

## Anleitung

# Aufsetzen von Online-Werkzeugen

für Lehrveranstaltungen und Konferenzen

## Übersicht

Im ersten Teil dieser Anleitung ([online-werkzeuge.pdf](#)) wurden die Werkzeuge **Mumble**, **noVNC** und **OpenMeetings** aus Sicht der Teilnehmenden vorgestellt, im zweiten Teil ([online-werkzeuge-extra.pdf](#)) dann **TightVNC** und **PuTTY** aus Sicht der Vortragenden.

Dieser dritte Teil der Anleitung soll IT-Verantwortliche dabei unterstützen, die von uns eingesetzten Online-Werkzeuge auch bei sich aufzusetzen. **Diese Anleitung ist keine Einführung und keine Schritt-für-Schritt-Installationsanleitung.** Wir setzen vielmehr voraus, daß Sie **mit der Administration Ihres GNU/Linux-Servers vertraut** sind und in der Lage, online in englischer Sprache vorliegende Installationsanleitungen umzusetzen.

Diese Anleitung bezieht sich auf Debian GNU/Linux, Version 10 (Buster). Wenn Sie mit einer anderen Distribution arbeiten, ersetzen Sie nötigenfalls z. B. `apt` durch Ihren Paket-Manager, z. B. `yum`.

## Warum gerade diese Online-Werkzeuge?

Die Medien sind voll von Berichten über Online-Werkzeuge, die den Menschen helfen, einander auch bei physikalischem Abstand nahe zu bleiben. Nicht ganz so häufig sind die Analysen dieser Werkzeuge unter Gesichtspunkten wie Datenschutz und Privatsphäre. Nur am Rande wird gelegentlich erwähnt, daß Personen, die auf Datenschutz und Privatsphäre Wert legen, ja stattdessen GNU/Linux und freie Software (Open Source) verwenden könnten.

Genau darum geht es hier.

Alle in den ersten beiden Anleitungen beschriebenen Werkzeuge – **Mumble**, **OpenMeetings**, **TightVNC**, **noVNC** und **PuTTY** – sind freie Software. Insbesondere haben Sie das Recht, die Software bis hin zur Quelltextebene zu analysieren und sicherzustellen, daß sie wirklich nur das tut, wofür sie gedacht ist, und keine Hintertür enthält. Wenn Sie diese Werkzeuge für Ihre Organisation auf einem Server Ihrer Organisation in Ihrer Eigenschaft als Admin Ihrer Organisation installieren, ist dies die bestmögliche Garantie für Datenschutz und Privatsphäre, die es bei Online-Kommunikationswerkzeugen überhaupt geben kann.

Wenn man vorschlägt, diese Werkzeuge zu benutzen, hört man häufig: „Ja, aber die kommerziellen Werkzeuge sind doch viel einfacher in der Benutzung, haben mehr Möglichkeiten und sind stabiler.“ Dieser Argumentation haben wir folgendes entgegenzusetzen:

- Auch freie Software kann kommerziell sein. Wenn Sie jemanden dafür bezahlen, Ihnen die Software schlüsselfertig zu installieren, Sie darin zu schulen und Ihnen Telefon-Support zu leisten, erhalten Sie in der Regel sogar besseren Service als bei proprietären (nicht-freien, „Closed Source“) Produkten. Diese Dienstleistung hat dann zwar sicherlich auch einen angemessenen Preis in Euro, aber zumindest bezahlen Sie nicht mit Ihren privaten Daten und/oder Firmengeheimnissen.
- Wir haben einige proprietäre Online-Kommunikationswerkzeuge ausprobiert und können das oft behauptete Plus an Funktionsumfang und Stabilität *unter gleichen Bedingungen* nicht bestätigen. (So darf man zum Beispiel ein Web-Interface nur mit einem anderen Web-Interface vergleichen, nicht jedoch mit einer installierten Software.) Ob etwas korrekt funktioniert oder nicht und ob es viel oder wenig kann, hängt von vielen Faktoren ab, aber nicht unmittelbar von der Lizenz.
- Ein Werkzeug, das Sie ausspioniert, weiß naturgemäß mehr über Sie als eins, das Ihre informationelle Selbstbestimmung respektiert. Ein SSH-Tunnel, den Sie selbst konfigurieren müssen, ist besser als eine Anwendung, die ungefragt „nach Hause telefoniert“. Der einmalig etwas höhere Aufwand bei der Installation ist nun mal der Preis für mehr Datenschutz und Privatsphäre. Die Situation ist vergleichbar mit dem Sicherheitsgurt im PKW: Ohne Gurt mag es komfortabler sein, aber mit Gurt ist es sicherer.

Für weitere Details siehe den Artikel *Nachhaltige Online-Werkzeuge für Home Office und Lehre* in der Datei [online-werkzeuge-nachhaltig.pdf](#).

Neben den in diesen Anleitungen vorgeschlagenen freien Werkzeugen können wir im Kontext der Online-Kommunikation noch die folgenden empfehlen:

- **BigBlueButton** ist ein Online-Konferenz-Werkzeug mit einem ähnlichem Funktionsumfang wie *OpenMeetings*. In unseren Tests war es hinsichtlich Audio-Übertragung und Desktop-Sharing stabiler als *OpenMeetings*; es erreicht hingegen nicht die Klangqualität von *Mumble*.
- **NextCloud** ist ein bewährtes Werkzeug für den Austausch von Dateien und hat auch eine Funktion für Online-Konferenzen.
- **Jitsi Meet** ist ein sehr einfach zu bedienendes Werkzeug für Online-Konferenzen, an denen man auch per Telefon teilnehmen kann. In unseren Tests erreichte es nicht die Klangqualität von *Mumble* und war auch „nur“ bis zu ca. 10 Teilnehmenden stabil. Je nach Einsatzzweck kann es jedoch sehr nützlich sein.
- **OBS** ist eine Software für Live-Filmaufnahmen und Live-Streaming.
- **kdenlive** ist eine Software zum Bearbeiten von Filmen (Schnitte, Übergänge etc.).
- **VLC** kann Filme und Kamerabilder nicht nur wiedergeben, sondern auch streamen. In Kombination mit **Icecast** wird daraus eine Online-Sendestation.
- **GitLab** ist ein Web-Interface für **Git**.
- Für den öffentlichen Online-Chat ist **IRC** nach wie vor eine interessante Option. Es gibt dafür auch zahlreiche Web-Interfaces.

... und natürlich kennen Sie **SSH**, **GNU Screen** und **ytalk** für die Online-Kommunikation zwischen Experten.

# Let's Encrypt

Für sichere Verbindungen mittels TLS benötigen die Online-Dienste X.509-Zertifikate. Wir zeigen hier, wie man mit Hilfe von **Let's Encrypt** sichere Zertifikate erhält.

**Let's Encrypt** stellt ein Zertifikat dann aus, wenn man nachweisen kann, die Kontrolle über eine Domain zu haben. Wir benutzen für diesen Nachweis die DNS-Methode. Dabei stellt **Let's Encrypt** eine DNS-Anfrage an unseren Server, die wir mit einem unmittelbar zuvor von **Let's Encrypt** erhaltenen Zufalls-String beantworten müssen. Dies wird durch eine spezielle Software automatisiert, den **Let's Encrypt**-Client (**certbot**).

In unserem Beispiel läuft der DNS-Server auf einem Rechner **comm-0**, der **Let's Encrypt**-Client hingegen auf einem anderen Rechner **main-1**, auf dem das Zertifikat tatsächlich benötigt wird.

Die Installation erfolgt mittels:

```
root@main-1:~# apt install certbot
```

Die Konfiguration erfolgt in der Datei **/etc/letsencrypt/cli.ini**:

```
root@main-1:~# cat /etc/letsencrypt/cli.ini
# Because we are using logrotate for greater flexibility,
# disable the internal certbot logrotation.
max-log-backups = 0

email = $SOME_E_MAIL_ADDRESS

authenticator = manual
preferred-challenges = dns
domains = $HOST1, $HOST2

expand = True
manual-public-ip-logging-ok = True
preferred-challenges = dns
manual-auth-hook = /usr/local/sbin/letsencrypt_dns_update
post-hook = /usr/local/sbin/letsencrypt-install-new-keys
```

Hierbei steht

- **\$SOME\_E\_MAIL\_ADDRESS** für eine E-Mail-Adresse, an die bei Bedarf relevante Mitteilungen vom **Let's Encrypt**-Projekt gesendet werden sollen (z.B. ablaufende Zertifikate, die nicht erneuert wurden),
- **\$HOST1** und **\$HOST2** für die Namen beliebiger Rechner, für die jeweils ein Zertifikat beantragt werden soll, und
- **\$DOMAIN** für die Internet-Domain, in der sich die Rechner **main-1** und **comm-0** befinden.

**Hinweis:** Wenn Sie diese Code-Beispiele für sich kopieren wollen, verwenden Sie dazu besser nicht diese PDF-Datei, sondern den zugehörigen **LaTeX**-Quelltext (Datei **online-werkzeuge-admin.tex**). Dadurch vermeiden Sie Probleme mit der Zeichen-Codierung, die in einer PDF-Datei teilweise anders ist als in einer reinen Textdatei.

Zum Aktualisieren des DNS-Records für die Teilnahme an **Let's Encrypt** werden die Programme **dnssec-keygen** (einmalig für die Schlüsselerzeugung) und **nsupdate** (jedesmal)

benötigt. Die Installation dieser Programme erfolgt mittels `apt install bind9utils` bzw. `apt install dnsutils`.

Wir erstellen zunächst im Verzeichnis `/etc/bind/keys` mit `dnssec-keygen` ein Schlüsselpaar:

```
root@main-1:/etc/bind/keys# dnssec-keygen -a hmac-sha512 -n host \
-b 512 main-1.$DOMAIN
```

(Auch hier muß `$DOMAIN` durch die von Ihnen verwendete Internet-Domain ersetzt werden.)

`dnssec-keygen` erzeugt zwei Dateien:

```
root@main-1:/etc/bind/keys# ls
Kmain-1.$DOMAIN.$aaa+$iiii.key
Kmain-1.$DOMAIN.$aaa+$iiii.private
```

Diese Dateinamen werden wir im folgenden mit `$KEY.key` bzw. `$KEY.private` abkürzen. (`$aaa` und `$iiii` stehen für Ziffernfolgen.)

Wir müssen nun dafür sorgen, daß der auf `main-1` erzeugte Schlüssel von `comm-0` auch akzeptiert wird. Hierzu tragen wir auf `comm-0` in die Datei `/etc/bind/named.conf.local` folgendes ein:

```
zone "_acme-challenge.mumble.$DOMAIN" {
    type master;
    file "/etc/bind/dynamic/db._acme-challenge.mumble.$DOMAIN";

    # look for dnssec keys here:
    key-directory "/etc/bind";

    allow-update { key main-1.$DOMAIN; };
};

include "/etc/bind/keys/key.main-1.$DOMAIN";
```

Inhalt der Include-Datei `/etc/bind/keys/key.main-1.$DOMAIN`:

```
key main-1.$DOMAIN. {
    algorithm HMAC-SHA512;
    secret "$PRIVATE_KEY";
};
```

Hierbei steht `$PRIVATE_KEY` für den in der Datei `/etc/bind/keys/$KEY.private` auf dem Rechner `main-1` enthaltenen privaten Schlüssel, d. h. alles, was dort hinter `key:` steht. (Der private Schlüssel ist Base64-kodiert und endet daher typischerweise mit mehreren Gleichheitszeichen.)

Danach kann mit Hilfe des folgenden Shell-Skripts das für *Let's Encrypt* notwendige dynamische DNS-Update ausgelöst werden:

```
root@main-1:~# cat /usr/local/sbin/letsencrypt_dns_update
#!/bin/bash
# Generate key using:
# dnssec-keygen -a hmac-sha512 -b 512 -n host main-1.$DOMAIN

if [ -z "$CERTBOT_DOMAIN" ] || [ -z "$CERTBOT_VALIDATION" ]
then
echo "EMPTY DOMAIN OR VALIDATION"
exit -1
```

```

fi

HOST="_acme-challenge"

/usr/bin/nsupdate -k \
    /etc/bind/keys/$KEY.private << EOM
server comm-0.$DOMAIN
zone ${HOST}.${CERTBOT_DOMAIN}
update delete ${HOST}.${CERTBOT_DOMAIN} TXT
update add ${HOST}.${CERTBOT_DOMAIN} 300 TXT "${CERTBOT_VALIDATION}"
send
EOM
echo ""

```

Wenn der eigene DNS-Server nicht der primäre Server für die Domain ist, wird das dynamische Update nicht sofort wirksam, sondern erst nach einem AXFR-Zonentransfer. Dadurch entstehen Verzögerungen, die die Authentifikation bei *Let's Encrypt* gefährden.

Um dies zu vermeiden, müssen wir den entsprechenden DNS-Eintrag auf den sonst für `$DOMAIN` benutzten Servern auf unseren eigenen Server delegieren. Dies geschieht auf dem DNS-Server `comm-0` in der Datei `/var/lib/bind/db.$DOMAIN`:

```
_acme-challenge.mumble IN NS dns-0
```

(`dns-0` steht für dieselbe IP-Adresse wie `comm-0`.)

Wenn wir unsere Rechner mit speziellen Namen für die angebotenen Dienste ansprechen möchten (z. B. `mumble.$DOMAIN` anstelle von `main-0.$DOMAIN`), geschieht dies in derselben Datei durch einen `CNAME`-Eintrag:

```
mumble IN CNAME main-0
```

Aus Sicherheitsgründen wollen wir den UDP-Port 53 nicht allgemein beantworten, sondern uns eigentlich auf die Server von *Let's Encrypt* beschränken. Leider scheinen deren Anfragen von sehr vielen verschiedenen Servern bei Amazon zu kommen, so daß die offensichtliche Lösung mittels `iptables` nicht anwendbar ist.

Als Umgehungslösung wird daher aktuell beim Durchführen der Zertifikatsanfrage auf `main-1` (durch `certbot certonly`) kurzzeitig auf dem Firewall-Rechner mittels

```
iptables -I INPUT -p udp --dport 53 -j ACCEPT
```

die Abfrage von `bind` von außen erlaubt.

## Mumble

Der Rechner, auf dem der *Mumble*-Server läuft, ist in diesem Beispiel ebenfalls `main-1.$DOMAIN`.

Die Installation erfolgt mittels: `apt install mumble-server`

*Mumble* besetzt (standardmäßig) den Port 64738.

Diesen müssen wir auf der Firewall freigeben.

Der *Mumble*-Servers ist vorkonfiguriert (Datei: `/etc/mumble-server.ini`). Sofern Sie keine speziellen Anforderungen haben (z. B. mehrere Instanzen von *Mumble* auf derselben Hardware), können Sie diese Konfiguration direkt übernehmen.

## Server-Zertifikat

Standardmäßig läuft *Mumble* mit einem selbst-signierten Zertifikat. Dies führt zu einem Verlust an Sicherheit und zu Warnmeldungen beim ersten Verbinden. Um optimale Sicherheit zu gewährleisten, installieren wir stattdessen das mit *Let's Encrypt* erzeugte Zertifikat (siehe oben) zusammen mit dem Schlüssel für *Mumble*. Dies geschieht mit Hilfe eines Shell-Skripts `/usr/local/sbin/letsencrypt-install-new-keys`:

```
#!/bin/sh

install -m 644 -o root -g mumble-server \
    /etc/letsencrypt/live/mumble.DOMAIN/fullchain.pem \
    /var/lib/mumble-server/cert.pem
install -m 640 -o root -g mumble-server \
    /etc/letsencrypt/live/mumble.DOMAIN/privkey.pem \
    /var/lib/mumble-server/privkey.pem

systemctl force-reload mumble-server
```

(Wir verwenden hier den Server-Namen `mumble` als `CNAME` für `main-1`.  
`$DOMAIN` steht wieder für die Internet-Domain des Servers.)

## Benutzerverwaltung

Die Benutzerverwaltung in *Mumble* erfolgt komplett innerhalb der Client-Software.

Zu Beginn existiert lediglich der Benutzer-Zugang `SuperUser`. Neue Benutzer\*innen können sich direkt mit dem *Mumble*-Server verbinden und dann mittels des Menüpunktes „Self“ → „Register“ ihr beim Starten des *Mumble*-Clients erzeugtes (oder von einem anderen Rechner importiertes) Zertifikat mit dem aktuell verwendeten Benutzernamen verknüpfen.

Mit dem *Mumble*-Client kann der `SuperUser`-Zugang dann genutzt werden, um die entsprechenden registrierten Personen in eine neu erstellte privilegierte Gruppe (z. B. „Lehrende“) aufzunehmen und diesen Rechte zu geben, die es ihnen z. B. ermöglichen, störende Teilnehmer\*innen eines Gesprächs ggf. stummzuschalten oder auszuschließen.

## OpenMeetings

Da die Installationsanleitungen für *OpenMeetings* von der Installation auf einem Ubuntu-System ausgehen und wir von anderen Software-Paketen her wissen, daß für Ubuntu vorgesehene Software u. U. nur problematisch auf einem Debian GNU/Linux zum Laufen zu bringen ist, installieren wir *OpenMeetings* in einer eigens dafür erzeugten virtuellen Maschine (VM), in unserem Fall auf Basis von *Xen*.

Die Erzeugung der virtuellen Maschine geschieht auf dem äußeren System mittels:

```
xen-create-image --size=15Gb --swap=4096Mb --memory=4096Mb \
    --maxmem=4096Mb --dist=bionic --ip=$IP --no-pygrub \
    --hostname=ubuntu-0.$DOMAIN --mirror=$UBUNTU_MIRROR
```

(`$UBUNTU_MIRROR` steht für die URL des `packages`-Verzeichnisses eines Ubuntu-Spiegels; `$IP` steht für die interne IP-Adresse der VM.)

Die hier beschriebene Installation folgt weitgehend der Anleitung

<https://cwiki.apache.org/confluence/display/OPENMEETINGS/Tutorials+for+installing+OpenMeetings+and+Tools?preview=/27838216/140774282/Installation%20OpenMeetings%205.0.0-M3%20on%20Ubuntu%2018.04%20LTS.pdf>

mit dem Unterschied, daß `ffmpeg` nicht selbst kompiliert, sondern aus den Paketquellen installiert wird.

## Kurento Media Server

Der **Kurento-Media-Server** ist für die Verarbeitung von Medienströmen (Audio-, Video- und Screensharing-Streams) notwendig und wird von *OpenMeetings* verwendet. Damit Medienströme weitergeleitet werden können, obwohl sich sowohl *OpenMeetings* als auch i. d. R. die Teilnehmenden hinter einem NAT befinden, wird dieser so konfiguriert, daß er Zugriff auf einen TURN-Server besitzt.

Dies geschieht durch Ergänzen der Zeile

```
turnURL=kurento:$TURN_PASSWORD@$IP:3478
```

in der Konfigurationsdatei `/etc/kurento/modules/kurento/WebRtcEndpoint.conf.ini` auf dem Rechner, auf dem *OpenMeetings* läuft (hier: VM mit Ubuntu). `$IP` steht hier für die von außen erreichbare IP-Adresse des TURN-Servers.

## Coturn

Der für die Verwendung des *Kurento Media Server* hinter einem NAT notwendige **TURN-Server** wird außerhalb einer VM direkt auf `dom0` (siehe <https://en.wikipedia.org/wiki/Xen>) mittels `apt install coturn` installiert. Die verwendete Konfiguration befindet sich dort in der Datei `/etc/turnserver.conf`:

```
listening-port=3478
tls-listening-port=5349
fingerprint
lt-cred-mech
user=kurento:$TURN_PASSWORD
realm=root-0.$DOMAIN
total-quota=100
stale-nonce
cert=/etc/ssl/le/fullchain.pem
pkey=/etc/ssl/le/privkey.pem
cipher-list="ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:
            DH+AES:ECDH+3DES:DH+3DES:RSA+AES:RSA+3DES:!ADH:!AECDH:!MD5"
no-loopback-peers
no-multicast-peers
verbose
```

(Der Wert von `cipher-list` ist ein durchgehender String und darf in der Datei nicht umbrochen werden. Dies geschieht hier nur notgedrungen aus Platzgründen. `$TURN_PASSWORD` steht für das gewählte Passwort.)



## Raumkonfiguration

Das Anlegen von Räumen und Benutzerkonten in *OpenMeetings* erfolgt über die Web-Oberfläche.

Da wir keine einfache Möglichkeit gefunden haben, die Lehrenden jeweils automatisch zu Moderatoren ernennen zu lassen, werden diese jeweils über das hier wiedergegebene Shell-Skript `/usr/local/sbin/give_rights_to_all_lecture_rooms` zu Moderatoren jeden Raumes ernannt:

```
#!/bin/sh
(for user in 2 5 6 7 8 12 30 31 32 33 34 35; do
  for room in $(seq 21 32); do
    echo "insert into room_moderator" \
      "(roomId, user_id) values ($room, $user);"
  done
done) | mysql --database=open503 -B
```

Die Zahlen sind die Benutzer- bzw. Raum-IDs in der SQL-Datenbank von *OpenMeetings* und müssen Ihrer Installation entsprechend angepaßt werden.

Nach Anwendung ist in der Ubuntu-VM einmalig `/etc/init.d/tomcat3 restart` auszuführen.

## NoVNC

Wir installieren zunächst *noVNC* mittels `apt install novnc`.

### systemd

Um mehrere Räume bereitstellen zu können, wird ein entsprechendes Skript für *systemd* geschrieben. Mit dieser Methode läuft *noVNC* dauerhaft im Hintergrund. (Alternativ wäre auch eine Aktivierung beim Kontaktaufbau zum entsprechenden TCP-Socket denkbar.)

Das *systemd*-Skript wird auf dem Rechner *main-0*, auf dem *noVNC* läuft, in der Datei `/etc/systemd/system/novnc@.service` abgelegt:

```
[Unit]
Description = start noVNC service %i
After=syslog.target network.target

[Service]
Type=simple
User=novnc
ExecStart = /usr/local/sbin/hsbo_novnc %i

[Install]
WantedBy=multi-user.target
```

Das *systemd*-Skript ruft ein Shell-Skript `hsbo_novnc` auf, das sich im Verzeichnis `/usr/local/sbin/hsbo_novnc` befindet und das eigentliche Launch-Skript von *noVNC* aufruft:



```
#!/bin/bash
```

```
i=$1; shift
```

```
exec /usr/share/novnc/utils/launch.sh --listen $((6080 + $i)) \  
--vnc localhost:$((5900 + $i)) $*
```

In unserem Beispiel starten wir 13 Instanzen von *noVNC*, durchnummeriert von 0 bis 12. Die Nummern sind gleichzeitig die Nummern der VNC-Server, auf die *noVNC* als VNC-Client zugreift.

Mit dem Shell-Befehl

```
for i in $(seq 0 12); do \  
    systemctl start novnc@$i; \  
    systemctl enable novnc@$i; \  
done
```

starten wir diese 13 Instanzen und aktivieren sie dauerhaft, so daß sie nach einem Reboot wieder zur Verfügung stehen.

## Apache

*noVNC* soll hinter einem Apache-Proxy bleiben. Zur Kommunikation werden WSS (secure Websockets) benötigt. Daher müssen mittels `a2enmod proxy_wstunnel proxy_http` die entsprechenden Module aktiviert werden.

Die entsprechende Apache-Konfiguration erfolgt in der Konfigurationsdatei `/etc/apache2/sites-enabled/novnc.$DOMAIN.conf`:

```
<VirtualHost _default_:80>  
    ServerAdmin webmaster@$DOMAIN  
    ServerName novnc.$DOMAIN  
    ServerAlias www.novnc.$DOMAIN  
  
    Redirect / https://novnc.$DOMAIN/  
  
    ErrorLog ${APACHE_LOG_DIR}/novnc.$DOMAIN-error.log  
    LogLevel warn  
    CustomLog ${APACHE_LOG_DIR}/novnc.$DOMAIN-access.log combined  
</VirtualHost>  
  
<VirtualHost _default_:443 _default_:8443>  
    ServerAdmin webmaster@$DOMAIN  
    ServerName novnc.$DOMAIN  
    ServerAlias www.novnc.$DOMAIN  
  
    ProxyPass /websocket ws://localhost:6080/websocket  
    ProxyPassReverse /websocket ws://localhost:6080/websocket  
  
    ProxyPass /websocket/0/ ws://localhost:6080/websocket  
    ProxyPassReverse /websocket/0/ ws://localhost:6080/websocket  
  
    ProxyPass /websocket/1/ ws://localhost:6081/websocket  
    ProxyPassReverse /websocket/1/ ws://localhost:6081/websocket
```

[...]

```
ProxyPass /websocket/12/ ws://localhost:6092/websocketify
ProxyPassReverse /websocket/12/ ws://localhost:6092/websocketify

ProxyPass / http://localhost:6080/
ProxyPassReverse / http://localhost:6080/

ErrorLog ${APACHE_LOG_DIR}/novnc.$DOMAIN-error.log
LogLevel warn
CustomLog ${APACHE_LOG_DIR}/novnc.$DOMAIN-access.log combined
SSLEngine on
</VirtualHost>
```

Dadurch wird sichergestellt, daß grundsätzlich alle Verbindungen verschlüsselt werden.

Auf die einzelnen VNC-Kanäle kann jetzt über URLs der Form

```
https://novnc.$DOMAIN/vnc.html?host=novnc.$DOMAIN&port=443
&path=websocket/$CHANNEL/&reconnect=1&resize=scale
```

zugegriffen werden. (`$CHANNEL` steht für die Nummer des VNC-Kanals. Der Umbruch in der URL geschieht hier nur aus Platzgründen und darf natürlich nicht in dieser Form übernommen werden.)

Die Optionen `&reconnect=1&resize=scale` bewirken eine Vorkonfiguration des *noVNC*-Clients, so daß er sich bei Verbindungsabbruch automatisch wiederverbindet und das VNC-Bild auf die Größe des Browser-Fensters skaliert. Diese Konfiguration läßt sich während der Benutzung von *noVNC* ändern (Icon: Zahnrad-Symbol).

## Lokale VNC-Server und Skalierung

Jeder Zugriff auf *noVNC* von außen baut eine VNC-Verbindung zum angeschlossenen VNC-Server auf. Um die Netzwerkverbindungen der Vortragenden nicht zu überlasten, leiten wir *noVNC* nicht direkt dorthin weiter, sondern zunächst auf einen lokal laufenden VNC-Server.

Als lokalen VNC-Server verwenden wir *tigervncserver*. (Mit *tightvncserver* kam es bei Zugriffen durch Browser, in denen fehlerhafte Plug-Ins installiert waren, zu Abstürzen, die dazu führten, daß alle bereits angemeldeten Teilnehmenden aus dem System geworfen wurden.)

Um auch die Netzwerkverbindungen der Betrachtenden nicht zu überlasten, bieten wir jeden VNC-Kanal (hier: von 1 bis 6) auf einem parallelen Kanal (hier: von 7 bis 12) auch in herunterskalierter Form an. Dies geschieht, indem wir in dem lokal laufenden VNC-Server einen *x11vnc*-Server starten, der das Bild in herunterskalierter Form exportiert. Ein im Vollbildmodus laufender VNC-Viewer zeigt das herunterskalierte Bild dann im Parallel-Kanal an. Hierfür verwenden wir *xtightvncviewer*. (Bei Verwendung von *xtigervncviewer* gab es Schwierigkeiten mit der Übergabe des Passworts für den Lesezugriff.)

Dies hat den Nebeneffekt, daß sich Teilnehmende bereits anmelden können, bevor die Vortragenden ihre Bildschirme per VNC exportiert haben. Um sie im VNC-Kanal willkommen zu heißen, zeigen wir ihnen mittels *xpdf* im Vollbildmodus eine Startseite.

Es müssen also wie folgt Programme installiert werden:

```
apt install tigervnc-standalone-server x11vnc xtightvncviewer xpdf
```

Das unten wiedergegebene Shell-Skript `vnc` startet mit dem Parameter `start` für alle eingetragenen Kanäle (hier: 1 bis 6) jeweils zwei `tigervncserver`: einen für die Betrachtung in voller Auflösung (hier: 1920×1080) und einen für die Betrachtung in herunterskalierter Auflösung (hier: 960×540).

Danach startet das Skript für jeden VNC-Kanal ein `xpdf`, das die Startseite (hier: `$HOME/vnc-testbild.pdf`) im lokalen `tigervncserver` des nicht skalierten Kanals anzeigt. Die angezeigte Seite der PDF-Datei entspricht dabei der Nummer des lokalen VNC-Servers.

Außerdem startet das Skript dort einen `x11vnc`-Server, der das Bild herunterskaliert und auf einem eindeutig gewählten Port lokal freigibt, von wo aus ein weiterer `xtightvnc-viewer` es im `tigervncserver` des skalierten Kanals anzeigt.

```
#!/bin/sh

channels=$(seq 6)
vncserver="tigervncserver"
vncserveroptshires="-depth 32 -geometry 1920x1080"
vncserveroptslores="-depth 32 -geometry 960x540"
# add "-localhost no" to make it listen on all IPs
testfile="$HOME/vnc-testbild.pdf"
action=help

while [ -n "$1" ]; do
  case "$1" in
    "-c") channels="$2"; shift 2;;
    "-s") vncserver="$2"; shift 2;;
    "-o") vncserveroptshires="$2"; shift 2;;
    "-t") testfile="$2"; shift 2;;
    start|stop|status|help) action="$1"; shift;;
    *)      echo "$0: unknown option $1" 1>&2; exit 1;;
  esac
done

case "$action" in

  start)

    "$vncserver" :0 $vncserveroptshires

    for x in $channels; do
      y=$((x + 6))
      HOME=$HOME/vnc$x "$vncserver" :$x $vncserveroptshires
      HOME=$HOME/vnc$y "$vncserver" :$y $vncserveroptslores
    done

    sleep 1

    for x in $channels; do
      (HOME=$HOME/vnc$x DISPLAY=: $x \
        xpdf -fullscreen "$testfile" $x &)
      z=$((x + 9500))
      (DISPLAY=: $x HOME=~/.vnc/$x \
        x11vnc -localhost -scale 1/2 -rfbport $z &)
    done
```

```

sleep 10

for x in $channels; do
    y=$((x + 6))
    z=$((x + 9500))
    (DISPLAY=:y xtightvncviewer -passwd $HOME/vnc$y/.vnc/passwd \
    -depth 16 -geometry 960x540+0+0 :$z &)
done

;;

stop)

"$vncserver" -kill :0

for x in $channels; do
    y=$((x + 6))
    HOME=$HOME/vnc$x "$vncserver" -kill :$x
    HOME=$HOME/vnc$y "$vncserver" -kill :$y
done

;;

status)

ps aux | grep -o "X.*vnc.*:[0-9][0-9]*"

;;

help)

echo Usage: $0 'start|stop|status|help [-c "<channels>"]' \
    '[-s <server>] [-o <options>] [-t <testfile>]'

;;

esac

```

Mit dem Parameter **stop** beendet das Shell-Skript alle lokalen VNC-Server, wodurch sich auch die lokalen VNC-Viewer und die lokalen **x11vnc**-Server automatisch beenden. (Weitere zulässige Parameter und deren Wirkung können Sie dem Shell-Skript selbst entnehmen.)

Das Skript setzt voraus, daß die Unterverzeichnisse **vnc1** bis **vnc12** existieren und schreibbar sind, so daß die lokalen VNC-Server dort ihre Konfiguration ablegen können. Diese Konfigurationen müssen getrennt werden, um unterschiedliche Passwörter für den Zugriff von außen vergeben zu können. Diese Passwörter können Sie mit

```
HOME=$(pwd) /vnc$NR vncpasswd
```

eingeben bzw. ändern. (**\$NR** steht für die Nummer des lokalen VNC-Servers, gleichbedeutend mit der Nummer des VNC-Kanals, ggf. um ein Offset erhöht (hier: 6) für die Version mit reduzierter Auflösung.) Es ist sinnvoll, für den Kanal in voller bzw. in reduzierter Auflösung jeweils das gleiche Passwort zu vergeben. (Symbolische Links zwischen den Konfigurations-Verzeichnissen führen zu Konflikten zwischen den lokalen VNC-Servern

und können nicht daher verwendet werden. Kopieren der Verzeichnisse ist hingegen unproblematisch.)

Momentan starten wir das Skript `vnc` manuell in einer `screen`-Sitzung auf `main-1`. Ein automatischer Start beim Reboot des Servers ist natürlich ebenfalls möglich.

## Anbindung an die SSH-Tunnel der Vortragenden

Bisher haben wir „nur“ dafür gesorgt, daß Teilnehmende von außen auf lokale VNC-Server auf unserem Server `main-1` zugreifen können. Nun gilt es, die lokalen VNC-Server über die Startseite hinaus mit Inhalten zu füllen. Typischerweise sind dies die von den Vortragenden per VNC exportierten Bildschirmhalte. (Natürlich ist es auch möglich, lokale X-Clients in den lokalen VNC-Servern laufen zu lassen und auf diese Weise beliebige andere Inhalte anzubieten. Tatsächlich geschieht dies bereits in Gestalt der Startseiten.)

Das hier wiedergegebene Shell-Skript `autoview` lauscht mittels `nmap` auf den für die SSH-Tunnel der Lehren vorgesehenen TCP-Ports. (`nmap` kann mit `apt install nmap` installiert werden.) Sobald der Port offen ist, startet es einen lokalen `xtightvncviewer` und übergibt über eine Pipe das von dem\*der Vortragenden festgelegte Passwort für den Lesezugriff. (Dieses braucht nicht dasselbe Passwort zu sein, das für den Lesezugriff von außen verwendet wird.)

```
#!/bin/sh

vncviewer="xtightvncviewer"
vncopts="-viewonly -depth 32 -compresslevel 9 -quality 9 \
        -geometry 1920x1080+0+0"

log="$0.log"
date_format="%Y-%m-%d %H:%M:%S"

while true; do
    ports=$(nmap localhost -p 5913-5918 \
            | grep "open" | cut -d '/' -f 1)
    if [ -z "$ports" ]; then
        echo "no VNC sessions offered"
        echo $(date +"$date_format"): "no VNC sessions offered" >> "$log"
    else
        for port in $ports; do
            incoming_channel=$(echo "$port" | cut -b 3-4)
            case $incoming_channel in
                13) channel=1; password="XXXXXXXX" ;;
                17) channel=2; password="XXXXXXXX" ;;
                16) channel=3; password="XXXXXXXX" ;;
                14) channel=4; password="XXXXXXXX" ;;
                15) channel=5; password="XXXXXXXX" ;;
                *)  channel=6; password="XXXXXXXX" ;;
            esac
            echo -n "$incoming_channel --> $channel: "
            echo -n $(date +"$date_format"): \
                "$incoming_channel --> $channel: " >> "$log"
            process=$(ps aux \
                    | grep "$vncviewer.*:$incoming_channel" \
                    | grep -v "grep")
```

```

if [ -n "$process" ]; then
    echo "vncviewer is running"
    echo "vncviewer is running" >> "$log"
else
    echo -n "start of vncviewer ..."
    echo "start of vncviewer ..." >> "$log"
    channel_directory="$HOME/vnc$channel"
    (echo "$password" \
      | DISPLAY="$channel" HOME="$channel_directory" \
      "vncviewer" :"$incoming_channel" -autopass $vncopts \
      >> "$log" 2>&1) &
    sleep 1
    echo " initiated"
    echo "$(date +"$date_format"): "initiated" >> "$log"
fi
done
fi
sleep 10
done

```

Die Passwörter für den Lesezugriff auf die VNC-Bildschirme der Vortragenden sind in dem Shell-Skript zusammen mit den jeweils zugeordneten Port-Nummern für die SSH-Tunnel im Klartext abgelegt. (In diesem Beispiel lauten die Portnummern für die SSH-Tunnel [5913](#) bis [5918](#) und sind nicht fortlaufend mit den VNC-Kanälen. Dies hat sich bei der Einführung des Systems so ergeben, ließe sich aber auf beliebige Port-Nummern ändern.)

Auch das Skript [autoview](#) starten wir z. Zt. manuell in einer [screen](#)-Sitzung auf [main-1](#). Auch hier wäre natürlich ein automatischer Start beim Reboot des Servers ebenfalls möglich.

## Konfiguration der SSH-Tunnel für die Vortragenden

Auf dem Server haben wir für alle Vortragenden gemeinsam das Benutzerkonto [ssh-tunnel](#) angelegt. Dieses hat keinen Shell-Zugriff; in der Datei [/etc/passwd](#) steht entsprechend ein `x` für das Passwort und `/bin/false` für die Shell.

Die öffentlichen Schlüssel, die uns die Vortragenden zusenden, hinterlegen wir in [/home/ssh-tunnel/.ssh/authorized\\_keys](#) und beschränken dabei den Zugriff auf den Aufbau des jeweils zugeordneten SSH-Tunnels.

```

permitlisten="localhost:5915",no-pty,no-x11-forwarding,no-agent-
forwarding ssh-rsa AAAAB3NzaC1yc2EAA[...]X== user@host

```

(Auch hier erfolgt der Umbruch nur aus Platzgründen; in der Datei muß alles in einer einzigen Zeile stehen.) In diesem Beispiel lautet die Port-Nummer für den SSH-Tunnel [5915](#), und die zweite Zeile stellt (stark verkürzt) den öffentlichen SSH-Schlüssel des\*der Vortragenden dar.

Während Sie die Vortragenden dabei unterstützen, ihre SSH-Tunnel zu konfigurieren, können Sie mit

```
lsof | grep ssh-tunnel | grep "591[0-9]"
```

prüfen, ob der SSH-Tunnel besteht. ([ssh-tunnel](#) ist das in diesem Beispiel verwendete gemeinsame Benutzerkonto für die SSH-Tunnel, und [591\[0-9\]](#) ist ein regulärer Ausdruck für die den Vortragenden zugewiesenen Port-Nummern.)

Wenn das [autoview](#)-Skript läuft (siehe oben), sollte spätestens 10 Sekunden nach Aufbau des SSH-Tunnels und Start des VNC-Servers durch den\*die Vortragende\*n der VNC-Bildschirm auf dem entsprechenden Kanal zu sehen sein.

## VNC ohne SSH-Tunnel

In der bisher beschriebenen Konfiguration stellt grundsätzlich der VNC-Viewer eine Verbindung zu einem VNC-Server her, der einen Bildschirminhalt exportiert. VNC läßt aber auch Verbindungen in der umgekehrten Richtung zu: Ein VNC-Viewer lauscht auf einem Port, und ein VNC-Server exportiert sein Bild dorthin. (In TightVNC für MS-Windows heißt der entsprechende Menüpunkt „Attach Listening Viewer...“; bei [x11vnc](#) heißt die Kommandozeilen-Option `-connect`.)

Um dies zu ermöglichen, genügt es, auf den Kanälen, für die dies beabsichtigt ist, einen [xtightvncviewer](#) lauschen zu lassen:

```
for x in 5 6; do
    (HOME=$HOME/vnc$x DISPLAY=: $x xtightvncviewer \
    -depth 24 -geometry 1920x1080+0+0 -listen &); \
done
```

(In diesem Beispiel geschieht dies für die Kanäle 5 und 6.)

Die Angabe des Home-Verzeichnisses ist notwendig, damit sich [xtightvncviewer](#) auch ohne Passwort-Abfrage mit dem lokalen VNC-Server verbinden kann. (Stattdessen könnte man auch [xtightvncviewer](#) mit der Option `-autopass` starten und das Passwort durch eine Pipe übergeben.)

Die Angabe der `DISPLAY`-Variablen teilt dem VNC-Viewer mit, auf welchem der lokalen VNC-Server das Bild zu sehen sein soll. (Der lokale VNC-Server tritt hier nicht als VNC-Server auf, sondern als X-Server.)

Auch dieses Programm lassen wir z. Zt. der Einfachheit halber in einer [screen](#)-Sitzung laufen. Ein automatischer Start wäre natürlich auch hier möglich.

**Warnung:** Im Gegensatz zu allen anderen in dieser Anleitung beschriebenen Kommunikationsformen zwischen dem Server und allen, die ihn benutzen, ist diese Methode, ein VNC-Bild zu übertragen, **unverschlüsselt**.

Sie sollte daher nur in Fällen zur Anwendung kommen, in denen (a) der Aufwand, SSH-Tunnel für die Vortragenden einzurichten, zu hoch wäre, (b) es nicht auf Vertraulichkeit ankommt und (c) nötigenfalls andere Methoden existieren, Authentizität zu gewährleisten. (Beispiel: Während einer Übung wollen Studierende zur Diskussion ihrer Lösungen ihre eigenen Bildschirmhalte veröffentlichen.)

## Websockify

Der im folgenden beschriebene Schritt ist möglicherweise nicht notwendig.

Während unserer ersten Lasttests traten Instabilitäten auf. Diese waren begleitet von Fehlermeldungen des Programms [websockify](#), die sich auf einen lange bekannten Fehler in diesem Programm zurückführen ließen, und sich auch durch ein Upgrade von der Python-2- auf die Python-3-Version nicht beseitigen ließen.



Wir haben daraufhin die Python-Version von `websockify` durch eine Node-JS-Version ersetzt. Diese kann man unter <https://raw.githubusercontent.com/novnc/websockify-js/master/websockify/websockify.js> herunterladen.

Die Instabilitäten wurden dadurch weniger, verschwanden aber erst völlig, nachdem wir für die lokalen VNC-Server `tightvncserver` durch `tigervncserver` ersetzt hatten. Dies kann bedeuten, daß die Instabilitäten überhaupt nichts mit `websockify` zu tun hatten. Weitere Untersuchungen sind geplant.

Um `websockify.js` ausführen zu können, müssen der Node-JS-Interpreter und die benötigten Bibliotheken installiert werden:

```
apt install node node-optimist node-mime-types
```

Damit `noVNC` das Node-JS-Programm `websockify.js` aufrufen kann, verwenden wir ein Shell-Skript `/usr/local/bin/websockify` als Wrapper:

```
#!/bin/sh
node /usr/local/bin/websockify.js "$@"
```

Sollte sich in zukünftigen Lasttests herausstellen, daß der hier beschriebene Austausch von `websockify` doch nicht benötigt wird, werden wir diese Anleitung entsprechend aktualisieren.

## Zum Nachtisch: tmux

Fortgeschrittenen Studierenden Informatik-naher Studiengänge kann man auch eine gemeinsame Shell für den Lesezugriff anbieten.

Hierfür lege man ein „normales“ Benutzerkonto `lecture` an und hinterlege die öffentlichen Schlüssel der Studierenden wie folgt in der Datei `~lecture/.ssh/authorized_keys`:

```
command="tmux attach -r",restrict,pty
ssh-rsa AAAAB3NzaC1yc2EAAA[...]Q== user@host
```

(Wie immer gilt: kein Umbruch, sondern eine lange Zeile pro Schlüssel.)

Anschließend können sich die Studierenden mit einem vorher von der\*dem Vortragenden unter demselben Benutzerkonto gestarteten `tmux` lesend verbinden.

*Viel Erfolg!*

Stand: 14. April 2020

Copyright © 2020 Peter Gerwinski, Benedikt Wildenhain

Lizenz: CC-by-sa (Version 3.0) oder GNU GPL (Version 3 oder höher)

Sie können diese Anleitung einschließlich  $\LaTeX$ -Quelltext herunterladen unter:  
<https://gitlab.cvh-server.de/pgerwinski/ow>