“Die einzige Konstante ist die Veränderung.”

Heraclitus

OLD DOG NEW TRICKS

THINGS YOU MIGHT NOT KNOW ABOUT OPENSSH

Created by Marcus Rückert or just darix ;)

ÜBER MICH

- SUSE Nutzer seit 5.1
- Bei SUSE seit 2005-06
- Packager, Reviewer, Admin
- SUSE und openSUSE Infrastruktur

KONVENTIONEN

Alle Optionen sind jeweils eine Zeile. Bei einem Beispiel wie dem folgendem, einfach alle Backslash und nachfolgenden Zeilenmbrüche entfernen. Aus ...

```
# /etc/ssh/ssh_config || ~/.ssh/config
SomeOption somevalue, \
  anothervalue
```

... wird ...

```
# /etc/ssh/ssh_config || ~/.ssh/config
SomeOption somevalue,anothervalue
```

Kommentare am Anfang der Codeblöcke zeigen in welcher Datei die Optionen benutzt werden koennen, Im Beispiel kann die Option entweder in `/etc/ssh/ssh_config` oder `~/.ssh/config` benutzt werden.

DER SSH CLIENT HAT EINE KONFIGURATIONSDATEI?

UNSER PROBLEM

Für die openSUSE VMs benutzen wir SSH Weiterleitungen mit xinetd. Damit haben wir aber viele SSH Verbindungen auf nicht Standardports wie:

```
ssh -p 1234 root@proxyhost.example.com
```

Besonders nervig ist dass `scp` für das selbe eine andere Option benutzt:

```
scp -P 1234 somefile root@proxyhost.example.com:
```

DAS MUSS EINFACHER GEHEN

```
# ~/.ssh/config
Host myvm
    Port 1234
    User root
    Hostname proxyhost.example.com
```

```
ssh myvm
```

```
man ssh_config
```

ADVANCING CRYPTO

DIE ZEITEN ÄNDERN SICH

Punkte, die wir vielleicht überdenken sollten

- 1024 bit keys
- DSA
- NIST Kurven
- SHA1
- MD5
- RC4

Basierend auf:
"Secure Secure Shell" by @stribika

KEY EXCHANGE PROTOCOLS

1. curve25519-sha256: ECDH over Curve25519 with SHA2
2. diffie-hellman-group1-sha1: 1024 bit DH with SHA1
3. diffie-hellman-group14-sha1: 2048 bit DH with SHA1
4. diffie-hellman-group-exchange-sha1: Custom DH with SHA1
5. diffie-hellman-group-exchange-sha256: Custom DH with SHA2
6. ecdh-sha2-nistp256: ECDH over NIST P-256 with SHA2
7. ecdh-sha2-nistp384: ECDH over NIST P-384 with SHA2
8. ecdh-sha2-nistp521: ECDH over NIST P-521 with SHA2

KEY EXCHANGE PROTOCOLS

ECDH KURVEN

1. curve25519-sha256: ECDH over Curve25519 with SHA2
2. diffie-hellman-group1-sha1: 1024 bit DH with SHA1
3. diffie-hellman-group14-sha1: 2048 bit DH with SHA1
4. diffie-hellman-group-exchange-sha1: Custom DH with SHA1
5. diffie-hellman-group-exchange-sha256: Custom DH with SHA2
6. ecdh-sha2-nistp256: ECDH over NIST P-256 with SHA2
7. ecdh-sha2-nistp384: ECDH over NIST P-384 with SHA2
8. ecdh-sha2-nistp521: ECDH over NIST P-521 with SHA2

KEY EXCHANGE PROTOCOLS

KÖNNEN ES NOCH EIN PAAR BIT MEHR SEIN?

1. curve25519-sha256: ECDH over Curve25519 with SHA2
2. diffie-hellman-group1-sha1: 1024 bit DH with SHA1
3. diffie-hellman-group14-sha1: 2048 bit DH with SHA1
4. diffie-hellman-group-exchange-sha1: Custom DH with SHA1
5. diffie-hellman-group-exchange-sha256: Custom DH with SHA2
6. ecdh-sha2-nistp256: ECDH over NIST P-256 with SHA2
7. ecdh-sha2-nistp384: ECDH over NIST P-384 with SHA2
8. ecdh-sha2-nistp521: ECDH over NIST P-521 with SHA2

KEY EXCHANGE PROTOCOLS

GUTER HASH? SCHLECHTER HASH?

1. curve25519-sha256: ECDH over Curve25519 with SHA2
2. diffie-hellman-group1-sha1: 1024 bit DH with SHA1
3. diffie-hellman-group14-sha1: 2048 bit DH with SHA1
4. diffie-hellman-group-exchange-sha1: Custom DH with SHA1
5. diffie-hellman-group-exchange-sha256: Custom DH with SHA2
6. ecdh-sha2-nistp256: ECDH over NIST P-256 with SHA2
7. ecdh-sha2-nistp384: ECDH over NIST P-384 with SHA2
8. ecdh-sha2-nistp521: ECDH over NIST P-521 with SHA2

KEY EXCHANGE PROTOCOLS - FAZIT

Am Ende bleiben uns curve25519-sha256 und diffie-hellman-group-exchange-sha256.

KEY EXCHANGE PROTOCOLS - EMPFOHLENE KONFIGURATION

```
# /etc/ssh/sshd_config
KexAlgorithms curve25519-sha256@libssh.org, \
  diffie-hellman-group-exchange-sha256
```

```
# /etc/ssh/ssh_config || ~/.ssh/config
Host *
     KexAlgorithms curve25519-sha256@libssh.org, \
       diffie-hellman-group-exchange-sha256
```

AUTHENTIFIZIERUNG

AUF DEM SERVER

Kein SSH1 mehr. Keine DSA Schlüssel. Falls noch nicht geschehen ED25519 anschalten.

```
# /etc/ssh/sshd_config
Protocol 2
HostKey /etc/ssh/ssh_host_ed25519_key
HostKey /etc/ssh/ssh_host_rsa_key
PubkeyAuthentication yes
PasswordAuthentication no
ChallengeResponseAuthentication no
```

```
# /etc/ssh/sshd_config
AuthenticationMethods publickey,password publickey,keyboard-interactive
```

AUTHENTIFIZIERUNG

AUF DEM CLIENT

```
# /etc/ssh/ssh_config || ~/.ssh/config
Host *
    PasswordAuthentication no
    ChallengeResponseAuthentication no
    PubkeyAuthentication yes
    HostKeyAlgorithms ssh-ed25519-cert-v01@openssh.com, \
      ssh-rsa-cert-v01@openssh.com, \
      ssh-rsa-cert-v00@openssh.com, \
      ssh-ed25519,ssh-rsa
```

SYMMETRIC CIPHERS

3des-cbc aes128-cbc aes192-cbc aes256-cbc aes128-ctr aes192-ctr aes256-ctr aes128-gcm@openssh.com aes256-gcm@openssh.com arcfour arcfour128 arcfour256 blowfish-cbc cast128-cbc chacha20-poly1305@openssh.com

SYMMETRIC CIPHERS

SICHERHEIT

3des-cbc aes128-cbc aes192-cbc aes256-cbc aes128-ctr aes192-ctr aes256-ctr aes128-gcm@openssh.com aes256-gcm@openssh.com arcfour arcfour128 arcfour256 blowfish-cbc cast128-cbc chacha20-poly1305@openssh.com

SYMMETRIC CIPHERS

SCHLÜSSELLÄNGE UND BLOCKGRÖSSE

3des-cbc aes128-cbc aes192-cbc aes256-cbc aes128-ctr aes192-ctr aes256-ctr aes128-gcm@openssh.com aes256-gcm@openssh.com arcfour arcfour128 arcfour256 blowfish-cbc cast128-cbc chacha20-poly1305@openssh.com

SYMMETRIC CIPHERS

CIPHER MODES

3des-cbc aes128-cbc aes192-cbc aes256-cbc aes128-ctr aes192-ctr aes256-ctr aes128-gcm@openssh.com aes256-gcm@openssh.com arcfour arcfour128 arcfour256 blowfish-cbc cast128-cbc chacha20-poly1305@openssh.com

SYMMETRIC CIPHERS - FAZIT

```
# /etc/ssh/sshd_config
Ciphers chacha20-poly1305@openssh.com, \
  aes256-gcm@openssh.com, \
  aes128-gcm@openssh.com, \
  aes256-ctr,aes192-ctr,aes128-ctr
```

```
# /etc/ssh/ssh_config || ~/.ssh/config
Host *
    Ciphers chacha20-poly1305@openssh.com, \
      aes256-gcm@openssh.com, \
      aes128-gcm@openssh.com, \
      aes256-ctr,aes192-ctr,aes128-ctr
```

MESSAGE AUTHENTICATION CODES

- *Encrypt-then-MAC*: encrypt the message, then attach the MAC of the ciphertext.
- *MAC-then-encrypt*: attach the MAC of the plaintext, then encrypt everything. (TLS)
- *Encrypt-and-MAC*: encrypt the message, then attach the MAC of the plaintext. (SSH)

AVAILABLE MESSAGE AUTHENTICATION CODES

hmac-md5 hmac-md5-96 hmac-ripemd160 hmac-sha1 hmac-sha1-96 hmac-sha2-256 hmac-sha2-512 umac-64 umac-128 hmac-md5-etm@openssh.com hmac-md5-96-etm@openssh.com hmac-ripemd160-etm@openssh.com hmac-sha1-etm@openssh.com hmac-sha1-96-etm@openssh.com hmac-sha2-256-etm@openssh.com hmac-sha2-512-etm@openssh.com umac-64-etm@openssh.com umac-128-etm@openssh.com

MESSAGE AUTHENTICATION CODES

SICHERHEIT DER HASHFUNKTION

hmac-md5 hmac-md5-96 hmac-ripemd160 hmac-sha1 hmac-sha1-96 hmac-sha2-256 hmac-sha2-512 umac-64 umac-128 hmac-md5-etm@openssh.com hmac-md5-96-etm@openssh.com hmac-ripemd160-etm@openssh.com hmac-sha1-etm@openssh.com hmac-sha1-96-etm@openssh.com hmac-sha2-256-etm@openssh.com hmac-sha2-512-etm@openssh.com umac-64-etm@openssh.com umac-128-etm@openssh.com

MESSAGE AUTHENTICATION CODES

SCHLÜSSELLÄNGE UND GRÖSSE DES TAGS

hmac-md5 hmac-md5-96 hmac-ripemd160 hmac-sha1 hmac-sha1-96 hmac-sha2-256 hmac-sha2-512 umac-64 umac-128 hmac-md5-etm@openssh.com hmac-md5-96-etm@openssh.com hmac-ripemd160-etm@openssh.com hmac-sha1-etm@openssh.com hmac-sha1-96-etm@openssh.com hmac-sha2-256-etm@openssh.com hmac-sha2-512-etm@openssh.com umac-64-etm@openssh.com umac-128-etm@openssh.com

MESSAGE AUTHENTICATION CODES - FAZIT

```
# /etc/ssh/sshd_config
MACs hmac-sha2-512-etm@openssh.com, \
  hmac-sha2-256-etm@openssh.com, \
  hmac-ripemd160-etm@openssh.com, \
  umac-128-etm@openssh.com, \
  hmac-sha2-512,hmac-sha2-256, \
  hmac-ripemd160,umac-128@openssh.com
```

```
# /etc/ssh/ssh_config || ~/.ssh/config
Host *
  MACs hmac-sha2-512-etm@openssh.com, \
    hmac-sha2-256-etm@openssh.com, \
    hmac-ripemd160-etm@openssh.com, \
    umac-128-etm@openssh.com, \
    hmac-sha2-512,hmac-sha2-256, \
    hmac-ripemd160,umac-128@openssh.com
```

DIE KOMPLETTE KONFIGURATION FÜR DEN CLIENT (OPENSSL >= 1)

```
# /etc/ssh/ssh_config || ~/.ssh/config
Host *
   KexAlgorithms curve25519-sha256@libssh.org,diffie-hellman-group-excha
   HostKeyAlgorithms ssh-ed25519-cert-v01@openssh.com,ssh-rsa-cert-v01@o
   Ciphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-g
   MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac
   PubkeyAuthentication yes
   PasswordAuthentication no
   ChallengeResponseAuthentication no
```

DIE KOMPLETTE KONFIGURATION FÜR DEN CLIENT (OPENSSL < 1)

```
# /etc/ssh/ssh_config || ~/.ssh/config
Host *
   KexAlgorithms diffie-hellman-group-exchange-sha256
   HostKeyAlgorithms ssh-rsa-cert-v01@openssh.com,ssh-rsa-cert-v00@opens
   Ciphers aes256-gcm@openssh.com,aes128-gcm@openssh.com,aes256-ctr,aes1
   MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac
   PubkeyAuthentication yes
   PasswordAuthentication no
   ChallengeResponseAuthentication no
```

DIE KOMPLETTE KONFIGURATION FÜR DEN SERVER (OPENSSL >= 1)

```
# /etc/ssh/sshd_config
Protocol 2
HostKey /etc/ssh/ssh_host_ed25519_key
HostKey /etc/ssh/ssh_host_rsa_key
KexAlgorithms curve25519-sha256@libssh.org,diffie-hellman-group-exchange
Ciphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@
MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-ri

PubkeyAuthentication yes
PasswordAuthentication no
# reenable if you want to use OTP together with ssh keys
ChallengeResponseAuthentication no
```

DIE KOMPLETTE KONFIGURATION FÜR DEN SERVER (OPENSSL < 1)

```
# /etc/ssh/sshd_config
Protocol 2
HostKey /etc/ssh/ssh_host_rsa_key
KexAlgorithms diffie-hellman-group-exchange-sha256
Ciphers aes256-gcm@openssh.com,aes128-gcm@openssh.com,aes256-ctr,aes192-
MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-ri

PubkeyAuthentication yes
PasswordAuthentication no
# reenable if you want to use OTP together with ssh keys
ChallengeResponseAuthentication no
```

KONTROLLE IST BESSER

```
$ ssh -v user@hostname /bin/true
[snip]
debug1: kex: server->client chacha20-poly1305@openssh.com <implicit> nor
debug1: kex: client->server chacha20-poly1305@openssh.com <implicit> nor
[snip]
debug1: Server host key: ED25519 cf:c0:.... [MD5]
[snip]
```

CASUS GITHUB

NOCH MAL KURZ KEY EXCHANGE

```
# Github needs diffie-hellman-group-exchange-sha1
# some of the time but not always.
Host github.com
    KexAlgorithms curve25519-sha256@libssh.org, \
       diffie-hellman-group-exchange-sha256, \
       diffie-hellman-group-exchange-sha1, \
       diffie-hellman-group14-sha1
```

PUBLIC KEYS SIND FÜR ALLE

https://github.com/username.keys

- Auditing GitHub users' SSH key quality
- Batch-GCDing Github SSH Keys

SSH SCHLÜSSEL ERZEUGEN

```
ssh-keygen -t ed25519 -o -a 100
ssh-keygen -t rsa -b 4096 -o -a 100
```

"`-o -a 100`" (openssh >= 6.5)

SIGNING

Basierend auf:
"OpenSSH certificates" by Thomas Habets

Benötigt mindestens openssh >= 5.5 oder noch besser >= 6.0

ERSTMAL DIE SERVER

ICH KENN DICH ZWAR NICHT ABER ICH VERTRAU DIR MAL

- Benötigt manuelle Überprüfung der Schlüssel/Fingerprints
- Mit der Zeit sammelt man eine große Anzahl von Schlüsseln in `~/.ssh/known_hosts`
- Und falls sich mal die Server- oder Clientkonfiguration ändert werden es noch mehr

HUH EIN NEUER SSH KEY?

- Jeder hat schon mal vergessen den SSH Schlüssel zu sichern vor der Neuinstallation
- Manchmal will mal sie auch absichtlich austauschen
- Nutzer müssen `~/.ssh/known_hosts` manuell pflegen, da nur ein Teil von "`ssh-keygen -R`" abgedeckt ist
- D.h. wir haben mindestens 2 fehlgeschlagene Verbindungen bei einem ausgetauschten Schlüssel

SCHLÜSSEL WECHSEL DICH

- Falls man immer dran gedacht hat die Schlüssel zu sichern, genügen die vll nicht mehr aktuellen Standards.
- SSL Schlüssel wechseln alle 1-3 Jahre
- Key Rotation Support in 6.8+

DAS WECHSELN VON SSH SERVER KEYS SOLLTE NICHT NERVIG SEIN

DIE NUTZERSEITE - SETUP

```
# ~/.ssh/known_hosts
@cert-authority *.example.com ssh-rsa AAAAB3[...]== Comment
```

Immer noch ein etwas dass wir ausserhalb des SSH Protokols kommunizieren müssen - aber nur noch einmal

DIE NUTZERSEITE - DAS ERSTE MAL

```
darix@mylaptop:~ $ ssh unknownhost.example.com
darix@unknownhost:~ $
```

DIE NUTZERSEITE - LANGE NICHT GESEHEN

```
darix@mylaptop:~ $ ssh oldmachinenewkey.example.com
darix@oldmachinenewkey:~ $
```

SO SOLLTE ES IMMER SEIN ODER?

MAGIE? NICHT WIRKLICH

```
ssh-keygen -f host_ca
```

```
cat host_ca.pub
```

```
ssh-keygen -s host_ca -I 'host:unknownhost.example.com' -h \
  -n unknownhost.example.com -V +52w \
  unknownhost.example.com_ssh_host_rsa_key.pub
```

```
scp unknownhost.example.com_ssh_host_rsa_key-cert.pub \
  root@unknownhost.example.com:/etc/ssh/ssh_host_rsa_key-cert.pub
```

```
# /etc/ssh/sshd_config
HostCertificate /etc/ssh/ssh_host_rsa_key-cert.pub
```

Und das ganze für alle anderen Schlüsseltypen (DSA, ED25519)

SSH Server Konfiguration neuladen

UND WO SIND DIE NACHTEILE?

- Beim Verbinden muß der volle Hostname benutzt werden
- Zertifikate mit Ablaufzeit brauchen Monitoring/zeitnahen Austausch
- Hostnamen sind im Zertifikat verdrahtet
  - Neuer CNAME, Neues Zertifikat
  - Theoretisch kann man den Hostname auch weglassen (MITM-Gefahr)

SCHLÜSSELAUSTAUSCH - DER ANFANG

1. SSH Certificate Authority aufsetzen
2. Aktuelle Schlüssel einsammeln und signieren
3. Zertifikate auf alle Server ausrollen
4. Kommunikation des öffentlichen Schlüssels an alle Nutzer.

SCHLÜSSELAUSTAUSCH - JETZT RICHTIG

1. Neue signierte Schlüsselpaare für alle Server erstellen und ausrollen
2. Nutzersicht:

   Ein Nutzer der unseren Empfehlungen gefolgt ist?

   Merkt nix. Alles funktioniert weiter.

   Der Rest ...

   ... sieht ganz viele Warnungen wegen ausgetauschten Schlüsseln

CLIENTS?

~/.SSH/AUTHORIZED_KEYS

- Funktioniert im Prinzip
- Wie blocke ich den SSH Schlüssel eines ehemaligen Mitarbeiters?
- Wie verhindere ich dass ein Nutzer anderen Nutzern Zugang gibt?

WIEDER KEINE MAGIE NÖTIG

```
ssh-keygen -f user_ca
```

```
scp user_ca.pub root@somehost.example.com:/etc/ssh/user_ca.pub
```

```
# /etc/ssh/sshd_config
TrustedUserCAKeys /etc/ssh/user_ca.pub
```

SSH Server Konfiguration neuladen

```
ssh-keygen -s user_ca -I user_darix \
  -n darix,root -V +52w darix_id_rsa.pub
```

Im Zweifel das ganze noch für die anderen Schlüsseltypen (DSA, ED25519)

```
scp darix_id_*-cert.pub darix@mylaptop.example.com:.ssh/
```

WAS ÄNDERT SICH?

```
ssh -i ~/.ssh/id_25519.pub somehost.example.com
```

```
sshd[10219]: Accepted publickey for darix from 1.2.3.4 port 39190 \
  ssh2: ED25519 b4:3d:... [MD5]
```

```
ssh -i ~/.ssh/id_25519-cert.pub somehost.example.com
```

```
sshd[10241]: Accepted publickey for darix from 1.2.3.4 port 39194 \
  ssh2: ED25519-CERT ID user_darix (serial 0) CA RSA 58:5a:... [MD5]
```

```
ssh -i ~/.ssh/id_25519-cert.pub nobody@somehost.example.com
```

```
sshd[6377]: error: Certificate invalid: name is not a listed principal
```

DU KOMMST HIER NICHT REIN

```
ssh-keygen -k -f /etc/ssh/revoke.crl darix_id_rsa.pub
```

```
ssh-keygen -ku -f /etc/ssh/revoke.crl darix_id_ed25519.pub
```

```
# /etc/ssh/sshd_config
RevokedKeys /etc/ssh/revoke.crl
```

```
ansible all -m copy \
  -a 'src=/etc/ssh/revoke.crl dest=/etc/ssh/revoke.crl'
```

```
sshd[11462]: error: WARNING: authentication attempt with a revoked \
  RSA key 99:41... [MD5]
sshd[11462]: error: WARNING: authentication attempt with a revoked \
  RSA-CERT key 99:41... [MD5]
```

DA WAR NOCH WAS

- TrustedUserCAKeys funktioniert nur wenn man die Nutzernamen im Zertifikat einträgt (-n [listofusernames])
- Man kann Clientzertifikate auch mit `~/.ssh/authorized_key` benutzen
- Empfehlung: Wenn möglich authorized_keys deaktivieren in sshd_config
- Alle Optionen verfügbar in authorized_keys können auch beim Erstellen eines Clientzertifikates angegeben werden.
- Um zu sehen was ein Clientzertifikat erlaubt: `ssh-keygen -L -f id_25519-cert.pub`

Q & A

https://p.nordisch.org/olddognewtricks-slac2015/

THANK YOU FOR FLYING WITH OPENSUSE