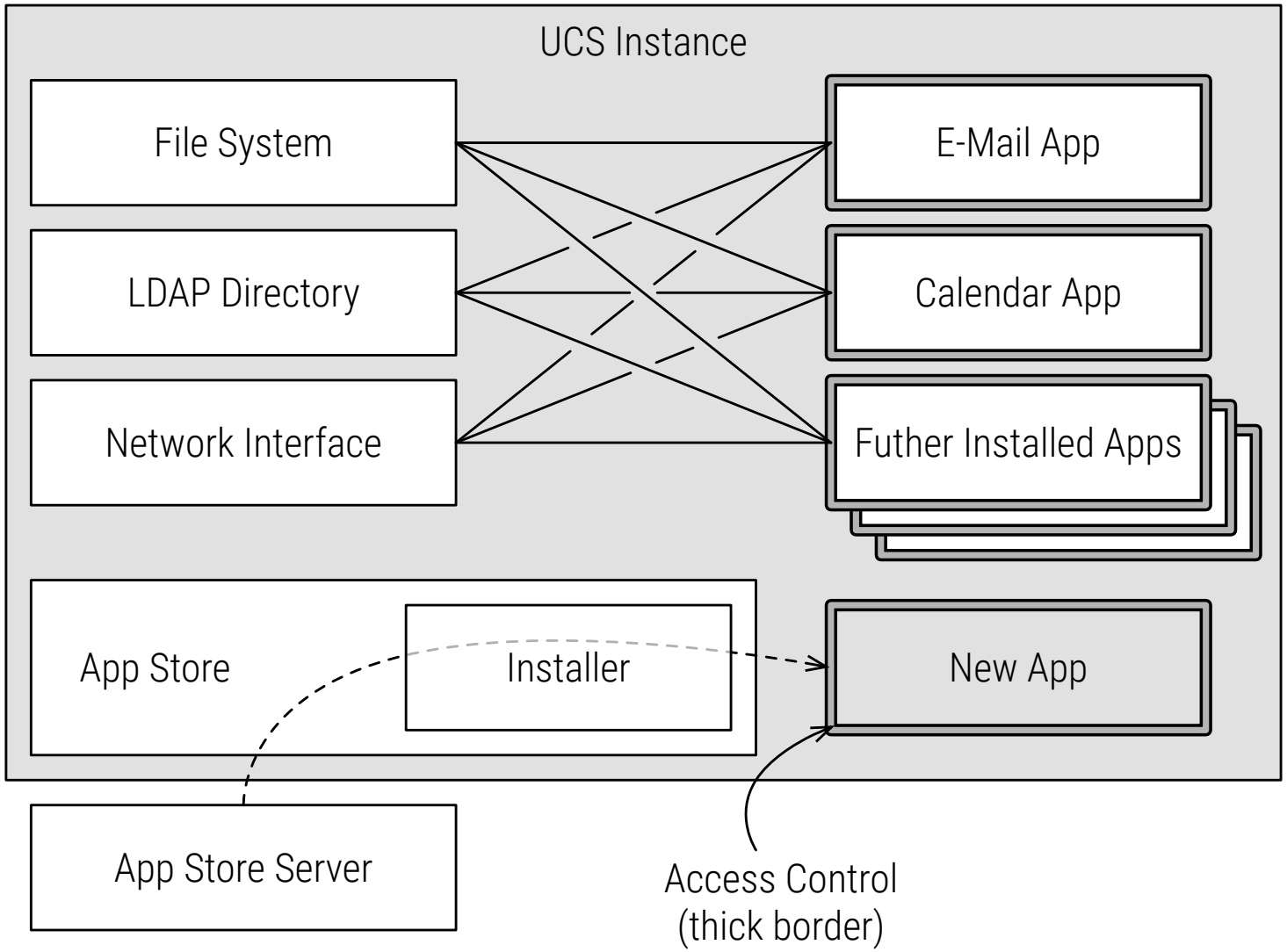
Zugriffskontrolle schränkt Zugriffe des Containers ein

- *Unterscheidung:* Zugriffe außerhalb und innerhalb des Containers
- *Standardmäßig:* Zugriffe außerhalb nicht erlaubt!
- Zugriffe außerhalb müssen explizit freigegeben werden
  – via Sicherheitspolitik (Policy)

Umsetzung via . . .

- Docker – Container
- SELinux Policy Module – für Datei- und Netzwerkzugriffe
- LDAP ACL Regeln – für Zugriffe auf LDAP-Verzeichnis

Überblick


UCS Instance
File System
LDAP Directory
Network Interface
E-Mail App
Calendar App
Futher Installed Apps
App Store
Installer
New App
App Store Server
Access Control
(thick border)

Definition der Zugriffsregeln

- Policy Regeln erteilen immer Berechtigungen, ...
- werden durch den App-Hersteller festgelegt, ...
- da dieser die Bedürfnisse seiner App kennt

Regelaufbau

- Eine Regel ist eine Menge von Regelattributen
- Standardattribute

Attribut	Bezeichner	Werte
Regeltyp	`type`	`file`, `net`, `ldap`
Berechtigung	`permission`	`ro`, `rw`, `listen`, `connect`

- Weitere Attribute abhängig vom Regeltyp

Regeltyp	Beschreibung	Bezeichner
`file`	Datei-, Verzeichnispfad	`path`
`net`	Host-, Port-Kombination	`socket`
`ldap`	Bereich im LDAP-	`area`
`ldap`	LDAP Filter	`filters`
`ldap`	Eingrenzung auf Attribute	`permitted_attrs`

Beispiele für Zugriffsregeln

Regel zum Dateizugriff

```
1 type: file
2 permission: rw
3 path: "/AppData/Zarafa/"
```

Regel zum Netzwerkzugriff

```
1 type: net
2 permission: connect
3 socket:
4   host: 134.96.191.142
5   port: 443
```

Regel zum LDAP-Zugriff

```
 1 type: ldap
 2 permission: ro
 3 area:
 4   dn: ou=employees,o=example
 5   dn_selector: one
 6 filters:
 7   - "telephoneNumber=*"
 8   - "emailAddress=*"
 9   - "objectClass=inetPerson"
10 permitted_attrs:
11   - "firstname"
12   - "surname"
```

Kollision von Zugriffsregeln I

- Kollisionen müssen erkannt werden, . . .
- denn sie deuten auf einen Informationsfluss zwischen Apps hin

Eine Kollision tritt auf, wenn . . .

- die Regeln einen gemeinsamen (Teil)Speicherbereich innerhalb derselben Ressource adressieren und . . .
- sich ihre Zugriffsrechte überschneiden:

Perm. App 1	Perm. App 2	Implikation
`ro`	`ro`	Keine Kommunikation
`ro`	`rw`	App 1 liest Daten von App 2
`rw`	`ro`	App 1 sendet Daten an App 2
`rw`	`rw`	App 1 sendet an und liest von App 2

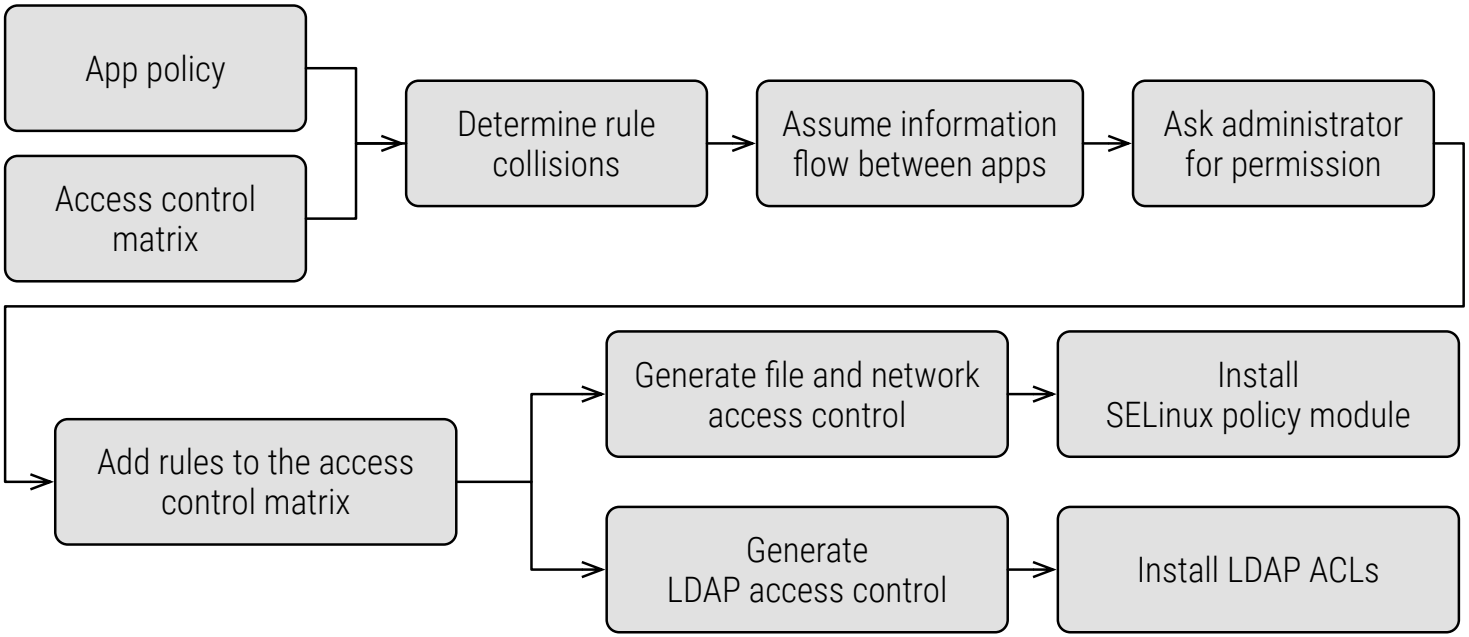
Kollision von Zugriffsregeln II

- Eine Policy Regel $r \in R$ ist eine Menge von Attributen $a \in A$.
- Ein Attribut ist ein Tupel (k, v)
  aus einem Typ k und einem Attributwert v
- Eine Regel kann nie mehrere Attribute desselben Typs enthalten

- Rule Collision Check
  $rcc : R \times R \longrightarrow \{\top, \bot\}$
  - $rcc(r_1, r_2) = \bot$ wenn gilt
    $\exists (k, v) \in r_1, (k', v') \in r_2 : k = k' \wedge v \cap v' = \emptyset$
  - $rcc(r_2, r_2) = \top$ sonst

- $rcc(r_2, r_2) = \top \Longrightarrow r_1$ und r_2 kollidieren

Installation



Grundlegende Installationsschritte...

1. Erkennung von Regelkollisionen zwischen App Policy und Access Control Matrix
2. Informationsfluss schlussfolgern
3. Freigabe durch den IT-Administrator erfragen
4. Zugriffskontrolle erstellen

LDAP-Zugriff

Was ist ein LDAP-Verzeichnis?

- Zentrale Datenbank für Verwaltungsdaten (Nutzer, Kontaktdaten, etc.)
- Bietet eine eigene Zugriffskontrolle,
  welche über ACL (Zugriffsregeln) konfigurierbar ist

Wie erfolgt der LDAP-Zugriff?

- App authentisiert sich am LDAP mit eigenen Zugangsdaten oder . . .
- mit den Zugangsdaten des jeweiligen Nutzers

Problematik

- Zugriffsrechte im LDAP-Verzeichnis sind i.d.R.
  vom angemeldeten Nutzer abhängig und . . .
- Zugriffsrechte können daher variieren

Lösungsansatze

1. Teile des LDAP-Verz. werden in den Container repliziert
   - \+ App kann maximal auf die replizierten Daten zugreifen
   - – Performance und Datensynchronisation ist problematisch

2. ACLs werden an die IP-Adresse des Containers bzw. der App gebunden
   - \+ Einfach realisierbar
   - \+ Performance wird nicht beeinträchtig / Daten müssen nicht synchronisiert werden.
   - – Nutzerspezifische ACLs werden überschrieben (ggf. werden zuviele Rechte eingeräumt)

   - ▸ **Ausgewählter Ansatz**

Authentifizierung der Nutzer

Problemstellung

- Manche App erfordern eine Anmeldung der Systembenutzer
- App prüft die Zugangsdaten
  i.d.R. durch Anmeldung am LDAP-Verzeichnis
- Zugangsdaten des Nutzer dürfen aber nicht in die App gelangen!

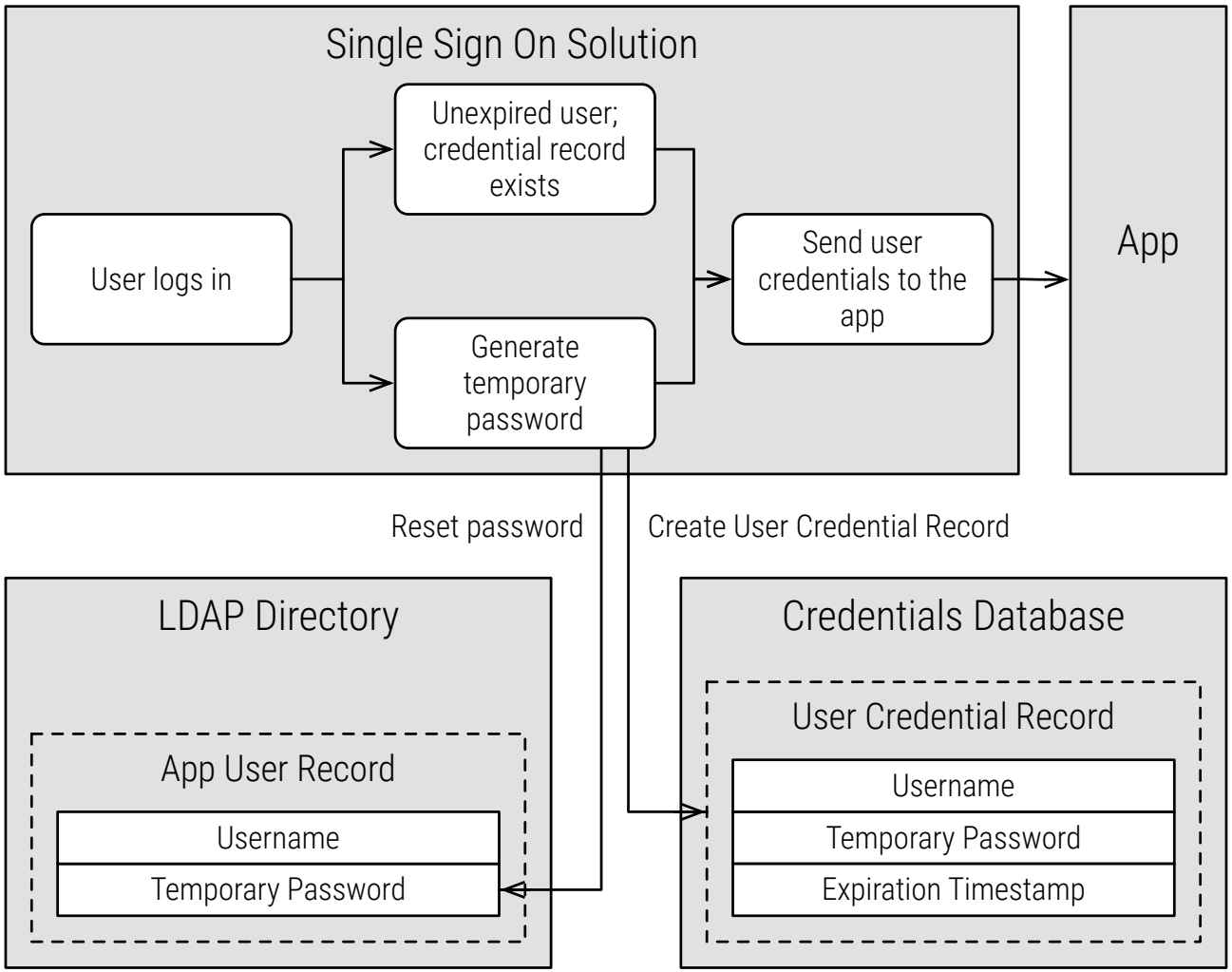
Single Sign On (SSO) – intuitiver Ansatz

1. App leitet den Nutzer zur SSO-Lösung weiter
2. Nutzer authentisiert sich gegenüber der SSO-Lösung
3. SSO-Lösung teilt App mit, dass der Nutzer zu ihrer Nutzung authorisiert ist

- **Zusätzlicher Vorteil:**
  Nutzer muss sich nur einmalig authentisieren
- **Nachteil:** Nur wenige Apps sind kompatibel mit SSO

Alternatives SSO Konzept

Austausch der Zugangsdaten



Ausblick

Kurzfristig

- Spezifizierung der Übersetzung von Policy Regeln
- Sicherheit der Web-Schnittstelle von Apps?

Mittelfristig

- Formalisierung der Sicherheitsgarantien
- Beweis der Sicherheitsgarantien
  - Erkennung und Interpretation von Regelkollisionen
  - Übersetzung von Policy Regeln in SELinux Policy Modules oder LDAP ACL Regeln

Langfristig

- Feingranularere Informationsflusskontrolle (Non-Interference)

Vielen Dank für Ihre Aufmerksamkeit!

Kontakt Helmar Hutschenreuter

Anschrift: DFKI Bremen
Bibliothekstraße 1 (MZH)
28359 Bremen

Tel.: +49 421 218 59833
Fax.: +49 421 218 9859833
E-Mail: helmar.hutschenreuter@dfki.de
Web: dfki.de/cps