# Single Sign On: Passwörter in Zeiten von NSA Cloud Sync

Daniel Schreiber

Universitätsrechenzentrum

19. März 2016

# Single Sign-on und Single Password

## Philosophie seit 25 Jahren

Ein Passwort für alles.

## Konsequenzen

- Hoher Komfort für den Nutzer
- Technologisch schön (viele kerberisierte Dienste) :-)
- Sicherheit: ???

# Single Sign-on und Single Password

## Philosophie seit 25 Jahren

Ein Passwort für alles.

## Konsequenzen

- Hoher Komfort für den Nutzer
- Technologisch schön (viele kerberisierte Dienste) :-)
- Sicherheit: ???

# Systemlandschaft am URZ der TU Chemnitz

- Linux als primäre strategische Systemplattform
  - Single Sign on mit Kerberos: SSH, SMTP, IMAP, LDAP, AFS
  - Web-Login mit Shibboleth (akzeptiert auch Kerberos)
- Windows mit Active Directory
- weitere Systeme:
  - WLAN (eduroam), LAN (802.1x)
  - Drucker/Kopierer
  - … über 100 Dienste für Nutzer

# Single Sign-on und Single Password

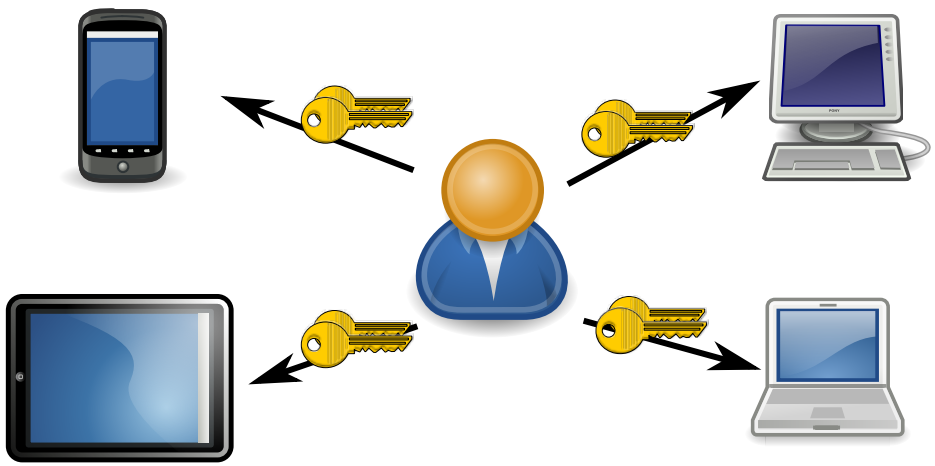
Vorteile:

- Benutzer muss sich nur **ein** Passwort merken
  → Durchaus zumutbar und vermeidet Klebezettel mit Passwörtern am Monitor
- Benutzer kann sich mit einmal beschafften Kerberos-Ticket an einer Vielzahl von Diensten anmelden (Shibboleth, IMAP, SMTP, AFS, SSH, …)
  → Passwort geht im *Idealfall* nicht ständig über die Leitung
- (Hoffentlich) größere Hemmschwelle bezüglich Passwortweitergabe

# Single Sign-on und Single Password

Nachteile:

- Das eine Passwort verteilt sich auf eine Vielzahl von Geräten (BYOD), oft mit NSA-Cloud-Sync gebackupt
- Passwort wird oft im Klartext abgespeichert
- Wird das Passwort gephisht/abgefangen → **Totalschaden**
- Bei Passwortänderung muss an gefühlt 30 Stellen[1] geändert werden



[1] WLAN, 2x Mail, SVN, WebDAV, Sync'n'Share, Kalender, VPN, …

# Motivation: Dienstspezifische Passwörter

- Ausgangssituation: DFN Betriebstagung Oktober 2014:
  - Mehrere Schwachstellen in mobilen Geräten bezüglich WLAN-Sicherheit entdeckt
  - eduroam-Passwort (== URZ Passwort) kann von Angreifern einfach ausgespäht werden

  → Motivation für separates WLAN-Passwort
- Evaluation Sync&Share-Dienst mit Shibboleth-Integration
  - separates Owncloud-Passwort zum Abspeichern in WebDAV/Sync-Klienten
  - URZ-Passwort wäre zwar denkbar, aber aus Security-Sicht nicht sinnvoll (da auf allen Geräten im Klartext gespeichert)

  → Dienst Nr. 2 mit separaten Passwort

# Diskussion

„Wenn wir einmal mit separaten Passwörtern anfangen (WLAN, evtl. Sync&Share), warum dann nicht gleich konsequent und konsistent für eine größere Menge von Diensten?“

# Idee

- Ein separates *automatisch generiertes* Passwort für jeden Anwendungsfall
  → Für jeden Dienst auf jedem Gerät ein anderes Passwort
- Besonders für Dienste mit mobiler Relevanz und hohem Missbrauchspotential (Mail, WLAN, …)
- Konsistente, einheitliche Passwortverwaltung für den Nutzer (über IdM-Portal)
- URZ-Passwort bleibt weiterhin „Master-Passwort“ für Shibboleth, Kerberos & Co.
  → SSO weiterhin möglich

# Ziel

# Vor- und Nachteile

Vorteile:

- Schadensbegrenzung im Verlustfall
- Bequemes Widerrufen verlorener Passwörter
- Bei Dienstmissbrauch reicht Sperre der Dienstpasswörter
- Änderungen des URZ-Passwortes sind wesentlich angenehmer (Akzeptanz für Passwort-Policy würde steigen)
- Öffnet Tür für 2-Faktor-Auth

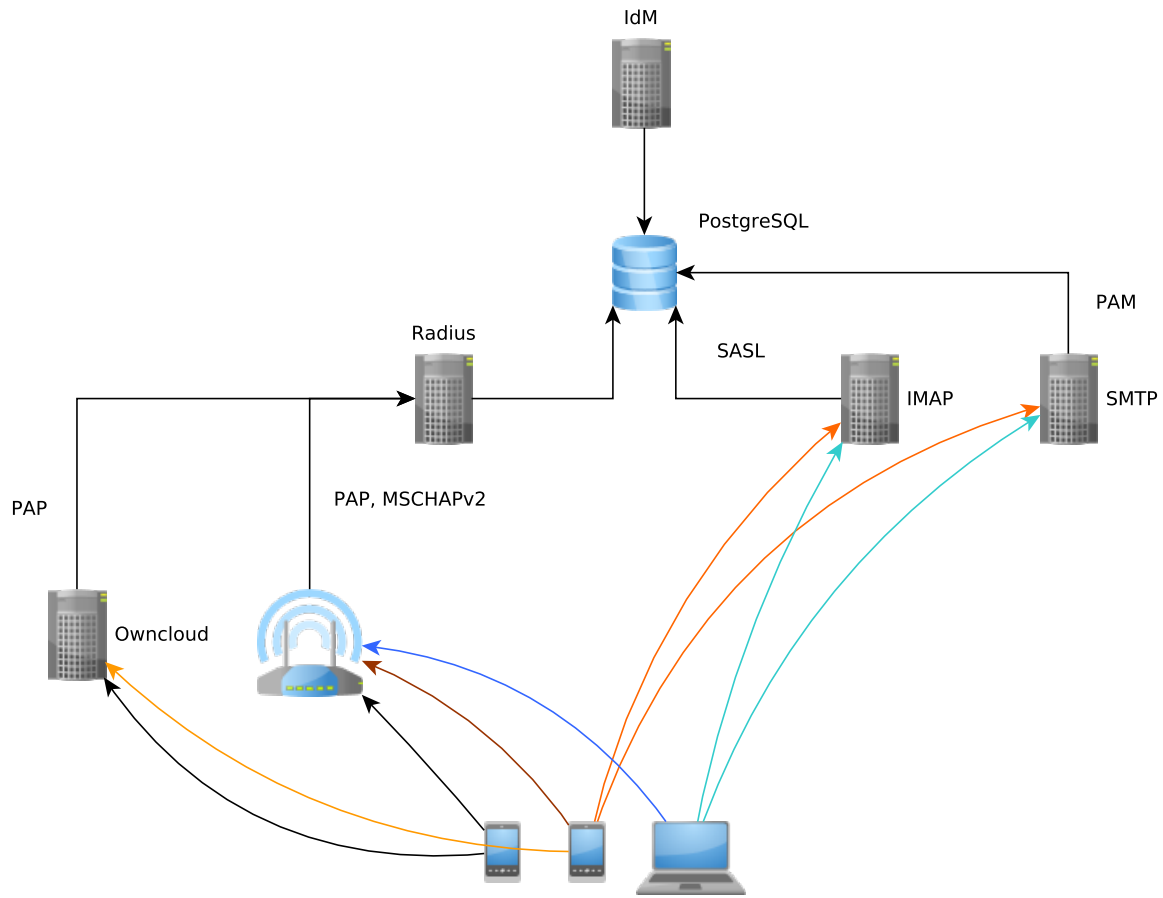
Nachteile:

- evtl. Supportaufwand (aber: reduziert Phishing-Schäden)
- Projektaufwand (Umsetzung, Entwicklungsarbeit, Security-Audit, …)
- Exchange, Sharepoint

# Umsetzung

- Benutzerkennung bleibt bestehen: `otto@tu-chemnitz.de`
- IdM generiert zusätzliche zufällige Passworte
  - Nur für **einen** Dienst gültig
  - einmalig angezeigt → nur zum Abspeichern geeignet
- Passworthashes in HA PostgreSQL Cluster gespeichert
  - WLAN: NT-Hashes wegen MSCHAP
  - andere: bcrypt
- Zugriff auf Datenbanktabelle immer über Funktionen → Angreifer auf kompromittiertem Anwendungsserver kann keine Hashliste abgreifen
- Zugriffe in memcached loggen und periodisch in PostgreSQL persistieren

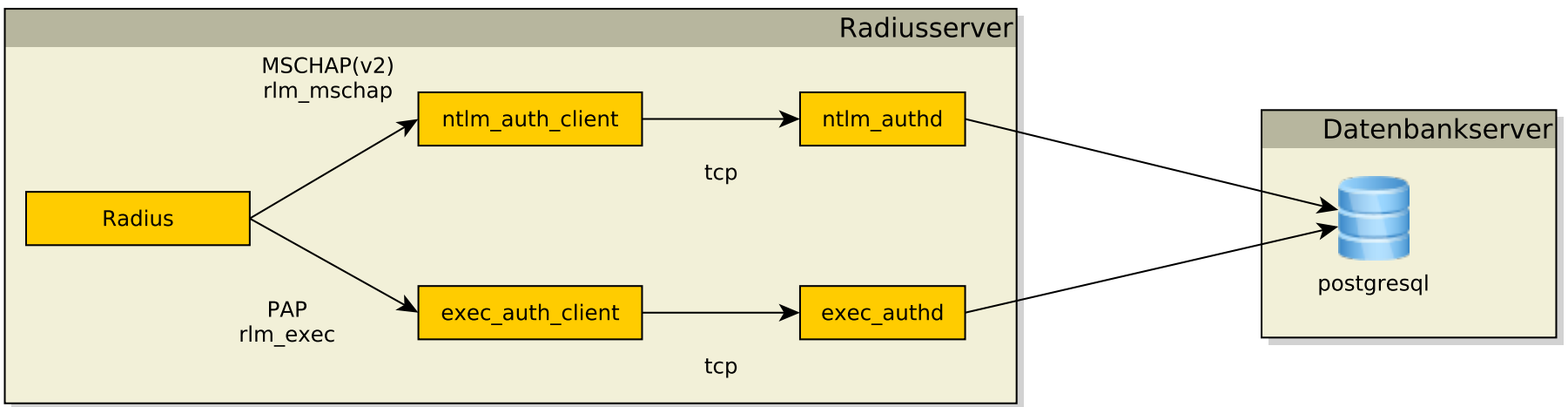
# Architektur



# Integration

- PAM: pam_pgsql.so
- SASL: Auth über PAM, Caching
- Freeradius:
  - mschap über eigenes ntlm_auth Binary, prüft alle Passwörter des Nutzers aus PostgreSQL
  - PAP über rlm_exec, gleicher Mechanismus wie pam_pgsql.so
  - Prüfung in Daemon, Connection pooling

# Architektur (Radius)



# PostgreSQL API

```
auth_user(_user varchar, _password varchar, _service
    varchar) returns boolean
radius_user(_user varchar, _service varchar) returns
    table (salted_hash varchar)

create_or_update_auth_data(_user varchar, _salted_hash
    varchar, _hash_type varchar, _service varchar,
    _resource_id integer, _expire_date timestamp with
    time zone) returns void
delete_auth_data(_resource_id integer) returns void
```

# Status

- Erledigt
  - Eduroam
  - exim/cyrus
  - Owncloud
  - WebDAV
  - SVN
- Offen
  - **Windowsdienste**
  - Marketing
  - 2FA