# Login-“Tickets“ mit OpenSSH Certificates

Michael Ströder <michael@stroeder.com>

# ldapwhoami

- Michael Ströder <michael@stroeder.com>, Freiberufler
- Schwerpunkte
  - X.509-PKI, angewandte Verschlüsselung, dig. Signatur
  - Verzeichnisdienste (LDAP etc.), Identity & Access Management
  - Single Sign-On
  - Mehr-Faktor Authentifizierung
- Open Source / Freie Software:
  web2ldap, Æ-DIR, OATH-LDAP, EKCA

# Authentizität des öffentlichen Schlüssels

- Schlüssel muss sicher einem Namen zugeordnet werden
- Gängige Praxis:
  - Hosts: trust on first use (TOFU) bei Client
  - Benutzer: Passworteingabe bei Self-Service
- Besser:
  - Hosts: KnownHosts-Datei verteilen → skaliert nicht
  - Benutzer: kein Self-Service, Admin prüft (zuverlässig?)
- nicht wirklich praktikabel und sicher

# Sicherheit des privaten Client-Schlüssels

- Sicherheitsanforderungen (neben allg. Anforderungen):
  - Algorithmus / Schlüssellänge
  - Erzeugung Zufallszahlen
  - Speicherung: Datei (Passphrase optional) o. Smartcard
  - Einschränkung Nutzungsdauer (Key-Rollover)
- Server-seitig prüfbar:
  - Algorithmus / Schlüssellänge
  - Debian weak keys (macht das jemand?)

# sshd_config -- AuthorizedKeysFile

- Default: $HOME/.ssh/authorized_keys
  - ssh-copy-id: Benutzer verwaltet autorisierte Schlüssel
  - flexibel, aber keinerlei Kontrolle → extremer Wildwuchs
- Besser: /etc/ssh/authorized_keys/%u
  - nur privilegierte Schreibzugriffe, kein ssh-copy-id
  - zentrale Aussteuerung von Schlüsseln (CRON-Job o.ä.)
  - benötigt Benutzerverwaltung (z.B. Æ-DIR)
- *AuthorizedKeysCommand* für externen Zugriff

# Mögliche Lösung: Crypto Token / Smartcard

- Hardware-Treiber
  → Entwicklung dauert…
  → Client-Rollout
- Häufige Produktanpassung
  → Kompabilitätsprobleme
  → Probleme bei Beschaffung und Support
- Sehr langsame Signaturberechnung (800 - 2000 ms)
  → unbrauchbar für Massenzugriffe (ansible, Fabric, o.ä.)
- meist unbenutzbar in Remote-Sitzungen (VNC o.ä.)

# OpenSSH User Certificates

- Nicht X.509 !
- Principal (vulgo Benutzername/-kennung)
- öffentlicher Benutzersschlüssel
- öffentlicher CA-Schlüssel
- Key-ID
- Seriennummer
- Gültigkeitsdauer
- Key Options (source-address, force-command, permit-*)

# Key Revocation Lists

- Liste widerrufener OpenSSH Certificates mit
  - öffentlicher Benutzersschlüssel
  - Fingerprint Benutzersschlüssel
  - Key-ID
  - Seriennummer
- unsigniert
- Verteilung auch bei Config-Management langsam (> 4h)
- Lösung: sehr begrenzte Gültigkeitsdauer der Zertifikate

# CA Policy -- User Certificates

- Benutzerauthentifizierung (Password, OTP, etc.)
- Algorithmen / Schlüssellänge
- Gültigkeitszeitraum / -dauer
- Länge der Passphrase
- Proof of possession of private key (PoP)

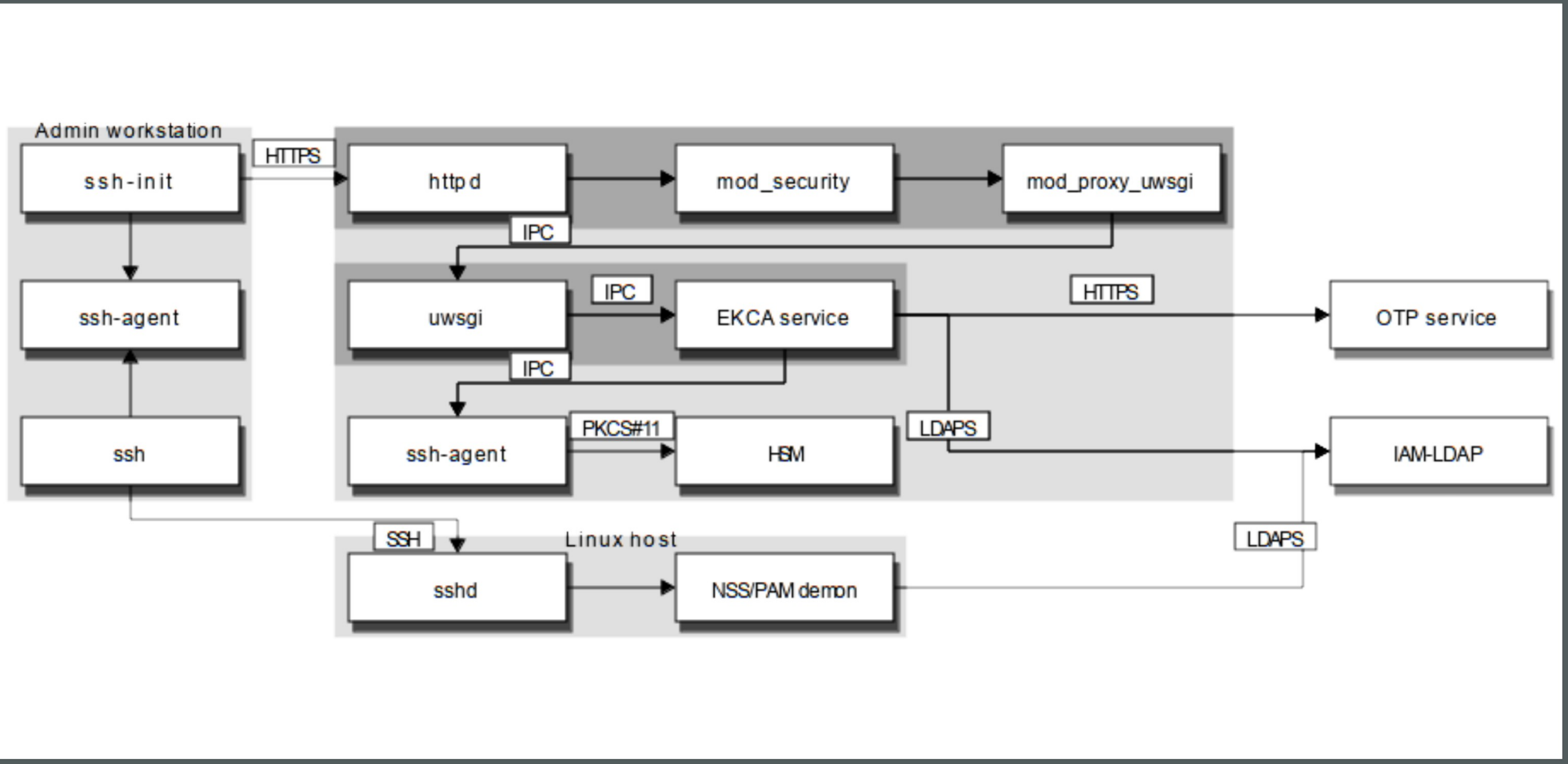
# CA Security

- CA-Schlüssel kompromittiert alles kompromittiert!
- Maßnahmen:
  - Private CA-Schlüssel (Plural) im HSM
  - Private CA-Schlüssel inaktiv auf Vorrat
- Benutzung *ssh-agent* mit PKCS#11-Integration ohne automatischen Start (manuelle PIN-Eingabe)
- Strikte Separation der Subkomponenten
- Härtung! Reduktion, AppArmor, mod_security, etc.

# sshd_config -- User Certificates

- *TrustedUserCAKeys* enthält öffentliche CA-Schlüssel
  - nur der öffentliche CA-Schlüssel, keine Meta-Daten
  - Keine CA-Kette (CA chain): Widerruf der CA unmöglich!
  - Am besten gleich mehrere Schlüssel für CA Key-Rollover!
  - Config-Management!
- *AuthorizedPrincipalsFile* erlaubt Benutzer-Mapping
- *AuthorizedPrincipalsCommand* für externen Zugriff
- *RevokedKeys* gibt KRL-Datei an

# Architektur



# Nutzung HSM via PKCS#11

- PKCS#11: C-API für Zugriff auf Crypto-Tokens
- PKCS#11-Module sind Shared Libs z.B. vom Hersteller
- herstellerspezifische Konfiguration von Slots etc.
- Nutzung in der CA:
  - ssh-add -s vendor_pkcs11_module.so
  - ssh-keygen -s ca_key.pub -D vendor_pkcs11_module.so

# Kompabilität Client-Software

- Yepp:
  - OpenSSH 5.6+ (verfügbar auch als Windows-Port)
  - ansible
- Nope:
  - PuTTY et al
  - Appliances, Netzwerk-Hardware
- Unklar
  - Appliances, Netzwerk-Hardware → Hersteller anfragen!

# Demo
# User Certificates

# UserKnownHostsFile

- Default: $HOME/.ssh/known_hosts
- Benutzer akzeptiert unbekannten Schlüssel beim ersten Zugriff (TOFU)
- In der Praxis keine ausreichende Prüfung
  → MITM-Attacke beim ersten Zugriff möglich
- Insbesondere problematisch bei häufiger Neuinstallation
- Authentifizierung während Host-Installation notwendig
- mögliche Lösung: DNS RR *SSHFP*, aber braucht DNSSEC

# Host Certificates

- *HostCertificate* in sshd_config gibt Zertifikatdatei an
- Ähnliche Metadaten wie in User Certificates
- Name ist client-seitig verwendeter FQDN
- Host Certificate ausstellen
  - ssh-keygen [..] -h -n host.example.com host_key.pub
  - Idealerweise bei NeuInstallation (definierter Build-Prozess)
  - Laufzeit?
  - Nachsignieren alter Host-Schlüssel (Cross-Checks!)

# SSH-CA in UserKnownHostsFile

@cert-authority *.example.com ssh-rsa AAAAB3N[..]Vb5D

- je Domain-Pattern andere CA möglich
- am besten mehrere CA-Schlüssel für Key-Rollover

# Fragen?