SSH – The Secure Shell

Überblick, Technologie, Anwendungsbeispiele

Susanne Kießling

Hochschule Augsburg
Masterstudiengang Informatik
E-Mail: susanne.kiessling@hs-augsburg.de

7. Juli 2016

Kurzfassung: Die Möglichkeit, Daten innerhalb eines Computernetzwerks über eine gesicherte Verbindung übertragen zu können, ist heutzutage selbstverständlich. Wenn auch unbemerkt, kommt dafür häufig SSH zum Einsatz. Remote-Terminal-Sitzung, Datenübertragung, Einbinden von Verzeichnissen – all diese Funktionalitäten sind mit SSH komfortabel und vorallem gesichert möglich. Gleichwohl ist es wichtig, bei der Verwendung von SSH auf einige sicherheitsrelevante Aspekte zu achten.

1 Einleitung

Die Übertragung von Daten innerhalb von Computernetzwerken geschieht ohne entsprechende Vorkehrungen grundsätzlich ungesichert. E-Mails, Passwörter und sämtliche Daten, die übertragen werden, können abgefangen und gelesen werden. SSH (Secure Shell) ist ein Protokoll, das die verschlüsselte Übertragung von Daten über ein Netzwerk ermöglicht [BS01]. Zu den Vorgängern von SSH zählen telnet und die r-Befehle unter *UNIX*, beide ermöglichen jedoch nur eine unverschlüsselte Kommunikation zwischen Client und Server.

Obwohl die Entwicklung des SSH-Protokolls bereits 20 Jahre zurückliegt, ist die Verschlüsselung von Daten und damit auch SSH, ein hochaktuelles Thema. Vorallem die Enthüllungen durch Edward Snowden haben gezeigt, dass Daten massenhaft gesammelt und ausgewertet werden [GZW+14]. Um Daten vor dem Zugriff durch Dritte zu schützen, ist der Einsatz von Verschlüsselungsverfahren sinnvoll. Die Sicherheit von Verschlüsselungsverfahren hängt einerseits von den verwendeten Algorithmen ab und andererseits von der Konfiguration durch den Benutzer. Klarer ausgedrückt: Das sicherste Verschlüsselungsverfahren ist nutzlos, sobald der Benutzer beispielsweise schwache Passwörter verwendet, die leicht auf Brute-Force-Attacken ansprechen. SSH bietet hier beispielsweise die Möglichkeit der Public-Key-Authentifizierung. Neben Details zur Public-Key-Authentifizierung gibt das Paper einen generellen Überblick zu SSH. OpenSSH wird dazu verwendet, um elementare Funktionalitäten, wie beispielsweise den entfernten Login auf einem Server, vorzustellen. Für den Administrator des SSH-Servers werden empfehlenswerte Konfigurationen genannt.

Das Paper konzentriert sich auf die Verwendung von SSH und speziell die Implementierung OpenSSH unter Linux. Damit ist die Idee verbunden, ein eher unscheinbares Kommandozeilen-Tool vorzustellen, das gleichzeitig simpel in der Anwendung und hochwirksam für die Sicherheit bei Datenübertragungen ist.

2 Grundlegendes zu SSH

SSH verschlüsselt die forlaufende Verbindung zwischen Computern in einem Netzwerk. Dazu verwendet SSH eine Client/Server-Architektur. Abbildung 1 veranschaulicht das Grundprinzip von SSH, das laut seiner Spezifikation die Authentifizierung, die Verschlüsselung und die Integrität von Daten, die in einem Netzwerk übertragen werden, beinhaltet. [BS01, S. 4] [iet]

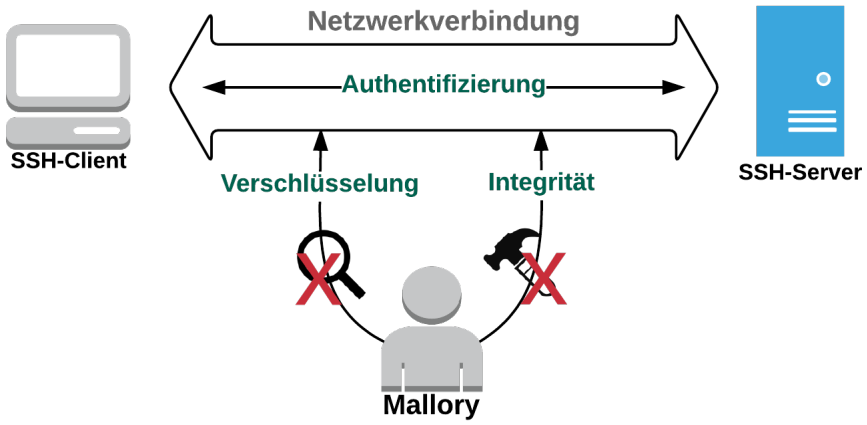

Abbildung 1: Grundprinzip von SSH, eigene Darstellung.

Wie in der Abbildung 1 zu sehen ist, verlangt eine SSH-Verbindung zwei Authentifizierungen [BS01, S. 47]. Der Server authentifiziert sich beim Client und umgekehrt (näheres zum Ablauf unter 4.1). Zum Verschlüsseln der Daten werden zufällige Sitzungsschlüssel ausgehandelt. Nach Beenden der Sitzung verlieren diese Schlüssel ihre Gültigkeit. SSH garantiert die Integrität der übertragenen Daten, d. h. die Daten können nicht durch einen Dritten (Mallory) verändert werden.

2.1 Protokolle

Das SSH-Protokoll existiert in zwei unterschiedlichen Versionen (SSH-1.x und SSH-2.x), die nicht kompatibel sind. SSH-1 wurde 1995 von Tatu Ylönen an der University of Technology in Helsinki entwickelt. Da es Mängel in der Integritätsprüfung aufweist, ist von einer weiteren Verwendung des SSH-1 Protokolls dringend abzuraten. [BKB+07, S. 456]

SSH-2 ist modular aufgebaut. Die Architektur besteht aus drei Hauptkomponenten: Die Transportschicht, die Authentifizierungsschicht und die Verbindungsschicht. Beschrieben sind die Komponenten durch die *Request for Comments* 4250 bis 4254 der IETF (Internet Engineering Task Force) [iet]. Ziel der Entwicklung von SSH-2 war es, neben Erweiterungen, die Sicherheitsmängel von SSH-1 zu beheben. Eine Zusammenfassung der Unterschiede zwischen SSH-1 und SSH-2 ist unter [BS01, S. 87] zu finden.

2.2 Implementierungen

Als erste Implementierung des SSH-Protokolls gilt das von Tatu Ylönen 1995 veröffentlichte SSH1. Es wurde als freie Software veröffentlicht [BS01, S. 11]. Die Implementierung des SS-

H-2-Protokolls war hingegen ein kommerzielles Produkt. Die kostenlose Nutzung war nur für Bildungseinrichtungen und den nicht-gewerblichen Bereich erlaubt [BS01, S. 22]. Folglich kam weiterhin vielerorts die Implementierung des SSH-1-Protokolls zum Einsatz. 1999 entstand das OpenSSH-Projekt, das auf die letzte frei verfügbare Version von SSH aufbaut [Ope].

Aktuell gibt es verschiedene Implementierungen des SSH-Protokolls. Je nach Einsatzzweck (mobiles Gerät/Workstation) oder Betriebssystem gibt es verschiedene Alternativen. Im Folgenden ist eine kleine Auswahl von Implementierungen mit Kurzbeschreibung aufgeführt.

PuTTY steht unter der MIT-Lizenz und stellt einen SSH-Client zur Verfügung. Hauptsächlich wird es unter Windows verwendet, ist aber auch für UNIX-Systeme verfügbar.[put] [Sta06, S. 241]

Dropbear ist eine Implementierung des SSH-2-Protokolls und steht unter der MIT-Lizenz [Joh]. Dropbear ist vorallem für Systeme mit geringer Prozessorleistung und geringer Speicherkapazität interessant. Es ist auf verschiedenen POSIX-basierten Systemen lauffähig. Eine entsprechende Auflistung ist auf der offiziellen Webseite von Dropbear [1] zu finden.

Mosh (mobile shell) bietet erweiterte Funktionalitäten, vorallem für mobile Geräte. Beispielsweise wird die Verbindung bei Roaming aufrechterhalten. Mosh ist keine direkte Implementierung des SSH-Protokolls. Es überträgt keinen Byte-String zwischen Client und Server. Vielmehr tauschen Client und Server einen Snapshot des aktuellen Bildschirms aus [mos].

OpenSSH ist eine der am weitesten verbreiteten Implementierungen des SSH-1 und SSH-2 Protokolls für Server und Client, vorallem deshalb, weil es auf vielen unixoiden Betriebssystemen vorinstalliert ist. Es steht unter der BSD-Lizenz. [Ope] Für Windows gibt es eine cygwin-Portierung. [Sta06, S. 263]

3 OpenSSH

OpenSSH wird nun dazu verwendet, um elementare Funktionalitäten einer SSH-Implementierung vorzustellen. Der Client wird unter OpenSSH mit *ssh* und der Server mit *sshd* bezeichnet. Alle systemweiten Konfigurationen bezüglich Client und Server sind in den jeweiligen Konfigurationsdateien (Client: ssh_config, Server: sshd_config) zu finden.

Für alle vorgestellten OpenSSH-Teilkomponenten gibt es unter der jeweiligen *manpage* (Manual Page) eine Übersicht der verfügbaren Optionen. Der Aufruf der manpage für sshfs lautet beispielsweise:

```
$ man sshfs
```

3.1 Remote Terminal Session

Der enfternte Login auf einem Server ist das Basis-Feature von OpenSSH. Dazu ist lediglich der Befehl *ssh* und der *hostname*, zu dem eine Verbidung aufgebaut werden soll, notwendig. Alle anderen Werte sind optional. Eine Übersicht der Parameter ist auf den manpages von *ssh* zu finden.

[1] https://matt.ucc.asn.au/dropbear/dropbear.html#platforms

```
[sue@kaktus]$ ssh micra@login.rz.hs-augsburg.de
micra@login.rz.hs-augsburg.de's password:
Plan your installation, and FAI installs your plan.

Last login: Mon Apr 25 22:38:45 2016 from p508[...]
micra@bug:~$
micra@bug:~$ exit
logout
Connection to login.rz.hs-augsburg.de closed.
```

Im Beispiel loggt sich der Benutzer *sue* (Client: *kaktus*) unter dem Benutzer *micra* auf dem Server *login.rz.hs-augsburg.de* ein. Anschließend authentifiziert der Server den Client durch eine Passwortabfrage. Mit *micra@bug: $* steht das Remote-Terminal zur Verfügung. Mit *exit* kann die Sitzung beendet werden. Neben der Passwortabfrage ist es möglich, sich über einen SSH-Schlüssel beim Server zu authentifizieren (siehe 3.4).

Bei jedem Sitzungsaufbau wird clientseitig geprüft, ob der öffentliche Schlüssel des Servers in der Liste bereits bekannter Hosts (*known_hosts*) enthalten ist. Handelt es sich um den ersten Login auf einem Server, gibt es einen Warnhinweis, dass es keinen Eintrag in der *known_hosts*–Datei gibt. Dieser Mechanismus soll Man-in-the-Middle Angriffe verhindern. Stimmt der Benutzer dem Fortfahren zu, wird der öffentliche Schlüssel des Servers in die *known_hosts*–Datei aufgenommen und beim nächsten Sitzungsaufbau kommt es zu keinem erneuten Warnhinweis. [BS01, S. 25]

3.2 Datenübertragung mit *scp*

scp steht für *secure copy* und ermöglicht die gesicherte Übertragung von Daten zwischen Client und Server. Es können einzelne Dateien bis hin zu ganzen Verzeichnisstrukturen kopiert werden. [Sta06, S. 80] Mit dem Befehl *scp*, der Datei, die kopiert werden soll und der Angabe des Zielservers sind alle Pflichtparameter vorhanden. Nach Abfrage des Passworts durch den Server wird der Fortschritt der Datenübertragung angezeigt.

```
[sue@kaktus ~]$ scp file.txt micra@login.rz.hs-augsburg.de:~/
micra@login.rz.hs-augsburg.de's password:
file.txt 100% 6297 6.2KB/s 00:00
```

Im Beispiel wird die Datei *file.txt* vom Homeverzeichnis des Clients (*sue@kaktus*) in das Homeverzeichnis des Servers (*micra@login.rz.hs-augsburg.de:∼/*) kopiert. Unter Angabe eines Dateinamens hinter dem Server kann die Datei auf dem Server beliebig benannt werden. In den manpages von *sshfs* sind noch zahlreiche Optionen zu finden.

Aus dem Beispiel wird deutlich, dass *scp* eine simple und effiziente Möglichkeit darstellt, Dateien sicher zwischen Client und Server zu übertragen. Es eignet sich beispielsweise hevorragend, um als Systemadministrator Aktualisierungen oder diverse Medien zu verteilen. [Sta06, S. 80]

3.3 Dateisystem einhängen mit *sshfs*

Mit *sshfs* (*Secure Shell File System*) lassen sich entfernte Verzeichnisse ins lokale System einhängen. Beim Client kommt dafür das *FUSE (Filesystem in Userspace)* Kernel-Modul zum Einsatz. Der große Vorteil ist hier, dass der Benutzer am Client-PC mit den eingehängten Dateien arbeiten kann, als wären es lokale Dateien. [BKB+07, S. 482]

Zum Einhängen ist der Befehl *sshfs*, der *host* und der *mountpoint*, d. h. der Pfad an dem das Verzeichnis eingehängt werden soll, notwendig. Mit dem Befehl *fusermount -u mountpoint* lässt sich das eingehängte Verzeichnis entfernen.

```
# Mountpoint erstellen und Inhalt mit ls anzeigen lassen:
$ mkdir localdir
$ ls localdir/

# Homeverzeichnis vom Benutzer micra,
# des Servers login.rz.hs-augsburg.de einhängen:
$ sshfs micra@login.rz.hs-augsburg.de:/rz2home/micra localdir/
micra@login.rz.hs-augsburg.de's password:

# Inhalt mit ls anzeigen lassen:
$ ls localdir/
Arbeitsfläche Dokumente [...]

# Verzeichnis aushängen:
$ fusermount -u localdir
```

Als Vorteil gegenüber Alternativen, wie NFS oder Samba, nennen die Entwickler von *sshfs* beispielsweise die Möglichkeit des Einhängens externer Ressourcen bei bereits existierendem SSH-Zugang. Bei NFS und Samba würde man hier in der Regel einen zusätzlichen Zugang bzw. Dienst, welcher die entsprechende Ressource exportiert, benötigen. Außerdem bietet *sshfs* den Vorteil, dass Verzeichnisse jederzeit vom Benutzer an beliebiger Stelle eingehängt werden können. [ssha]

3.4 Public-Key-Authentifizierung (*ssh-keygen, ssh-copy-id*)

Der Client authentifiziert sich beim Server standardmäßig über die Eingabe eines Passworts. Die Verwendung von Passwörtern bringt jedoch verschiedene Nachteile mit sich. Häufig werden unsichere Passwörter verwendet, die leicht auf Brute-Force-Attacken oder Wörterbuchattacken ansprechen.[Sch] Außerdem kann das Passwort auf einem unterwanderten Server abgefangen werden, auch wenn die Übertragung selbst gesichert war. [BS01, S. 29]

Daher bietet OpenSSH *Public-Key-Authentifizierung* an. Hierzu wird ein Schlüsselpaar bestehend aus einem privaten und einem öffentlichen Schlüssel benötigt. Der private Schlüssel ist geheim, der öffentliche Schlüssel wird auf dem Server hinterlegt (näheres hierzu siehe 4.1). [Sta06, S. 114] An dieser Stelle kommt *ssh-keygen* zum Einsatz. Mit diesem Tool ist es möglich, das Schlüsselpaar unkompliziert und komfortabel erstellen zu lassen. Standardmäßig erzeugt *ssh-keygen*

ein RSA-Schlüsselpaar [2]. Alternative Verschlüsselungsverfahren sind DSA, ECDSA und RSA1. Diese können mit der Option *-t* und der Angabe des gewünschten Verfahrens gewählt werden. Für RSA verwendet OpenSSH standardmäßig eine Schlüssellänge von 2048 Bit. Laut aktueller Einschätzung ist die Schlüssellänge von 2048 Bit langfristig ausreichend. Grundsätzlich gilt, je größer die Schlüssellängen, desto länger dauert es, den Schlüssel mit einer Brute-Force-Attacke zu berechnen. Längere Schlüssel verlangsamen das Verfahren jedoch. Je nach Einsatzzweck müssen diese Umstände berücksichtigt werden. [bsi, S. 27 ff.]

```
# Option -b für die Wahl der Schlüssellänge in Bits
[sue@kaktus ~]$ ssh-keygen -b 4096

# Option -t für die Wahl des Verschlüsselungsverfahrens
[sue@kaktus ~]$ ssh-keygen -t dsa
```

Nach dem Erstellen der Schlüssel legt *ssh-keygen* das lokale Verzeichnis *.ssh* an. Dort wird der öffentliche und private Schlüssel gespeichert. Hierbei ist der private Schlüssel durch ein Passwort geschützt, welches der Benutzer bei der Erstellung des Schlüsselpaars zu vergeben hat.

Im nächsten Schritt muss der öffentliche Schlüssel auf dem Server hinterlegt werden. Dazu stehen verschiedene Möglichkeiten zur Verfügung. Man könnte den öffentlichen Schlüssel mit dem Befehl *scp* auf den Server kopieren und zwar in das Verzeichnis *.sshauthorized_keys*. Wesentlich einfacher funktioniert es mit dem Befehl *ssh-copy-id*. Unter Angabe des *hostname* kopiert der Befehl *ssh-copy-id* den öffentlichen Schlüssel automatisch ins Verzeichnis *.sshauthorized_keys* des Servers.

```
[sue@kaktus ~]$ ssh-copy-id micra@login.rz.hs-augsburg.de
```

Um Public-Key-Authentifizierung durchführen zu können, muss es vom Server freigegeben sein. Das Attribut *PubKeyAuthentication* sollte dafür in der Konfigurationsdatei des Servers auf *yes* gesetzt sein. [Sta06, S. 116]

3.5 Weitere Funktionalitäten

SSH-Agenten: Die Eingabe der Passphrase für die Freigabe des privaten SSH-Schlüssels muss normalerweise bei jeder Remote-Verbindung neu durchgeführt werden. SSH-Agenten ermöglichen es, dass die Passphrase lediglich einmal eingegeben werden muss. Zukünftige Anfragen verwaltet der SSH-Agent im Hintergrund. [BS01, S. 230] [Sta06, S. 132]

Forwarding: SSH macht es möglich, den Datenstrom von TCP/IP-Anwendungen zu verschlüsseln. Das wird *Port-Forwarding*, oder auch *Tunneling* genannt. [BS01, S. 337] Um *Port-Forwarding* ausführen zu können, muss TCP-Forwarding in der Konfigurationsdatei des Servers als erlaubt gekennzeichnet sein. [Sta06, S. 151]

Mit *X-Forwarding* kann die entfernte grafische Oberfläche auf dem lokalen Rechner dargestellt werden. Das X-Protokoll selbst ist ein Window-System für *UNIX*, überträgt Daten aber ungesichert. [BS01, S. 363]

[2] RSA ist ein asymmetrisches Verschlüsselungsverfahren und steht für Rivest, Shamir und Adleman

4 Sicherheitsrelevante Betrachtung

4.1 Ablauf einer sicheren Verbindung

Verbindungsaufbau

Der Aufbau einer sicheren Verbindung läuft grob wie folgt ab (siehe Abb. 2): Der Client sendet eine Anfrage an den Server. SSH nutzt hier standardmäßig Port 22, das ist jedoch konfigurierbar. Der Server gibt seine Identität (host-ID), das verwendete Protokoll (SSH-1, SSH-2) und weitere Daten bekannt. Falls der Client zum ersten Mal mit dem Server kommuniziert, wird eine Warnung ausgegeben. Fährt der Client fort, wird die host-ID in die Liste der *known_hosts* aufgenommen. Für die Sitzung wird mittels Diffie-Hellman-Verfahren ein Session-Key erzeugt. Abschließend wählt der Client eine der vorgeschlagenen symmetrischen Verschlüsselungsverahren.

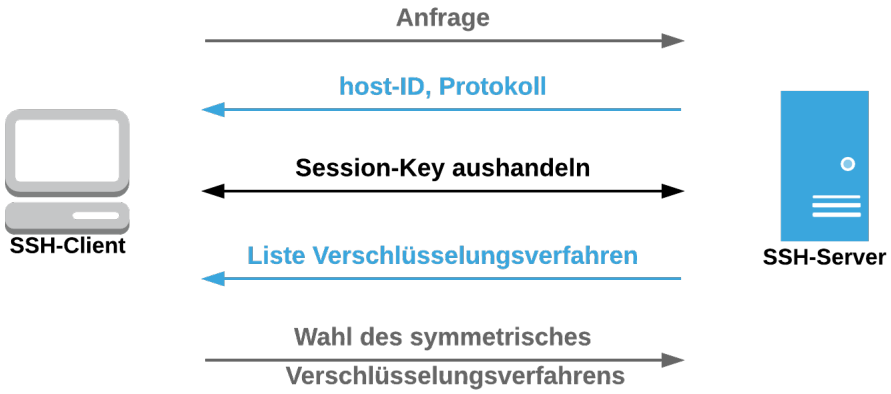

Abbildung 2: Grober Ablauf des Verbindungsaufbaus bei SSH, eigene Darstellung.

Public-Key-Authentifizierung

An dieser Stelle ist die Verbindung verschlüsselt und die Authentifizierung findet statt. Für eine Public-Key-Authentifizierung sieht dies folgendermaßen aus (siehe Abb. 3): Der Server generiert einen Zufalls-String und verschlüsselt diesen mit dem öffentlichen Schlüssel des Clients. Der Client entschlüsselt diesen Zufalls-String mit seinem privatem Schlüssel. Anschließend kombiniert der Client den entschlüsselten String mit dem Session-Key und generiert daraus eine md5-Hashsumme. Der Server führt diese Aktion ebenfalls durch und vergleicht die beiden md5-Hashsummen. Stimmen die beiden Summen überein, ist der Client erfolgreich authentifiziert. [BS01, S. 64] [man]

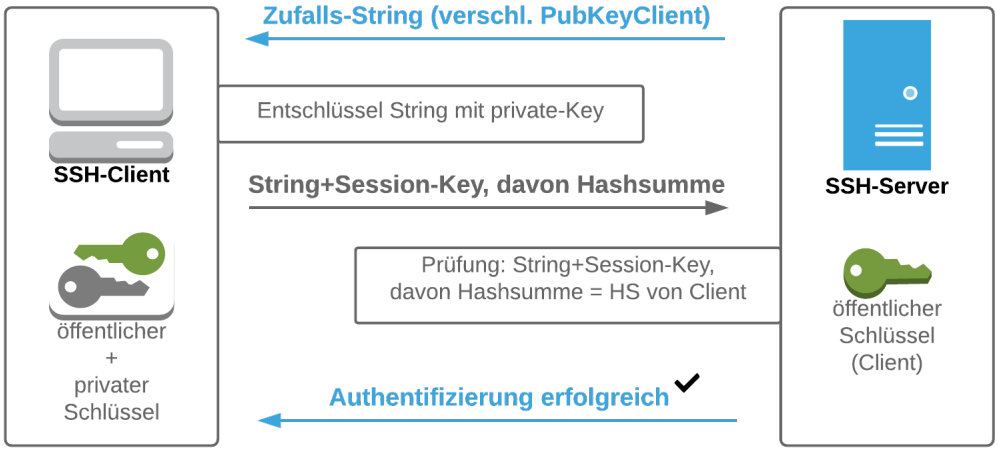

Abbildung 3: Ablauf der Public-Key-Authentifizierung bei SSH, eigene Darstellung, angelehnt an [BS01, S. 34]

4.2 Aktuelle Schwachstellen verfolgen

Es ist wichtig, über aktuelle sicherheitskritische Schwachstellen informiert zu sein. Möglicherweise ist es sinnvoll, einen *Workaround* einzurichten, bis es eine Aktualisierung der Software gibt.

OpenSSH listet die Schwachstellen auf deren offiziellen Internetseite. Es ist angegeben, welche Versionen betroffen sind und welche Auswirkungen der jeweilige Fehler hat. Zusätzlich wird teilweise eine Handlungsempfehlung gegeben. [sshb]

CVE-Datenbanken sind eine weitere Möglichkeit, aktuelle Sicherheitslücken zu verfolgen. CVE steht für *Common Vulnerabilities and Exposures*. Ziel von CVE ist es, eine einheitliche Benennung von Sicherheitslücken zu gewährleisten und das über verschiedene Systeme und Datenbanken hinweg. Eine bestimmte Sicherheitslücke soll einen einzigen Namen tragen, der sie eindeutig identifiziert. [cvea]

CVE-Details ist ein Beispiel für eine Internet-Plattform, auf der Sicherheitslücken zu einer bestimmten Software nach dem CVE-Standard gelistet sind. Eine Übersicht für OpenSSH ist unter [cveb] zu finden.

4.3 Anforderungen an den Serveradministrator

Der Administrator eines SSH-Servers muss grundsätzlich dafür sorgen, dass die verwendete Software aktuell ist. Die Konfigurationsdatei enthält Standardwerte und muss auf den jeweiligen Anwendungsfall angepasst werden. Änderungen in der Konfigurationsdatei sind nur mit Root-Rechten möglich. Handelt es sich um ein sehr sicherheitskritisches System, könnte man eine Prüfsumme (md5, SHA1) der Konfigurationsdatei erstellen, um mögliche Manipulationen zu erkennen. [Sta06, S. 50]

Im Folgenden sind empfehlenswerte Einstellungen für die *.sshd_config*-Datei genannt, die sich auf die OpenSSH-Version 7.2p2 beziehen. Da sich die Optionen und Standardwerte von Version zu Version ändern können, ist die Manpage der jeweiligen Version ausschlaggebend. Die genannten Einstellungen geben einen Überblick einer sinnvollen Konfiguration für einen Großteil der Anwendungsfälle.

PermitEmptyPasswords no: Diese Option muss zwingend auf *n*o gesetzt sein. Andererseits ist es möglich, dass sich Clients ohne eine Passwort einloggen können.

PasswordAuthentification no: Es ist empfehlenswert, diese Option zu setzten, sollte die Authentifizierung mittels Public-Key-Verfahren erlaubt sein und gleichzeitig keine Passwort-Authentifizierung gewünscht sein.

Protocol 2: Eine Verwendung von SSH-1-Protokoll müsste explizit angegeben werden. Standardmäßig ist Version 2 gesetzt.

AllowUsers: Der SSH-Zugriff wird dadurch auf bestimmte Benutzer beschränkt. Alternativ ist es auch möglich, bestimmte Benutzer über die Option DenyUsers auszuschließen.

ClientAliveInterval, ClientAliveCountMax: Diese Optionen regeln, wie lange der Server wartet, bis er die Verbindung trennt, sollte sie vom Client ungenutzt (im Leerlauf) sein.

PermitRootLogin no: Diese Option deaktiviert den Login als Root-Benutzer. In der Praxis ist dies sinnvoll, um automatisierte Angriffe auf das Root-Konto zu unterbinden. Müssen administrative Arbeiten durchgeführt werden, die Root-Rechte erfordern, ist das Arbeiten mittels sudo-Kommando generell empfehlenswert.

Port, ListenAddress: Standardmäßig läuft SSH auf Port 22. Um automatisierte Brute-Force-Attacken auf den Port 22 zu minimieren, ist die Wahl eines alternativen Ports empfehlenswert. Desweiteren kann der IP-Adressbereich, auf dem der Server lauscht, mittels ListenAddress begrenzt werden.

Ciphers, KexAlgorithms, MACs: Über diese Optionen lässt sich steuern, welche Algorithmen zugelassen werden. Verschlüsselungschiffren, Algorithmen für den Schlüsselaustausch und MACs (Message Authentication Code) für den Schutz der Integrität der Daten. Die erlaubten Algorithmen werden durch Komma separiert angegeben.

Detaillierte Informationen zu den genannten Optionen finden sich in der manpage zu sshd_config.

5 Fazit & Ausblick

Die Demonstration der Funktionalitäten von OpenSSH haben gezeigt, dass eine verschlüsselte Übertragung von Daten durch SSH relativ simpel und komfortabel möglich ist.

Die sichere Übertragung von Daten ist nicht nur für den privaten Gebrauch empfehlenswert. In sämtlichen öffentlichen und geschäftlichen Bereichen werden Verbindungen in Netzwerken hergestellt, die vermutlich nicht oder nur unzureichend gesichert sind. Im medizinischen Bereich handelt es sich beispielsweise um sensible Patientendaten, die man nicht in den Händen von Dritten wissen möchte.

Werden Daten über unsichere Verbindungen übertragen, liegt es häufig daran, dass Unwissenheit über die Gefahren herrscht, oder die Anwendung eines Verschlüsselungsverfahrens als zu kompliziert empfunden wird. Auch bei Verwendung von sicherer Kommunikation, beispielsweise über SSH, kann wie bereits erwähnt durch die Verwendung schwacher Passwörter oder auch Social Engineering die Sicherheit untergraben werden. Public-Key-Authentifizierung trägt in diesem Fall bereits zu einer Verbesserung bei.

Abschließend soll die Möglichkeit der Zwei-Faktor-Authentifizierung erwähnt werden. Die Zwei-Faktor-Authentifizierung mithilfe eines Tokens (z.B. YubiKey [Mau]) ermöglicht das Ablegen des privaten Schlüssels getrennt vom Client. Somit kann verhindert werden, dass Angreifer den privaten Schlüssel vom Client kopieren. Selbst bei passwortgeschützen Schlüsseln besteht die Möglichkeit, per Brute-Force an das Passwort zu gelangen. Vorallem bei den bereits erwähnten schwachen Passwörtern.

Dieser kurze Einschub zu YubiKey und privatem SSH-Schlüssel sollte einen Ausblick auf eine der vielen spannenden Themen im Bereich sicherer Datenübertragung mit SSH geben. Ich bin der Meinung, das Thema Verschlüsselung von Daten im Allgemeinen ist wichtiger denn je und hoffe in Form dieses Papers über SSH einige Menschen für das Thema sensibilisieren zu können.

Literatur

[BKB+07] BURNS, B. ; KILLION, D. ; BEAUCHESNE, N. ; MORET, E. ; SOBRIER, J. ; LYNN, M. ; MARKHAM, E. ; IEZZONI, C. ; BIONDI, P. ; GRANICK, J.S. u. a.: *Security Power Tools*. O'Reilly Media, 2007

[BS01] BARRETT, Daniel J. ; SILVERMAN, Richard E.: *SSH, the Secure Shell: the definitive guide*. O' Reilly Media, Inc., 2001

[bsi] *Bundesamt für Sicherheit in der Informationstechnik: Kryptografische Verfahren: Empfehlungen und Schlüssellängen*. `https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile`. – 15.02.2016

[cvea] *CVE: About CVE*. `http://cve.mitre.org/about/`. – 06.04.2016

[cveb] *CVE Details: OpenSSH Security Vulnerabilities*. `http://www.cvedetails.com/vulnerability-list/vendor_id-97/product_id-585/Openbsd-Openssh.html`

[GZW+14] GREENWALD, G. ; ZYBAK, M. ; WEISS, R.A. ; GOCKEL, G. ; WOLLERMANN, T.: *Die globale Überwachung: Der Fall Snowden, die amerikanischen Geheimdienste und die Folgen*. Droemer eBook, 2014

[iet] Network Working Group: The Secure Shell (SSH) Protocol Architecture. `https://tools.ietf.org/html/rfc4251`. – Januar 2006

[Joh] JOHNSTON, Matt: Dropbear SSH, offizielle Internetseite. `https://matt.ucc.asn.au/dropbear/dropbear.html`

[man] *OpenBSD: SSH–General Commands Manual*. `http://man.openbsd.org/ssh`. – 29.06.2016

[Mau] MAURO, Alessio D.: *Secure Shell With A YubiKey*. `https://www.yubico.com/wp-content/uploads/2015/08/Yubico_WhitePaper_Secure_Shell_With_YubiKey.pdf`. – 01.09.2015

[mos] *Mosh: Offizielle Internetseite*. `https://mosh.mit.edu/`

[Ope] *OpenSSH: Offizielle Internetseite*. `http://www.openssh.com/index.html`

[put] *PuTTY: Offizielle Internetseite*. `http://www.chiark.greenend.org.uk/~sgtatham/putty/`. – 10.03.2016

[Sch] SCHNEIER, Bruce: *Choosing a Secure Password*. `https://www.schneier.com/essays/archives/2014/02/choosing_a_secure_pa.html`. – 25.02.2014

[ssha] Entwickler von sshfs, sshfs FAQ. `https://github.com/libfuse/sshfs/blob/master/FAQ`. – 25.02.2016

[sshb] *OpenSSH: OpenSSH Security Issues*. `http://www.openssh.com/security.html`

[Sta06] STAHNKE, Michael: *Pro OpenSSH*. Apress, 2006