Linux Server Sicherheit - Einführung

Agenda

1. Grundlagen der Computersicherheit
2. Kryptografie fuer Linux-Systemadministratoren

Grundlagen der Computersicherheit

Sicherheitseinstellungen von Linux-Systemen sollten sich an der Gefährdung des Systems orientieren

- wer ist der Angreifer
- was sind die Ziele von Angreifer
- was kann angegriffen werden (Daten, Ressourcen, Personen, Infrastruktur ...)

Lebensdauer von Systemen

Bei der Planung von Sicherheitssystemen muss die Lebensdauer dieser Systeme berücksichtigt werden

- sind heute sichere System auch in 5,10,15 Jahren noch sicher?
- die Lebenszeit von Computersystemen verlängert sich
  - embedded Systeme (IoT)
  - Virtualisierung (Hardware wird erneuert, die VMs werden umgezogen)
  - Betriebssysteme mit langer Wartung (Red Hat Enterprise Linux 10+ Jahre)

Komplexität vs. Sicherheit

Unix-Systeme sind komplex. Spezielle Sicherheitseinstellungen entfernen das System von der vom Hersteller gelieferten Grund-Konfiguration.

Die Komplexität des Systems und der Systemadministration erhöht sich.

Spezielle Sicherheitseinstellungen sollten automatisiert eingerichtet und (regelmäßig) getestet werden. 'Configuration Orchestration' Werkzeuge können dabei helfen:

- Ansible
- SaltStack
- Puppet
- Chef
- cfengine

Dabei sollte das Werkzeug keine neuen Sicherheitslücken erzeugen!

Kryptographie für Linux-Systemadministratoren

Kryptographie für Systemadministratoren

Eine kleine Einführung in Begriffe und Kryptographische Algorithmen,
welche bei der Arbeit als Systemadmin vorkommen

1. Kryptographische Hashes (Fingerprints)
2. Hashed Message Authentication Codes (HMAC)
3. Symmetrische Verschlüsselung
4. Asymmetrische Verschlüsselung
5. Digitale Signaturen
6. Benutzung von OpenSSL
7. Benutzung von GnuPrivacyGuard

Funktionen von Kryptographie

1. Authentisierung - die Antwort auf die Frage "wer?"
2. Integrität - die Antwort auf die Frage "wurden Daten geändert?"
3. Geheimhaltung - Daten Verschlüsseln so das nur der/die Empfaenger die Daten lesen kann
4. Verbindlichkeit/Nichtabstreitbarkeit - "Sage nicht Du hast es nicht geschrieben"

Kryptographische Hashes

Kryptographische Hashes sind spezielle mathematische Funktionen, welche aus beliebigen Eingabe-Daten einen digitalen Fingerabdruck fester Länge erstellen.

Kryptografische Hash-Funktionen haben dabei vier Hauptmerkmale:

- die Berechnung des Hash-Wertes sollte einfach (nicht rechenintensiv) sein
- Die Hash-Funktion darf nicht umkehrbar sein, d.h. es darf nicht möglich sein nur aus dem Hash-Wert die ursprünglichen Daten wiederherzustellen
- Bei leichten Änderungen in den Eingabewerten muss sich der Hash komplett ändern
- Es muss unmöglich sein, zwei Eingabewerte zu finden welche in den gleichen Hash-Wert resultieren

Kryptographische Hash-Funktionen werden auch *Message Digest* genannt.

Bekannte kryptographische Hash-Funktionen

- MD5 (unsicher)
- RIPEMD160
- SHA1 (nicht mehr empfohlen)
- SHA256
- SHA512
- SHA3 (Keccak)

Benutzung von kryptografischen Hash-Funktionen

- Integritätsprüfung von Dateien und Nachrichten (z.B. nach Download)
- Password-Prüfung (Datei /etc/shadow)
- 'Proof-of-Work', Beweis das der Anfrager eine Arbeit geleistet hat (z.B. Bitcoin Mining)
- Datei/Versions-ID (Beispiel 'git')

Kryptografische Hash-Funktionen haben keine Authentisierungs-Funktion. Das prüfen von MD5-, SHA1- oder andern Hash-Werten nach dem Download einer Datei über ungesicherte Netzwerke gibt keine Auskunft über die Quelle der Daten!

Beispiele der Benutzung von kryptografischen Hash-Funktionen

```
# echo "Hallo" | md5sum
3290ec3c19a8a39362f7d70043f15627  -

# echo "Hallo" | sha1sum
6ce3fe1479a10edb2f1bdd6d181a5b0e210abe1a  -

# echo "Hallo" | openssl dgst -sha1
(stdin)= 6ce3fe1479a10edb2f1bdd6d181a5b0e210abe1a

# echo "Hallo" | openssl dgst -sha256
(stdin)= 5891b5b522d5df086d0ff0b110fbd9d21bb4fc7163af34d08286a2e846f6be03

# echo "Hello" | openssl dgst -sha256
(stdin)= 66a045b452102c59d840ec097d59d9467e13a3f34f6494e539ffd32c1bb35f18
```

HMAC - Hash-based Message Authentication Codes

HMAC-Funktionen verbinden eine kryptografische Hash-Funktion mit einem geteilten Geheimnis (pre shared secret, PSK).

Ein HMAC liefert Authentisierung (die Daten kommen von einem Ersteller des HMAC mit dem PSK) und Integrität (die Daten wurden seit Erstellung des HMAC nicht verändert).

HMACs sind symmetrische kryptografische Signaturen.

HMAC-Funktionen werden z.B. SSH und DNS (TSIG) benutzt.

HMAC Algorithmen

- HMAC-MD5 (siehe RFC 6151 "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms")
- HMAC-SHA1
- HMAC-SHA256
- HMAC-SHA512
- HMAC-RIPEMD160

Weitere MAC-Algorithmen

- UMAC-32/64/96/128 (OpenSSH)
- SKEIN-256/512 (SHA3-Finalist)
- Keccak/SHA3 (SHA3-Gewinner)
- Poly1305-AES (RFC 7539, TLS, OpenSSH)

Beispiele: HMAC-SHA1 mit OpenSSL

```
echo -n "value" | openssl sha1 -hmac "key"
```

Symmetrische Verschlüsselung

Bei symmetrischer Verschlüsselung kommt sowohl beim Verschlüsseln als auch beim Entschlüsseln der identische Schlüssel zur Anwendung (Preshared Secret Key, PSK).

Symmetrische Verschlüsselung ist in der Regel um ein vielfaches schneller als asymmetrische Verschlüsselung

Symmetrische Verschlüsselungs-Algorithmen

- DES (veraltet, unsicher)
- 3DES (unsicher)
- AES-128/256 (Rijndael)
- Twofish (AES Finalist, OpenSSH)
- Serpent (AES Finalist)
- RC4 (unsicher)
- Salsa20/ChaCha20 (RFC 7539, benutzt in OpenSSL/Android/Chrome/OpenSSH)

Beispiel: symmetrische Verschlüsselung mit OpenSSL

```
# RC4 (schnell, aber ggf. nicht sicher genug)
echo "Hallo LinuxHotel" | openssl enc -e -rc4 -a
enter rc4 encryption password:
Verifying - enter rc4 encryption password:
U2FsdGVkX1+/19xWWHObEzgY8tNP8MCQUrqOO83ylzzy

echo "U2FsdGVkX1+/19xWWHObEzgY8tNP8MCQUrqOO83ylzzy" |  openssl enc -d -rc4 -a

# AES
% echo "Hallo LinuxHotel" | openssl enc -e -aes256 -a
enter aes-256-cbc encryption password:
Verifying - enter aes-256-cbc encryption password:
U2FsdGVkX186OZf2oJrGRtSOHtSusHEZSapzohhY2JtlbDAdBXSYUlAVFX1UFtnM

echo "U2FsdGVkX186OZf2oJrGRtSOHtSusHEZSapzohhY2JtlbDAdBXSYUlAVFX1UFtnM" |  openssl enc -d -a
```

Asymmetrische Verschlüsselung

Bei der asymmetrischen Verschlüsselung kommen Schlüsselpaare, bestehend aus öffendlichen und privaten Schlüsseln zum Einsatz.

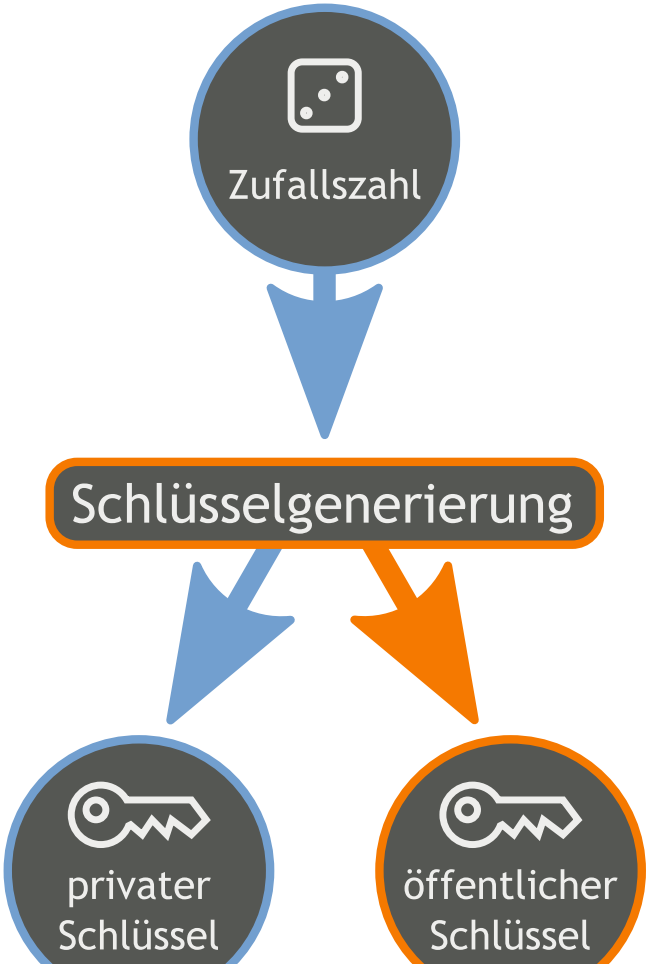
Daten, welche mit dem privaten Schlüssel verschlüsselt wurden, können nur mit dem öffentlichen Schlüssel wieder entschlüsselt werden (Digitale Signatur).

Daten, welche mit dem öffendlichen Schlüssel verschlüsselt wurden, koennen nur mit dem privaten Schluessel entschlüsselt werden.

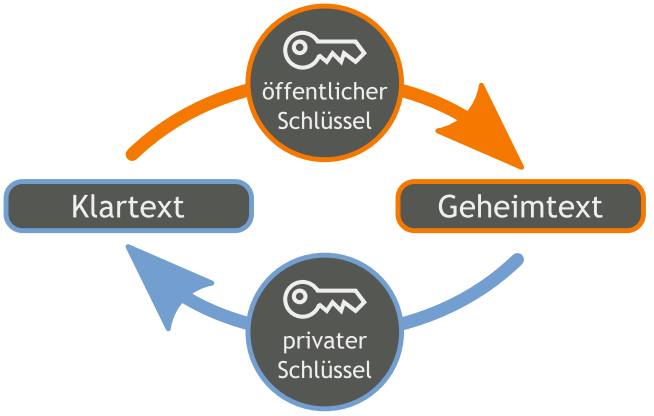
Asymmetrische Verschlüsselungsalgorithmen

- RSA
- Elliptische Kurven
- Elgamal
- McEliece (Post-Quantum-Safe)

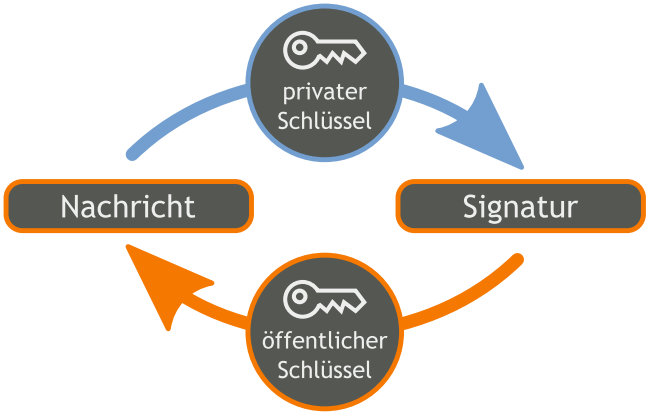
Asymmetrische Verschlüsselung



Asymmetrische Verschlüsselung



Asymmetrische Digitale Signatur



Beispiel asymmetrische Verschlüsselung mit OpenSSL

```
# einen neuen privaten RSA Schlüssel erstellen
openssl genpkey -algorithm RSA -out key.pem
# den privaten Schlüssel mit AES 256bit verschlüsseln
openssl pkey -in key.pem -aes256 -out keyaes.pem
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
rm key.pem
# den öffentlichen Schlüssel erzeugen
openssl pkey -in keyaes.pem -pubout -out pubkey.pem
# Daten mit dem öffentlichen Schlüssel verschlüsseln
echo "Hallo LinuxHotel" > plain.txt
openssl pkeyutl -in plain.txt -out cipher.txt -encrypt -pubin -inkey pubkey.pem
cat cipher.txt | od -x -a
# Daten mit dem privaten Schlüssel entschlüsseln
openssl pkeyutl -in cipher.txt -out plain2.txt -decrypt -inkey keyaes.pem
Enter pass phrase for keyaes.pem:
cat plain2.txt
Hallo LinuxHotel
```

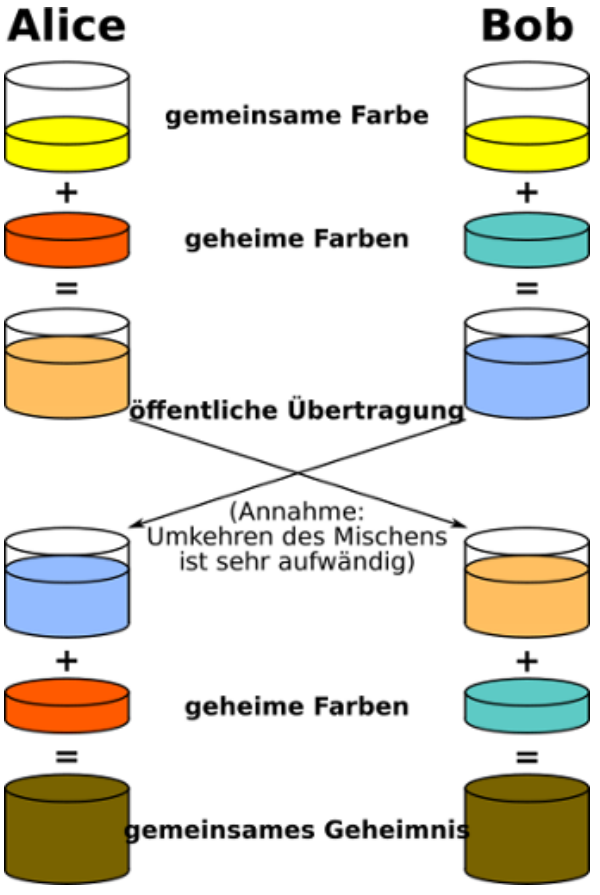
Schlüsselaustauschprotokolle

In verschlüsselten Kommunikationsverbindungen muessen die Kommunikationsparter oft einen gemeinsamen, geheimen Schluessel aushandeln. Das Diffie-Hellman-Verfahren erlaubt es, einen gemeinsamen geheimen Schluessel zu errechnen, ohne das ein Angreifer mit Zugriff auf die Kommunikation diesen Schluessel auch errechnen kann.

- Diffie-Hellman-Schlüsselaustausch
- Ephemeral Diffie-Hellman (Forward-Secrecy)
- Elliptic Curve Diffie-Hellman (ECDH)

Schlechte DH-Parameter vermeiden: https://weakdh.org/

Diffie-Hellmann-Schlüsselaustausch



Zufallszahlen und Linux

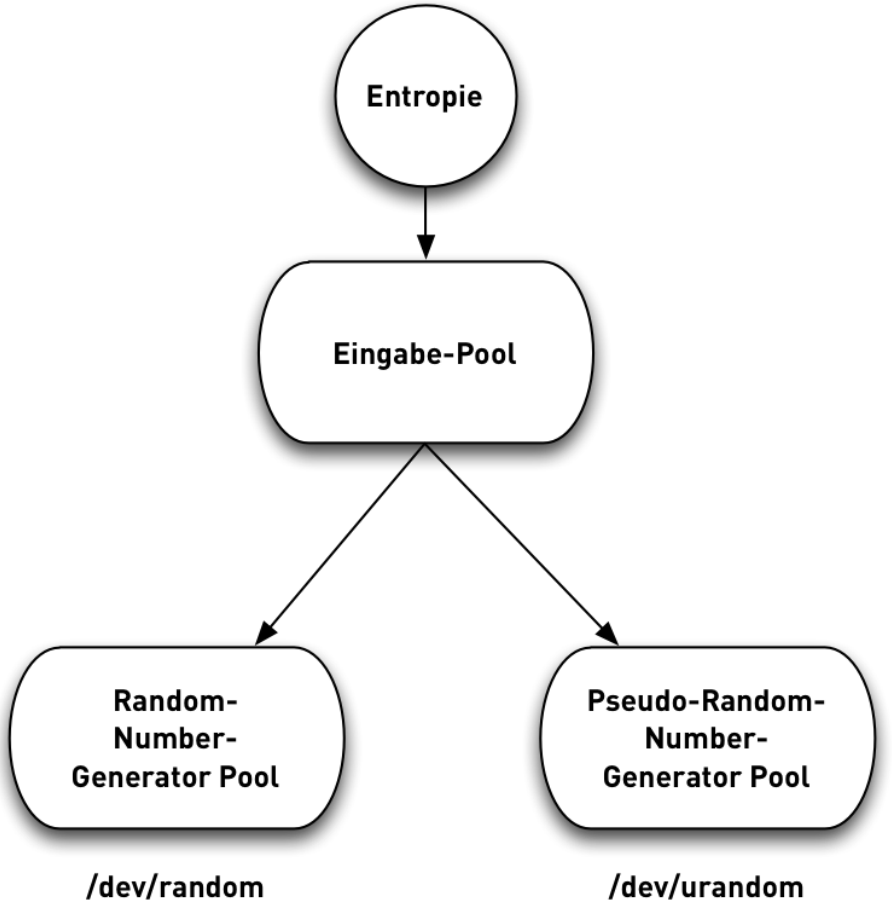
Einige kryptografische Protokolle benötigen qualitativ hochwertige (== statistisch zufällige) Zufallszahlen.

Die Erzeugung von echtem Zufall in einem Computersystem ist schwierig.

Der Linux-Kernel bietet zwei Geräte-Schnittstellen als Quelle für Zufallszahlen:

- `/dev/random` - liefert echte Zufallsereignisse (z.B. basierend auf Tastatur-Anschlägen des Benutzers, Interrupts der Netzwerkkarte etc). Dies ist echte Entropie. Ist nicht genug Entropie im System vorhanden, so blockiert `/dev/ramdom` und somit jeder Prozess, welcher aus diesem Gerät Zufallsdaten liesst
- `/dev/urandom` - liefert Zufall auf Basis eines Software Pseudo-Random-Number-Generator (PRNG)

Zufallszahlen und Linux



Zufallszahlen und Linux

In virtuellen Maschinen, aber auch auf echter Hardware kann es vorkommen, das nicht genügend Entropie (Zufall) im Linux-System vorhanden ist.

Dies zeigt sich, das Programme mit kryptografischen Funktionen (z.B. RSA-Schlüsselerzeugung) sehr lange laufen.

In solchen Situationen kann ein Entropie-Gathering-Dämon (EGD) helfen. Ein bekannter EGD für Mehr-Kern-CPU-Systeme is `haveged`.

```
yum install haveged
systemctl enable haveged
systemctl start haveged
```

Pseudo-Zufallszahlen per OpenSSL erzeugen

Der nachfolgende Befehl gibt 10 Pseudo-Zufallszahlen im hexadezimaler Format aus. Der Pseudo-Zufallszahlen-Genertator wird hierbei aus der Datei `~/.rnd` und `/dev/urandom` initialisiert ("seeding"):

```
openssl rand -rand /dev/urandom -hex 10
% ls -l ~/.rnd
-rw-------. 1 nutzer nutzer 1024 Nov 22 22:12 /home/cas/.rnd
```

OpenSSL legt in der Datei `~/.rnd` Entropie für spätere Aufrufe ab.

Nächstes Kapitel: Benutzerverwaltung, Anmeldung, Passwörter, PAM