C Sicherheit im Verhältnis System – Benutzer

C.1 Authentifizierung von Benutzern

1 Problem

- Wie stellt man sicher, dass der richtige Benutzer angebotene Dienste benutzt?
- Ist der Benutzer auch wirklich derjenige, der er vorgibt zu sein?
- Authentifizierungsprinzipien
  - ➤ etwas, was der Benutzer weiß
  - ➤ ... besitzt
  - ➤ ... ist
- … oder allgemeiner: Nachweis der Identität eines Subjekts
  - ➤ Vorlegen von Credentials

2 Passwort (Wissen)

- Benutzer weist sich durch Kenntnis eines vereinbarten Geheimnisses aus
- System muss Passwörter sicher verwalten
  - ➤ Passwort + Salt wird als Hash (DES/MD5/SHA-1/...) in einer nicht-öffentlichen Passwortdatei gespeichert
- Vom Benutzer übergebenes Passwort wird mit gespeichertem Salt erweitert, ,,verschlüsselt'' und der entstehende Hash verglichen
- Salt: Ablegen des gleichen Passworts mit unterschiedlichem Salt führt zu unterschiedlichen Hashes
  - ◆ Passwort ,,geheim'':
    `kUp6/YId/ULew`
    `X5tLeK8skOc4Q`
  - ◆ Dadurch wird das Vorgenerieren von Tabellen mit verschlüsselten Passwörtern (*Rainbow Tables*) sehr aufwändig (Mit Unix-Crypt kann jedes Passwort auf 2^{14} verschiedene Weisen dargestellt werden)
- Schwächen von Passwörtern

3 Einmalpasswort (Wissen)

- Vorgegebene Liste mit einmal gültigen Passwörtern
  - ➤ TANs
- Mithören des Passworts erlaubt keinen Zugang
- Verlust/Diebstahl der Passwortliste ermöglicht unauthorisierten Personen Zugang
- Ermöglicht sicheres Einloggen von nicht vertrauenswürdigen Rechnern

```
ZLs01fsA llk7qJzw bchdsAn6 1wSc0Gar 8MG495px
zCjZ7yd7 pBKpf7MR ihGC1bZ3 E3jjLEwC B9n1GZio
JYQcI8bf 9Necnh4B Lam8a17P xxx2CPzq YEan9iBL
9v1iGDuS jwrqSp30 0Z9yvKyo 426HIK4B QYIwh3MO
OYXG096e DAzCuF5f ALOOpww6 JkQp1cLt S3RZpP6A
swbV2OXp 9TUK59MT C57F5wsa ha8NdfBD 5WL9Fz5b
fS9ZZq4M PBccshZ2 ZcWc3Cdu 6tHpWZiL EJssWx2r
```


3 Einmalpasswort (Wissen)

- S/Key-Authentifikation
  - ➤ Basis für verschiedene OTP-Verfahren
  - ➤ mehrmaliges Anwenden einer Hash-Funktion auf ein Passwort
    - Passwort ***s***, kryptographische Hashfunktion ***f***, Seed-Wert ***k***
    - Passwort i: $p_i = f^i(s \mid k)$
    - letztes Passort p_n wird an Server übertragen
    - Authentifikation: Server fordert Passwort p_{i-1} an und überprüft ob $f(p_{i-1}) = p_i$
      - entweder p_i liegt beim Server bereits vor, oder f wird n-i mal angewendet

4 Zeitbasierte Passwortgenerierung (Wissen + Besitz)

- RSA SecurID
  - Token mit Uhr und geheimem Schlüssel (Seed)
  - Rechner mit Uhr und geheimen Schlüsseln aller Tokens
  - Token generiert jede Minute aus der Zeit und dem Schlüssel einen Wert, dieser ist für drei Minuten gültig
  - Authentifizierung am System durch Eingabe des generierten Werts und einer dem Besitzer bekannten PIN
  - Rechner speichert die Uhrabweichung des Tokens (3-Minuten Fenster) um abweichende Tokenuhren auszugleichen
  - Problem: Sollte die Tokenuhr zu stark abweichen ist keine Authentifizierung mehr möglich

5 Challenge-Response Verfahren (Wissen)

- Sehr viele geheime Fragen, deren Antwort nur der Benutzer kennt:
  - Wer ist die Schwester von Marjolein?
  - In welcher Straße war Ihre Grundschule
  - Was unterrichtete Mrs. Woroboff?
    - ➤ oft in Web-Schnittstellen zur Restaurierung vergessener Passwörter
- Input für eine kryptographische Funktion, deren Parameter (z. B. Schlüssel) nur System und Benutzer kennt
  - Einfaches Beispiel: Quadrieren
    - Challenge: 7
    - Response: 49
  - In echten Systemen wird eine leistungsfähige Form von Challenge-Response eingesetzt. z. B.: Cryptocard, Opie
- Einsatz in Geräten zur Authentifizierung von Systemkomponenten (Akkus, SIM-Karten, Toner-Kasetten, ...)

6 Tokenbasierte Identifikation (Besitz + Wissen)

➥ Smartcards/USB-Token

- Verfahren mit symmetrischer oder assymetrischer Verschlüsselung
- Geheimer Schlüssel des Benutzers ist auf Token gespeichert
  - ➤ symmetrisch: Token-Schlüssel-Paare auch auf Rechner abgelegt
  - ➤ assymmetrisch: öffentlicher Schlüssel zu Token ist Rechner bekannt
- Benutzer authentifiziert sich direkt am Token, welches wiederum den Benutzer gegenüber dem Rechner authentifiziert
  - ◆ Token führt kryptographische Operation auf vom System generierter Zufallszahl (challenge) aus, liefert Ergebnis (response) an das System
  - ◆ System überprüft Ergebnis und gestattet Zugang
    - ➤ 2-stufige Authentifizierung durch Besitz und Wissen
- Token-Passwort wird nie an das System übertragen

7 Biometrische Identifikation (pers. Eigenschaften)

- Biometrische Merkmale — notwendige Eigenschaften
  - ➤ Universalität
  - ➤ Eindeutigkeit
  - ➤ Beständigkeit
  - ➤ Quantitative Erfassbarkeit
  - ➤ Performanz
  - ➤ Akzeptanz
  - ➤ Fälschungssicherheit
- statische Merkmale
  - ➤ physiologische Eigenschaften
- dynamische Merkmale
  - ➤ verhaltenstypische Eigenschaften

7 Biometrische Identifikation (pers. Eigenschaften) (2)

- Fingerabdruck
  - ◆ Merkmale des Fingerabdrucks (Minutien) werden mit gespeicherten Werten verglichen
  - ◆ Alle gebräuchlichen Leser lassen sich mit etwas Aufwand täuschen
- Weitere Merkmale
  - ➤ Iris-Erkennung
  - ➤ Fingerlängen-Vergleich
  - ➤ Gesichtserkennung
  - ➤ ...
  - ◆ Leser sind noch nicht wirtschaftlich verfügbar

8 Sicherer Login-Prozess

- Login-Maske könnte von vorherigem Benutzer des Computers nachgestellt worden sein
- Neuer Benutzer meldet sich an gefälschter Login-Maske an, das Passwort wird gespeichert und der echte Login-Prozess wird gestartet
- Lösung: Nicht in Software abfangbares Aktivieren des Loginmanagers durch Secure Access Key (SAK) (In Windows: CTRL-ALT-DEL)
  - ◆ Input-Layer (Ebene oberhalb des Tastatur-Treibers) erkennt die Sequenz +
  - ◆ gezieltes Umschalten auf systemeigenen Login-Prozess

C.2 Authentifikation am Beispiel UNIX

- Authentifikation von Benutzern
  - ➤ einmalig beim Login (Identifikation + Authentifikation)
  - ➤ i. d. R. Passwort-basierte Verfahren (Geheimnis / Wissen)
  - ➤ Datei /etc/passwd
  - ➤ shadowed password file
- Authentifikation erfolgt i. d. R. nur einmalig
  - ➤ anschließend Repräsentanten innerhalb des Systems

C.2 Authentifikation am Beispiel UNIX (2)

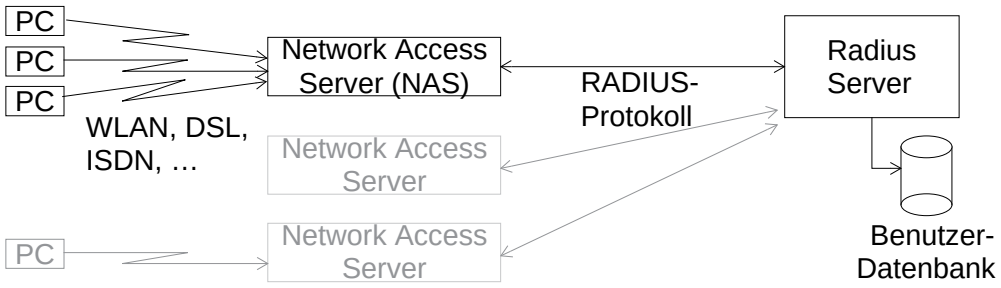
- Benutzer wird innerhalb des Systems durch Stellvertreter-Subjekte (= Prozesse) vertreten
  - ➤ Authentfikation von Prozessen
  - ➤ Credential-Struktur im Prozesskontrollblock (geschützte Datenstruktur = Besitz)
  - ➤ Autorisierungen erfolgen auf Basis der Credentials (uid, gid, ...)
- Schutz von Credentials
  - ◆ außerhalb des Systems
    - ➤ Geheimhaltung (durch Person)
    - ➤ Verschlüsselung (Geheimhaltung durch Technik)
    - ➤ Verwahrung (Besitz)
  - ◆ innerhalb des Systems
    - ➤ Verschlüsselung
    - ➤ Schutz durch Kapselung (Adressräume)

C.3 Authentifikation in verteilten Systemen

- Grundprobleme
  - ◆ Übertragung von Passwörtern
    - ➤ Abhören bei Klartext
    - ➤ Replay bei Übertragung von Hashes
  - ◆ Übertragung der Antwort
    - ➤ Fälschen eines "Access Accept"
- RADIUS (Remote Authentication Dial In User Service)
  - ➤ Passwort-basiertes Authentifzierungsprotokoll
  - ➤ Dial-In, WLAN
- Secure RPC
  - ➤ DES-verschlüsselte RPC-Kommunikation
- NIS
- Kerberos

1 RADIUS

- Herausforderung bei Einwahldiensten
  - ➤ viele Benutzer
  - ➤ mehrere Einwahlknoten
  - ➤ mehrere Administrationsbereiche
  - ➤ Mobilität der Benutzer (Roaming)



- Proxy zwischen NAS und Server ermöglicht mehrere Server mit unterschiedlicher Benutzerverwaltung (RADIUS-Zonen)

1 RADIUS (2)

- einfache AAA-Architektur
  (Authentication, Authorization, Accounting)
  - ➤ *Benutzer* – will Zugriff auf Dienst
  - ➤ *UHO* (User Home Organization) – bietet Benutzer Dienste an (vertraglich)
  - ➤ *Service Provider* – bietet Dienste an (technisch)
  - ➤ *AAA-Server* eines Service Providers – entscheidet über Zugang eines Benutzers zu einem Dienst auf Basis der Aussage der UHO
- RADIUS-Protokoll
  - ➤ Gemeinsames Geheimnis *S* von NAS und Radius-Server
  - ➤ Benutzer-Passwort: PW
  - ➤ Access-Request-Nachricht: *Client-IP, ID, RA, username, Hash(PW, S, RA)*
    RA (Request Authentictor) = 128 Bit Zufallszahl
  - ➤ Access-Accept/Reject-Nachricht: *ID, Response-Authenticator*
    Response-Authenticator: *MD5(... | ID | length | RA | S)*

1 RADIUS (3)

- Schwachstellen
  - ➤ Benutzerpasswort
  - ➤ Daten in Request-Nachricht = Klartext des Response-Authenticators
    ➥ Known-Plaintext-Analyse des Schlüssels *S* möglich
  - ➤ Zufallszahlen *RA* dürfen sich nicht wiederholen
    (sonst Maskierung des Servers möglich)
  - ➤ Nachrichten sind nicht verschlüsselt
    ➥ Anforderung an AAA-Architekturen wird nicht erfüllt
- Verbessertes RADIUS-Nachfolger: DIAMETER
  - ➤ Kommunikation zwischen NAS und Server über IPSec
  - ➤ Kommunikation zwischen Servern über TLS (Transport Layer Security)
  - ➤ …

2 Secure RPC

- Erweiterung des Sun Remote Procedure Call um Authentifikation (nicht um Nachrichten-Verschlüsselung!)
  - ◆ Sun RPC
    - ➤ Konzept zum (fast) transparenten Aufruf von Funktionen auf entfernten Rechnern
    - ➤ Basis von NFS (Network File System) und NIS (Network Information Service)
      - ➥ extrem sicherheitskritische Funktionalität!
- DES- oder Kerberos-basierte Authentifikation
- DES-Authentifikation (frühere SUN-Implementierungen)
  - ➤ Schlüsselaustausch initial mit Diffie-Hellman-Verfahren (192-Bit Schlüssel) - privater Schlüssel in NIS-Datenbank, mit Benutzer-PW verschlüsselt
  - ➤ Reduktion auf 56 Bit, damit Verschlüsselung eines Sitzungs-Schlüssels
  - ➤ Authentifikation des Clients über Kenntnis des Sitzungsschlüssels und Zeitstempel mit Gültigkeitsdauer
  - ➤ Probleme: Client erzeugt Sitzungs-Schlüssel, 56-BIT DES, Benutzer-PW

3 NIS

- Basis für Verwaltung zenraler Systeminformationen in einem UNIX-Netzwerk
  - ➤ Login-Information und Passwörter
  - ➤ Schlüssel für Secure RPC
  - ➤ Zuordnung Dienste → Portnummern
  - ➤ ...
- Login-Authentifikation
  - ➤ lokale Eingabe des Passwortes
  - ➤ Berechnung des Hashes
  - ➤ Anfordern der kompletten login-Daten vom NIS-Server (per RPC, unverschlüsselt)
  - ➤ lokaler Vergleich der Hash-Werte
  - ◆ weit verbreitet, sehr unsicher!

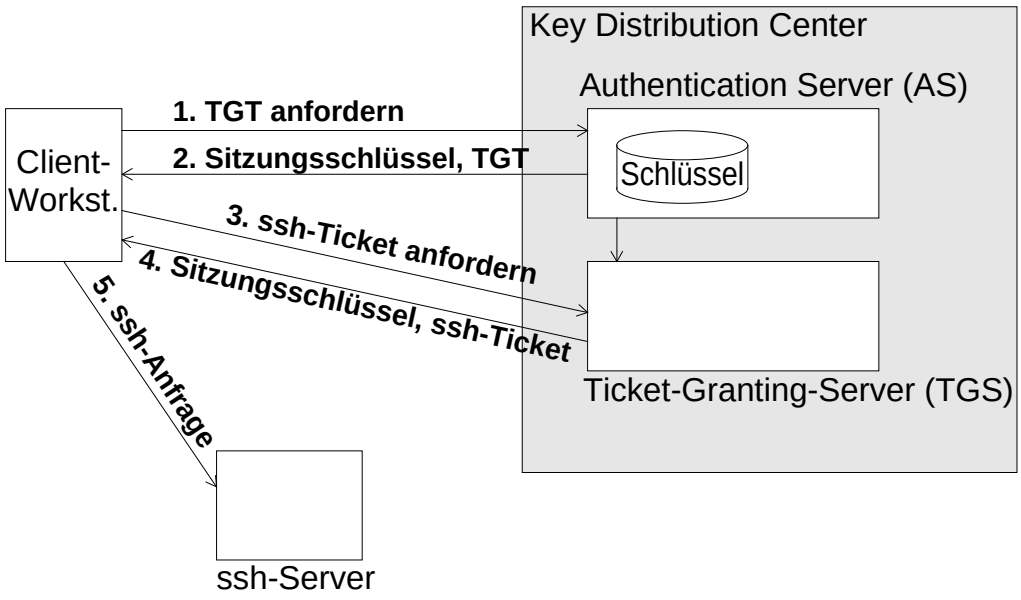
4 Kerberos

- siehe auch Kap. 7
- Athena-Projekt, MIT, ab 1983
- Basis für verteilte Authentifikation in Workstation-Netzen
  - ➤ UNIX, Windows, MacOS und Public-Domain-Implementierungen
  - ➤ Authentifikation gegenüber Diensten (ssh, http, NFS, ...)
  - ➤ nur Authentifikation des Benutzernamens - keine Abbildung auf systeminterne Identifikatoren (Autorisierungsdaten = uid, gid)
    - → Abbildung muss auf jedem Rechner lokal erfolgen
  - ➤ proprietäre Erweiterung bei Windows (Tickets um Autorisierungsdaten erweitert) schränkt Nutzung auf Windows-Kerberos-Server ein
- Basiskonzepte
  - ➤ Schlüsselaustausch auf Basis von Needham-Schroeder-Protokoll für symmetrische Verschlüsselung, erweitert um Zeitstempel (DS81)
  - ➤ verschiedene symm. Verschlüsselungsverfahren (AES256, AES128, DES-CBC-MD5, ...) und Hashverfahren (HMAC-SHA1-1996, MD5, ...)

4 Kerberos (2)

- Authentifikation von ***Principals***
  - ➤ Benutzer, Rechner, Dienste (ssh, http, ...)
- Vertrauenswürdiger zentraler Server (auch replizierbar)
  - ➤ Datenbank mit den Schlüsseln der Principals
    - – Schlüssel werden aus Passwörtern (bei Benutzern) erzeugt oder sind registrierte Zufallszahlen (bei anderen Diensten oder Rechnern)
    - – mit zentralem Systemschlüssel verschlüsselt
    - – Systemschlüssel ist immer im Hauptspeicher vorhanden (meist auch auf der Platte)
  - ➤ extrem sicherheitskritisch
  - ➥ keine anderen Dienste auf diesem Server, nur lokaler Zugriff, physikalisch gut sichern

4 Kerberos — Protokollablauf (wdh.)



4 Kerberos — Protokollablauf (wdh.)

(1) Benutzer `Joe` meldet sich an Client-WS an und fordert initiales Ticket an *(TGT: Ticket Granting Ticket)* (t_1 = Timestamp)

C → AS: (Joe, TGS, t_1)

(2) - wenn Benutzer `Joe` existiert, wird TGT erzeugt und mit Schlüssel des TGS verschlüsselt: $TGT=\{T_{Joe,TGS}\}^{K_{TGS}}$
- Aubau eines Tickets $T_{A,B}=(A, B, addr_A, t, lifetime, K_{A,B})$
- für die Kommunikation mit dem TGS erhält `Joe` außerdem einen Sitzungsschlüssel $K_{Joe,TGS}$

AS → C: ($\{K_{Joe,TGS}, t_1\}^{K_{Joe}}$,TGT)

- K_{Joe} ist aus dem Passwort des Benutzers ableitbar
  → Sitzungsschlüssel kann auf der Client-WS entschlüsselt werden

(3) - zur Nutzung eines Servers (z. B. ssh) muss `Joe` ein Dienst-spezifisches Ticket (ssh-Ticket) vom TGS anfordern
- für die Kommunikation mit dem ssh-Server wird ein Authentifikator erzeugt: $A_{Joe,ssh} = (ssh, addr_{Client\text{-}WS}, t_A)$

C → TGS: ($\{A_{Joe,ssh}\}^{K_{Joe,TGS}}$,TGT, ssh, t_2)

4 Kerberos — Protokollablauf (wdh.)

(4) - TGS prüft zur Authentifikation Name und addr in Authentifikator und TGT sowie die Aktualität des Zeitstempels t_A im Authentifikator
- falls ok wird ein Sitzungsschlüssel $K_{Joe,ssh}$ erzeugt und mit dem ssh-Session-Ticket $SesT=\{T_{Joe,ssh}\}^{K_{ssh}}$ an C geschickt

TGS → C: $(\{K_{Joe,ssh}, t_2\}^{K_{Joe,TGS}}, SesT)$

(5) - Nachrichten von Joe (Client-WS) an den ssh-Server können nun mit dem Authentifikator und ssh-Ticket authentifiziert werden

C → SSH: $(\{A_{Joe,ssh}\}^{K_{Joe,ssh}}, SesT)$

- $K_{Joe,ssh}$ ist in $Tssh$ enthalten
- wechselseitige Authentifikation ist möglich, wenn der ssh-Server $\{t_2+1\}^{K_{Joe,ssh}}$ an Joe zurückschickt

- Abbildung auf uid, gids unter UNIX erfolgt beim Server durch
  - ➤ lokale Passwortdatei, NIS-Abfrage oder LDAP-Abfrage
  - ➤ NIS-Abfrage nur bei Verw. von Secure-RPC sicher!

C.4 Literatur

DS81. D.E. Denning and G.M. Sacco. Timestamps in Key Distribution Protocols. *Communications of the ACM*, 24(8):553, 1981.

NT97. C. Neumann and T. Ts'o. The Evolution of the Kerberos Authentication System. TechReport, MIT. (ftp://athena-dist.mit.edu/pub/kerberos/doc/krb_evol.PS)