Wireguard Site2Site Tunnel

- Site2Site Tunnel mit nur einer Öffentlichen statischen ip auf einer Seite

Site2Site Tunnel mit nur einer Öffentlichen statischen ip auf einer Seite

Beschreibung:

Es gibt Situationen da möchte man zwei Netzwerke miteinander verbinden.
hier zu ein kleines Script das zwei Wireguard configs erstellt.

Das Script:

```
nano make_wg_s2s.py
```

Inhalt

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import argparse
import base64
import ipaddress
import os
import shutil
import subprocess
import sys
from typing import List, Optional, Tuple

def eprint(*args, **kwargs):
    print(*args, file=sys.stderr, **kwargs)

# ---------- Key generation ----------
```

```
def have_cmd(cmd: str) -> bool:
    return shutil.which(cmd) is not None

def gen_keys_with_wg() -> Tuple[str, str]:
    priv = subprocess.check_output(["wg", "genkey"], text=True).strip()
    pub = subprocess.check_output(["wg", "pubkey"], input=priv, text=True).strip()
    if not priv or not pub:
        raise RuntimeError("wg returned empty key(s)")
    return priv, pub

def gen_keys_with_pynacl_or_crypto() -> Tuple[str, str]:
    try:
        from nacl.public import PrivateKey
        sk = PrivateKey.generate()
        priv_b = bytes(sk._private_key)
        pub_b = bytes(sk.public_key._public_key)
        return base64.b64encode(priv_b).decode(), base64.b64encode(pub_b).decode()
    except Exception:
        from cryptography.hazmat.primitives.asymmetric.x25519 import X25519PrivateKey
        from cryptography.hazmat.primitives import serialization
        sk = X25519PrivateKey.generate()
        priv_b = sk.private_bytes(
            encoding=serialization.Encoding.Raw,
            format=serialization.PrivateFormat.Raw,
            encryption_algorithm=serialization.NoEncryption(),
        )
        pub_b = sk.public_key().public_bytes(
            encoding=serialization.Encoding.Raw,
            format=serialization.PublicFormat.Raw
        )
        return base64.b64encode(priv_b).decode(), base64.b64encode(pub_b).decode()

def gen_keypair() -> Tuple[str, str]:
    if have_cmd("wg"):
        try:
            return gen_keys_with_wg()
        except Exception as e:
            eprint(f"Warnung: wg keygen fehlgeschlagen: {e}")
```

```
    # Fallback
    try:
        return gen_keys_with_pynacl_or_crypto()
    except Exception as e:
        eprint("Fehler: Konnte keinen Schlüssel erzeugen. Installiere 'wireguard-tools' ODER 'pynacl' ODER
'cryptography'.")
        eprint(f"Details: {e}")
        sys.exit(3)

# ---------- Helpers ----------

def parse_ip(addr: str, allow_v6: bool) -> ipaddress._BaseAddress:
    ip = ipaddress.ip_address(addr)
    if ip.version == 6 and not allow_v6:
        raise ValueError("IPv6 angegeben, aber --ipv6 nicht gesetzt.")
    return ip

def parse_cidr_list(text: str, allow_v6: bool) -> List[str]:
    nets = []
    for token in [t.strip() for t in text.split(",") if t.strip()]:
        net = ipaddress.ip_network(token, strict=False)
        if net.version == 6 and not allow_v6:
            raise ValueError("IPv6-Netz angegeben, aber --ipv6 nicht gesetzt.")
        nets.append(str(net))
    return nets

def ensure_single_host_cidr(ip: str) -> str:
    ipobj = ipaddress.ip_address(ip)
    return f"{ip}/{32 if ipobj.version == 4 else 128}"

def prompt(msg: str, default: Optional[str]=None, required: bool=True) -> str:
    suffix = f" [{default}]" if default else ""
    while True:
        val = input(f"{msg}{suffix}: ").strip()
        if not val and default is not None:
            return default
        if val or not required:
            return val
        print("Bitte Wert eingeben.")
```

```
def uniq(seq: List[str]) -> List[str]:
    seen = set()
    out = []
    for x in seq:
        if x not in seen:
            seen.add(x)
            out.append(x)
    return out

# ---------- CLI ----------

def build_args() -> argparse.Namespace:
    p = argparse.ArgumentParser(
        description="Erstellt WireGuard Site-to-Site Configs (nur eine Seite öffentlich erreichbar).",
        formatter_class=argparse.ArgumentDefaultsHelpFormatter
    )
    p.add_argument("--name", help="Projektname (Verzeichnis wird angelegt)")
    p.add_argument("--public-endpoint", help="Domain oder IP der öffentlichen Seite (z.B. vpn.example.com)")
    #p.add_argument("--public-port", type=int, default=51820, help="UDP-Port der öffentlichen Seite")
    p.add_argument("--public-port", type=int, default=None, help="UDP-Port der öffentlichen Seite (Standard:
51820)")
    p.add_argument("--public-wg-ip4", help="WG-IPv4 der öffentlichen Seite (z.B. 10.10.0.1)")
    p.add_argument("--private-wg-ip4", help="WG-IPv4 der privaten Seite (leer lassen um keine Address zu
setzen)")
    p.add_argument("--public-lans", help="Komma-Liste LAN-Netze hinter öffentlicher Seite (z.B.
192.168.10.0/24,192.168.11.0/24)")
    p.add_argument("--private-lans", help="Komma-Liste LAN-Netze hinter privater Seite")
    p.add_argument("--ipv6", action="store_true", help="IPv6 zusätzlich konfigurieren")
    p.add_argument("--public-wg-ip6", help="WG-IPv6 der öffentlichen Seite (z.B. fd00:10:10::1)")
    p.add_argument("--private-wg-ip6", help="WG-IPv6 der privaten Seite (leer lassen erlaubt)")
    p.add_argument("-v", "--verbose", action="store_true", help="Mehr Ausgaben")
    return p.parse_args()

# ---------- Main ----------

def main():
    print("Start...")
    args = build_args()
```

```
if args.verbose:
    print("Argumente empfangen:", args)

# Eingaben einsammeln
proj = args.name or prompt("Projektname")
base_dir = os.path.abspath(proj)
if os.path.exists(base_dir):
    eprint(f"Projekt '{proj}' gibt’s schon – nichts überschrieben.")
    sys.exit(1)

public_endpoint = args.public_endpoint or prompt("Public Endpoint (Domain oder IP)")
# NEU:
if args.public_port is None:
    port_str = prompt("Public UDP-Port", default="51820", required=False)
    public_port = int(port_str or "51820")
else:
    public_port = args.public_port

public_wg_ip4 = args.public_wg_ip4 or prompt("WG-IPv4 der öffentlichen Seite (z.B. 10.10.0.1)")
private_wg_ip4 = args.private_wg_ip4
if private_wg_ip4 is None:
    private_wg_ip4 = prompt("WG-IPv4 der privaten Seite (leer = keine Address)", required=False)

public_lans = parse_cidr_list(args.public_lans, allow_v6=args.ipv6) if args.public_lans else []
if not args.public_lans:
    val = prompt("LAN-Netze öff. Seite (Komma, leer für keine)", required=False)
    if val:
        public_lans = parse_cidr_list(val, allow_v6=args.ipv6)

private_lans = parse_cidr_list(args.private_lans, allow_v6=args.ipv6) if args.private_lans else []
if not args.private_lans:
    val = prompt("LAN-Netze private Seite (Komma, leer für keine)", required=False)
    if val:
        private_lans = parse_cidr_list(val, allow_v6=args.ipv6)

v6_enabled = bool(args.ipv6)
public_wg_ip6 = args.public_wg_ip6
private_wg_ip6 = args.private_wg_ip6
if v6_enabled:
```

```
    public_wg_ip6 = public_wg_ip6 or prompt("WG-IPv6 öff. Seite (z.B. fd00:10:10::1)")
    if private_wg_ip6 is None:
        private_wg_ip6 = prompt("WG-IPv6 private Seite (leer = keine Address)", required=False)

# Validierung
try:
    parse_ip(public_wg_ip4, allow_v6=False)
    if private_wg_ip4:
        parse_ip(private_wg_ip4, allow_v6=False)
    if v6_enabled:
        parse_ip(public_wg_ip6, allow_v6=True)
        if private_wg_ip6:
            parse_ip(private_wg_ip6, allow_v6=True)
except Exception as e:
    eprint(f"IP-Fehler: {e}")
    sys.exit(2)

# Dirs
pub_dir = os.path.join(base_dir, "public")
prv_dir = os.path.join(base_dir, "private")
os.makedirs(pub_dir, exist_ok=False)
os.makedirs(prv_dir, exist_ok=False)

if args.verbose:
    print("Projektverzeichnis:", base_dir)

# Keys
print("Erzeuge Schlüsselpaare...")
server_priv, server_pub = gen_keypair()   # öffentliche Seite
client_priv, client_pub = gen_keypair()   # private Seite

# Interface-Adressen
iface_addrs_public = [f"{public_wg_ip4}/32"]
iface_addrs_private: List[str] = []
if private_wg_ip4:
    iface_addrs_private.append(f"{private_wg_ip4}/32")

peer_allowed_from_client = [ensure_single_host_cidr(public_wg_ip4)]
peer_allowed_from_server = []
```

```
if private_wg_ip4:
    peer_allowed_from_server.append(ensure_single_host_cidr(private_wg_ip4))

if v6_enabled:
    iface_addrs_public.append(f"{public_wg_ip6}/128")
    if private_wg_ip6:
        iface_addrs_private.append(f"{private_wg_ip6}/128")
    peer_allowed_from_client.append(ensure_single_host_cidr(public_wg_ip6))
    if private_wg_ip6:
        peer_allowed_from_server.append(ensure_single_host_cidr(private_wg_ip6))

# LANs ergänzen
if private_lans:
    peer_allowed_from_server.extend(private_lans)
if public_lans:
    peer_allowed_from_client.extend(public_lans)

peer_allowed_from_client = uniq(peer_allowed_from_client)
peer_allowed_from_server = uniq(peer_allowed_from_server)

# Config rendern
public_conf = "[Interface]\n"
public_conf += f"Address = {', '.join(iface_addrs_public)}\n"
public_conf += f"ListenPort = {public_port}\n"
public_conf += f"PrivateKey = {server_priv}\n"
public_conf += "# SaveConfig = false\n\n"
public_conf += "[Peer]\n"
public_conf += f"PublicKey = {client_pub}\n"
if peer_allowed_from_server:
    public_conf += f"AllowedIPs = {', '.join(peer_allowed_from_server)}\n"
public_conf += "PersistentKeepalive = 25\n"

private_conf = "[Interface]\n"
if iface_addrs_private:
    private_conf += f"Address = {', '.join(iface_addrs_private)}\n"
private_conf += f"PrivateKey = {client_priv}\n"
private_conf += "# SaveConfig = false\n\n"
private_conf += "[Peer]\n"
private_conf += f"PublicKey = {server_pub}\n"
```

```
    private_conf += f"Endpoint = {public_endpoint}:{public_port}\n"
    private_conf += f"AllowedIPs = {', '.join(peer_allowed_from_client)}\n"
    private_conf += "PersistentKeepalive = 25\n"

    # Schreiben
    with open(os.path.join(pub_dir, "wg0.conf"), "w") as f:
        f.write(public_conf)
    with open(os.path.join(prv_dir, "wg0.conf"), "w") as f:
        f.write(private_conf)

    readme = (
        f"# {proj} – WireGuard Site-to-Site\n\n"
        "Dieses Projekt erzeugt zwei Konfigurationen:\n\n"
        f"- public/wg0.conf  → Öffentliche Seite (ListenPort: {public_port}, Endpoint: {public_endpoint})\n"
        "- private/wg0.conf → Private Seite (hinter NAT; verbindet aktiv zum Endpoint)\n\n"
        "Start (Beispiel):\n"
        "  wg-quick up wg0\n\n"
        "Tipps:\n"
        "  IPv4 Forwarding:  sysctl -w net.ipv4.ip_forward=1\n"
        "  IPv6 Forwarding:  sysctl -w net.ipv6.conf.all.forwarding=1\n"
        "  Routen/Firewall: LANs erlauben/setzen.\n"
    )
    with open(os.path.join(base_dir, "README.md"), "w") as f:
        f.write(readme)

    print("Fertig!")
    print("  -", os.path.join(pub_dir, "wg0.conf"))
    print("  -", os.path.join(prv_dir, "wg0.conf"))
    print("Nichts wurde überschrieben.")

if __name__ == "__main__":
    try:
        main()
    except SystemExit as se:
        # argparse nutzt SystemExit – durchreichen
        raise
    except Exception as ex:
        eprint("Unerwarteter Fehler:", ex)
        import traceback
```

```
        traceback.print_exc()
        sys.exit(99)
```

Verwendung:

1) **Interaktiv** (alles wird abgefragt)

```
python3 make_wg_s2s.py
```

2) **Parameter gesteuert für ipv4**

```
python3 make_wg_s2s.py \
  --name firma-tunnel \
  --public-endpoint vpn.example.com \
  --public-port 51820 \
  --public-wg-ip4 10.10.0.1 \
  --private-wg-ip4 10.10.0.2 \
  --public-lans 192.168.10.0/24 \
  --private-lans 192.168.20.0/24
```

3) **Parameter gesteuert für ipv6**

```
python3 make_wg_s2s.py \
  --name firma-v6 \
  --public-endpoint v6.example.com \
  --public-port 51820 \
  --public-wg-ip4 10.10.0.1 \
  --private-wg-ip4 10.10.0.2 \
  --ipv6 \
  --public-wg-ip6 fd00:10:10::1 \
  --private-wg-ip6 fd00:10:10::2 \
  --public-lans 192.168.10.0/24,fd00:aaaa::/64 \
  --private-lans 192.168.20.0/24,fd00:bbbb::/64
```