Privacy-Preserving Network Configurations

Table of Contents

Anonymous Network Access

Tor Configuration with Enhanced Privacy

bash

```
#!/bin/bash
# tor_enhanced_setup.sh - Enhanced Tor configuration for maximum privacy

# Install Tor and related tools
install_tor_suite() {
    sudo apt-get update
    sudo apt-get install -y tor torsocks proxychains4 obfs4proxy

    # Install Tor Browser
    wget -q -O - https://deb.torproject.org/torproject.org/A3C4F0F979CAA22CDBA8F512EE8CBC9E886DDD89.asc | sudo
    echo "deb https://deb.torproject.org/torproject.org $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/tor.list
    sudo apt-get update
    sudo apt-get install -y tor-browser
}

# Configure Tor with bridges and enhanced settings
configure_tor() {
    sudo tee /etc/tor/torrc << 'EOF'
# Tor Configuration for Maximum Privacy

# Basic settings
SocksPort 9050
ControlPort 9051
CookieAuthentication 1

# Use bridges to avoid Tor blocking
UseBridges 1
ClientTransportPlugin obfs4 exec /usr/bin/obfs4proxy
Bridge obfs4 [BRIDGE_LINE_1]
Bridge obfs4 [BRIDGE_LINE_2]
Bridge obfs4 [BRIDGE_LINE_3]

# Enhanced privacy settings
AvoidDiskWrites 1
DisableAllSwap 1

# Circuit isolation
IsolateDestAddr 1
IsolateDestPort 1
IsolateSOCKSAuth 1
IsolateClientProtocol 1
IsolateClientAddr 1

# Restrict exit nodes to privacy-friendly countries
ExitNodes {ch},{is},{ro},{se}
StrictNodes 1

# Exclude problematic nodes
ExcludeNodes {us},{gb},{ca},{au},{nz},{cn},{ru}
ExcludeExitNodes {us},{gb},{ca},{au},{nz},{cn},{ru}

# Performance and security
NumEntryGuards 3
NumDirectoryGuards 3
GuardLifetime 2 months
CircuitBuildTimeout 60
LearnCircuitBuildTimeout 0
CircuitStreamTimeout 60
ClientOnly 1

# Disable dangerous features
ClientUseIPv6 0
ClientPreferIPv6ORPort 0
DNSPort 5353
AutomapHostsOnResolve 1
AutomapHostsSuffixes .exit,.onion

# Additional security
SafeLogging 1
TestSocks 1
WarnPlaintextPorts 23,109,110,143,80

# Hidden service client settings
```

```
HiddenServiceDir /var/lib/tor/hidden_service/
HiddenServicePort 80 127.0.0.1:8080
HiddenServicePort 22 127.0.0.1:22

# Stream isolation for different applications
SocksPort 9050 # Default
SocksPort 9051 IsolateDestAddr IsolateDestPort # Browser
SocksPort 9052 IsolateDestAddr IsolateDestPort # Email
SocksPort 9053 IsolateDestAddr IsolateDestPort # IM
SocksPort 9054 IsolateDestAddr IsolateDestPort # Other

# Bandwidth limits (optional)
# RelayBandwidthRate 100 KB
# RelayBandwidthBurst 200 KB
EOF

    # Get bridges from https://bridges.torproject.org/
    echo "Please get bridge lines from https://bridges.torproject.org/ and add them to /etc/tor/torrc"

    # Restart Tor
    sudo systemctl restart tor
}

# Configure system-wide proxy through Tor
configure_system_proxy() {
    # Configure proxychains
    sudo tee /etc/proxychains4.conf << 'EOF'
# ProxyChains Configuration

# Dynamic chain - Each connection will be done via chained proxies
dynamic_chain

# Quiet mode
quiet_mode

# Proxy DNS requests
proxy_dns

# Timeouts
tcp_read_time_out 15000
tcp_connect_time_out 8000

# ProxyList
[ProxyList]
# Tor
socks5 127.0.0.1 9050
# Additional proxies can be added here for chaining
# socks5 127.0.0.1 9051
# http 127.0.0.1 8118
EOF

    # Configure environment variables
    cat >> ~/.bashrc << 'EOF'
# Tor proxy settings
export ALL_PROXY="socks5://127.0.0.1:9050"
export HTTP_PROXY="socks5://127.0.0.1:9050"
export HTTPS_PROXY="socks5://127.0.0.1:9050"
export NO_PROXY="localhost,127.0.0.1,192.168.0.0/16,10.0.0.0/8"

# Alias for torified commands
alias torssh='torsocks ssh'
alias torcurl='torsocks curl'
alias torwget='torsocks wget'
alias torgit='torsocks git'
EOF
}

# Setup transparent proxy (route all traffic through Tor)
setup_transparent_proxy() {
    # Create Tor user
    sudo useradd -r -s /bin/false debian-tor 2>/dev/null

    # Configure iptables for transparent proxy
```

```
    sudo tee /usr/local/bin/tor-transparent.sh << 'EOF'
#!/bin/bash

# Tor transparent proxy setup
TOR_UID=$(id -u debian-tor)
TOR_PORT="9040"
TOR_DNS="5353"
INT_IF="eth0"

# Flush existing rules
iptables -t nat -F
iptables -t filter -F

# Allow established connections
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# Allow loopback
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# Allow Tor user
iptables -t nat -A OUTPUT -m owner --uid-owner $TOR_UID -j RETURN
iptables -A OUTPUT -m owner --uid-owner $TOR_UID -j ACCEPT

# Redirect DNS to Tor
iptables -t nat -A OUTPUT -p udp --dport 53 -j REDIRECT --to-ports $TOR_DNS
iptables -t nat -A OUTPUT -p tcp --dport 53 -j REDIRECT --to-ports $TOR_DNS

# Redirect all TCP traffic to Tor
iptables -t nat -A OUTPUT -p tcp --syn -j REDIRECT --to-ports $TOR_PORT

# Block all non-Tor traffic
iptables -A OUTPUT -m owner ! --uid-owner $TOR_UID -j DROP

echo "Transparent proxy enabled - all traffic now routed through Tor"
EOF

    sudo chmod +x /usr/local/bin/tor-transparent.sh
}

# Check Tor connection
check_tor_connection() {
    echo "Checking Tor connection..."

    # Check if Tor is running
    if systemctl is-active --quiet tor; then
        echo "✓ Tor service is running"
    else
        echo "✗ Tor service is not running"
        return 1
    fi

    # Check external IP through Tor
    echo "Your IP without Tor:"
    curl -s https://api.ipify.org
    echo -e "\n\nYour IP through Tor:"
    torsocks curl -s https://api.ipify.org
    echo -e "\n"

    # Check Tor circuit
    echo "Current Tor circuit:"
    echo "GETINFO circuit-status" | nc 127.0.0.1 9051
}

# Main execution
install_tor_suite
configure_tor
configure_system_proxy
setup_transparent_proxy
check_tor_connection
```

I2P Network Configuration

bash

```
#!/bin/bash
# i2p_setup.sh - I2P anonymous network setup

# Install I2P
install_i2p() {
    # Add I2P repository
    sudo apt-add-repository ppa:i2p-maintainers/i2p
    sudo apt-get update
    sudo apt-get install -y i2p i2p-keyring

    # Configure I2P as service
    sudo systemctl enable i2p
    sudo systemctl start i2p
}

# Configure I2P for enhanced privacy
configure_i2p() {
    I2P_CONFIG="/var/lib/i2p/i2p-config"

    # Configure bandwidth limits
    cat > $I2P_CONFIG/router.config << EOF
# I2P Router Configuration
router.sharePercentage=80
router.inboundKBytesPerSecond=256
router.outboundKBytesPerSecond=256
router.floodfillParticipant=false

# Enhanced privacy settings
router.hideFloodfillParticipant=true
router.maxParticipatingTunnels=100
router.maxMemory=512

# Network settings
i2np.udp.enable=true
i2np.udp.port=8887
i2np.tcp.enable=true
i2np.tcp.port=8887

# Tunnel settings
router.inboundTunnelCount=3
router.outboundTunnelCount=3
router.inboundTunnelLength=3
router.outboundTunnelLength=3
EOF

    # Configure HTTP proxy for I2P
    cat > $I2P_CONFIG/clients.config << EOF
# I2P Client Configuration
clientApp.0.main=net.i2p.i2ptunnel.TunnelControllerGroup
clientApp.0.name=I2P HTTP Proxy
clientApp.0.args=i2ptunnel.config

# HTTP Proxy on port 4444
tunnel.0.name=I2P HTTP Proxy
tunnel.0.type=httpclient
tunnel.0.interface=127.0.0.1
tunnel.0.listenPort=4444

# HTTPS Proxy on port 4445
tunnel.1.name=I2P HTTPS Proxy
tunnel.1.type=httpsclient
tunnel.1.interface=127.0.0.1
tunnel.1.listenPort=4445

# SOCKS proxy on port 4447
tunnel.2.name=SOCKS 4/4a/5 Proxy
tunnel.2.type=sockstunnel
tunnel.2.interface=127.0.0.1
tunnel.2.listenPort=4447
EOF

    # Restart I2P
    sudo systemctl restart i2p
```

```
}

# Setup browser for I2P
configure_browser_i2p() {
    # Firefox profile for I2P
    cat > ~/i2p-browser.sh << 'EOF'
#!/bin/bash
# Launch Firefox with I2P configuration

firefox -no-remote -CreateProfile "I2P"
firefox -P "I2P" \
    -pref "network.proxy.type:1" \
    -pref "network.proxy.http:127.0.0.1" \
    -pref "network.proxy.http_port:4444" \
    -pref "network.proxy.ssl:127.0.0.1" \
    -pref "network.proxy.ssl_port:4445" \
    -pref "network.proxy.socks:127.0.0.1" \
    -pref "network.proxy.socks_port:4447" \
    -pref "network.proxy.socks_remote_dns:true" \
    -pref "network.proxy.no_proxies_on:127.0.0.1,localhost" \
    -pref "browser.startup.homepage:http://127.0.0.1:7657/home"
EOF
    chmod +x ~/i2p-browser.sh
}
```

DNS Privacy Configuration

DNS over HTTPS (DoH) and DNS over TLS (DoT)

```
bash
```

```
#!/bin/bash
# dns_privacy.sh - Configure private DNS resolution

# Setup Stubby for DNS over TLS
setup_stubby() {
    sudo apt-get install -y stubby

    # Configure Stubby
    sudo tee /etc/stubby/stubby.yml << 'EOF'
# Stubby configuration for DNS over TLS

resolution_type: GETDNS_RESOLUTION_STUB
dns_transport_list:
  - GETDNS_TRANSPORT_TLS
tls_authentication: GETDNS_AUTHENTICATION_REQUIRED
tls_query_padding_blocksize: 256
edns_client_subnet_private: 1

idle_timeout: 10000
round_robin_upstreams: 1

# Upstream DNS servers
upstream_recursive_servers:
  # Quad9 (Security)
  - address_data: 9.9.9.9
    tls_auth_name: "dns.quad9.net"
    tls_pubkey_pinset:
      - digest: "sha256"
        value: /SlsviBkb05Y/8XiKF9+CZsgCtrqPQk5bh47o0R3/Cg=

  # Cloudflare
  - address_data: 1.1.1.1
    tls_auth_name: "cloudflare-dns.com"
    tls_pubkey_pinset:
      - digest: "sha256"
        value: GP8Knf7qBae+alfythytMbYnL+yowaWVeD6MoLHkVRg=

  # NextDNS
  - address_data: 45.90.28.0
    tls_auth_name: "dns.nextdns.io"

  # AdGuard DNS
  - address_data: 94.140.14.14
    tls_auth_name: "dns.adguard.com"

listen_addresses:
  - 127.0.0.1@53
  - 0::1@53
EOF

    # Configure system to use Stubby
    sudo systemctl stop systemd-resolved
    sudo systemctl disable systemd-resolved

    # Update resolv.conf
    sudo tee /etc/resolv.conf << EOF
nameserver 127.0.0.1
options edns0
EOF

    # Make resolv.conf immutable
    sudo chattr +i /etc/resolv.conf

    # Start Stubby
    sudo systemctl enable stubby
    sudo systemctl start stubby
}

# Setup dnscrypt-proxy for enhanced DNS privacy
setup_dnscrypt_proxy() {
    # Download and install dnscrypt-proxy
    cd /tmp
    wget https://github.com/DNSCrypt/dnscrypt-proxy/releases/latest/download/dnscrypt-proxy-linux_x86_64.tar.gz
```

```
    tar -xzf dnscrypt-proxy-linux_x86_64.tar.gz
    sudo mv linux-x86_64/dnscrypt-proxy /usr/local/bin/

    # Create configuration directory
    sudo mkdir -p /etc/dnscrypt-proxy

    # Configure dnscrypt-proxy
    sudo tee /etc/dnscrypt-proxy/dnscrypt-proxy.toml << 'EOF'
# DNSCrypt-Proxy Configuration

listen_addresses = ['127.0.0.1:53', '[::1]:53']
max_clients = 250
ipv4_servers = true
ipv6_servers = false
dnscrypt_servers = true
doh_servers = true
odoh_servers = true

# Require DNSSEC
require_dnssec = true
require_nolog = true
require_nofilter = true

# Server selection
server_names = [
  'cloudflare',
  'quad9-dnscrypt-ip4-filter-pri',
  'nextdns',
  'adguard-dns-doh'
]

# Anonymized DNS
[anonymized_dns]
routes = [
  { server_name='cloudflare', via=['anon-cs-fr', 'anon-cs-nl'] },
  { server_name='quad9-dnscrypt-ip4-filter-pri', via=['anon-cs-de', 'anon-cs-uk'] }
]

# Query logging (disabled for privacy)
[query_log]
file = '/dev/null'
format = 'tsv'

# Pattern blocking
[blocked_names]
blocked_names_file = '/etc/dnscrypt-proxy/blocked-names.txt'

# IP blocking
[blocked_ips]
blocked_ips_file = '/etc/dnscrypt-proxy/blocked-ips.txt'

# Cloaking rules (for local domains)
[cloaking_rules]
cloaking_rules_file = '/etc/dnscrypt-proxy/cloaking-rules.txt'

# Cache
cache = true
cache_size = 4096
cache_min_ttl = 2400
cache_max_ttl = 86400
cache_neg_min_ttl = 60
cache_neg_max_ttl = 600

# Additional security
tls_disable_session_tickets = true
tls_cipher_suite = [52392, 49199]
EOF

    # Create systemd service
    sudo tee /etc/systemd/system/dnscrypt-proxy.service << EOF
[Unit]
Description=DNSCrypt-proxy client
Documentation=https://github.com/DNSCrypt/dnscrypt-proxy
```

```
After=network.target
Before=nss-lookup.target
Wants=nss-lookup.target

[Service]
Type=simple
NonBlocking=true
ExecStart=/usr/local/bin/dnscrypt-proxy -config /etc/dnscrypt-proxy/dnscrypt-proxy.toml
Restart=always
RestartSec=10
StandardOutput=journal
StandardError=journal

[Install]
WantedBy=multi-user.target
EOF

    # Start dnscrypt-proxy
    sudo systemctl daemon-reload
    sudo systemctl enable dnscrypt-proxy
    sudo systemctl start dnscrypt-proxy
}

# Setup Unbound with DNS over TLS
setup_unbound() {
    sudo apt-get install -y unbound

    # Configure Unbound
    sudo tee /etc/unbound/unbound.conf.d/privacy.conf << 'EOF'
server:
    # Network settings
    interface: 127.0.0.1
    interface: ::1
    port: 53
    do-ip4: yes
    do-ip6: yes
    do-udp: yes
    do-tcp: yes

    # Privacy settings
    hide-identity: yes
    hide-version: yes
    qname-minimisation: yes
    qname-minimisation-strict: yes
    minimal-responses: yes

    # Security settings
    harden-glue: yes
    harden-dnssec-stripped: yes
    harden-algo-downgrade: yes
    harden-large-queries: yes
    harden-short-bufsize: yes
    use-caps-for-id: yes

    # DNSSEC
    val-clean-additional: yes
    val-permissive-mode: no
    val-log-level: 1

    # Cache settings
    cache-min-ttl: 3600
    cache-max-ttl: 86400
    prefetch: yes
    prefetch-key: yes

    # Privacy-preserving features
    rrset-roundrobin: yes
    minimal-responses: yes

    # Deny ANY queries
    deny-any: yes

    # Rate limiting
```

```
    ratelimit: 1000

    # Access control
    access-control: 127.0.0.0/8 allow
    access-control: ::1/128 allow
    access-control: 192.168.0.0/16 allow
    access-control: 10.0.0.0/8 allow

    # Private addresses
    private-address: 192.168.0.0/16
    private-address: 10.0.0.0/8
    private-address: 172.16.0.0/12
    private-address: fd00::/8
    private-address: fe80::/10

# Forward zones for DNS over TLS
forward-zone:
    name: "."
    forward-tls-upstream: yes

    # Quad9
    forward-addr: 9.9.9.9@853#dns.quad9.net
    forward-addr: 149.112.112.112@853#dns.quad9.net

    # Cloudflare
    forward-addr: 1.1.1.1@853#cloudflare-dns.com
    forward-addr: 1.0.0.1@853#cloudflare-dns.com

    # NextDNS
    forward-addr: 45.90.28.0@853#dns.nextdns.io
    forward-addr: 45.90.30.0@853#dns.nextdns.io
EOF

    # Restart Unbound
    sudo systemctl restart unbound
}

# Test DNS privacy
test_dns_privacy() {
    echo "Testing DNS Privacy Configuration..."

    # Test DNS resolution
    echo "Testing DNS resolution:"
    dig +short example.com @127.0.0.1

    # Test DNSSEC
    echo -e "\nTesting DNSSEC validation:"
    dig +dnssec example.com @127.0.0.1 | grep -E "ad|RRSIG"

    # Test DNS leak
    echo -e "\nChecking for DNS leaks:"
    curl -s https://bash.ws/dnsleak | bash

    # Check DNS over TLS
    echo -e "\nChecking DNS over TLS connection:"
    openssl s_client -connect 1.1.1.1:853 -servername cloudflare-dns.com < /dev/null
}

# Main execution
setup_stubby
setup_dnscrypt_proxy
setup_unbound
test_dns_privacy
```

VPN & Proxy Chains

Multi-Hop VPN Configuration

```
python
```

```python
#!/usr/bin/env python3
# multihop_vpn.py - Multi-hop VPN chain manager

import os
import subprocess
import json
import random
import time
from typing import List, Dict
import logging

class MultiHopVPN:
    def __init__(self):
        self.config_dir = "/etc/vpn_configs"
        self.active_connections = []
        self.setup_logging()

    def setup_logging(self):
        logging.basicConfig(
            level=logging.INFO,
            format='%(asctime)s - %(levelname)s - %(message)s'
        )
        self.logger = logging.getLogger(__name__)

    def create_vpn_chain(self, hops: int = 3) -> bool:
        """Create multi-hop VPN chain"""
        vpn_servers = self.get_available_servers()

        if len(vpn_servers) < hops:
            self.logger.error(f"Not enough VPN servers for {hops} hops")
            return False

        # Select servers from different countries
        selected_servers = self.select_diverse_servers(vpn_servers, hops)

        # Connect to each VPN in sequence
        for i, server in enumerate(selected_servers):
            self.logger.info(f"Connecting hop {i+1}: {server['country']} - {server['city']}")

            if i == 0:
                # First hop - direct connection
                success = self.connect_vpn(server, namespace=None)
            else:
                # Subsequent hops - through network namespace
                namespace = f"vpn_hop_{i}"
                self.create_namespace(namespace)
                success = self.connect_vpn(server, namespace=namespace)

                if success:
                    # Route previous namespace through this one
                    self.route_through_namespace(f"vpn_hop_{i-1}", namespace)

            if not success:
                self.logger.error(f"Failed to connect hop {i+1}")
                self.cleanup_connections()
                return False

            self.active_connections.append({
                'server': server,
                'namespace': namespace if i > 0 else None,
                'hop': i + 1
            })

        self.logger.info(f"Successfully created {hops}-hop VPN chain")
        return True

    def get_available_servers(self) -> List[Dict]:
        """Get list of available VPN servers"""
        # This would load from your VPN provider's API or config files
        servers = [
            {'country': 'Switzerland', 'city': 'Zurich', 'ip': '1.2.3.4', 'port': 1194},
            {'country': 'Iceland', 'city': 'Reykjavik', 'ip': '2.3.4.5', 'port': 1194},
            {'country': 'Romania', 'city': 'Bucharest', 'ip': '3.4.5.6', 'port': 1194},
```

```
            {'country': 'Sweden', 'city': 'Stockholm', 'ip': '4.5.6.7', 'port': 1194},
            {'country': 'Netherlands', 'city': 'Amsterdam', 'ip': '5.6.7.8', 'port': 1194},
        ]
        return servers

    def select_diverse_servers(self, servers: List[Dict], count: int) -> List[Dict]:
        """Select servers from different countries"""
        countries = {}
        for server in servers:
            country = server['country']
            if country not in countries:
                countries[country] = []
            countries[country].append(server)

        selected = []
        available_countries = list(countries.keys())
        random.shuffle(available_countries)

        for country in available_countries[:count]:
            server = random.choice(countries[country])
            selected.append(server)

        return selected

    def create_namespace(self, namespace: str):
        """Create network namespace for VPN isolation"""
        subprocess.run(['ip', 'netns', 'add', namespace], check=True)
        subprocess.run(['ip', 'netns', 'exec', namespace, 'ip', 'link', 'set', 'lo', 'up'], check=True)

    def connect_vpn(self, server: Dict, namespace: str = None) -> bool:
        """Connect to VPN server"""
        config_file = self.generate_vpn_config(server)

        cmd = ['openvpn', '--config', config_file, '--daemon']

        if namespace:
            cmd = ['ip', 'netns', 'exec', namespace] + cmd

        try:
            subprocess.run(cmd, check=True)
            time.sleep(5)  # Wait for connection
            return self.verify_connection(server, namespace)
        except subprocess.CalledProcessError:
            return False

    def generate_vpn_config(self, server: Dict) -> str:
        """Generate OpenVPN configuration"""
        config = f"""
client
dev tun
proto udp
remote {server['ip']} {server['port']}
resolv-retry infinite
nobind
persist-key
persist-tun
remote-cert-tls server
auth SHA512
cipher AES-256-CBC
comp-lzo
verb 3

# Authentication
auth-user-pass /etc/vpn_configs/auth.txt

# Security
tls-client
tls-version-min 1.2
tls-cipher TLS-DHE-RSA-WITH-AES-256-GCM-SHA384

# Privacy
script-security 2
up /etc/vpn_configs/up.sh
```

```
down /etc/vpn_configs/down.sh
        """

        config_path = f"/tmp/vpn_{server['country']}.conf"
        with open(config_path, 'w') as f:
            f.write(config)

        return config_path

    def route_through_namespace(self, source_ns: str, target_ns: str):
        """Route traffic from source namespace through target"""
        # Create veth pair
        veth_source = f"veth_{source_ns}"
        veth_target = f"veth_{target_ns}"

        subprocess.run([
            'ip', 'link', 'add', veth_source, 'type', 'veth',
            'peer', 'name', veth_target
        ], check=True)

        # Move interfaces to namespaces
        subprocess.run(['ip', 'link', 'set', veth_source, 'netns', source_ns], check=True)
        subprocess.run(['ip', 'link', 'set', veth_target, 'netns', target_ns], check=True)

        # Configure interfaces
        subprocess.run([
            'ip', 'netns', 'exec', source_ns,
            'ip', 'addr', 'add', '10.0.0.1/24', 'dev', veth_source
        ], check=True)

        subprocess.run([
            'ip', 'netns', 'exec', target_ns,
            'ip', 'addr', 'add', '10.0.0.2/24', 'dev', veth_target
        ], check=True)

        # Enable forwarding and NAT
        subprocess.run([
            'ip', 'netns', 'exec', target_ns,
            'iptables', '-t', 'nat', '-A', 'POSTROUTING', '-j', 'MASQUERADE'
        ], check=True)

    def verify_connection(self, server: Dict, namespace: str = None) -> bool:
        """Verify VPN connection is active"""
        cmd = ['curl', '-s', 'https://api.ipify.org']

        if namespace:
            cmd = ['ip', 'netns', 'exec', namespace] + cmd

        try:
            result = subprocess.run(cmd, capture_output=True, text=True, timeout=10)
            current_ip = result.stdout.strip()

            # Check if IP changed
            if current_ip and current_ip != self.get_real_ip():
                self.logger.info(f"Connected to {server['country']}, new IP: {current_ip}")
                return True
        except:
            pass

        return False

    def get_real_ip(self) -> str:
        """Get real IP address"""
        try:
            result = subprocess.run(['curl', '-s', 'https://api.ipify.org'],
                                    capture_output=True, text=True, timeout=5)
            return result.stdout.strip()
        except:
            return None

    def cleanup_connections(self):
        """Clean up all VPN connections"""
        for conn in reversed(self.active_connections):
```

```
            if conn['namespace']:
                subprocess.run(['ip', 'netns', 'delete', conn['namespace']], check=False)
            subprocess.run(['killall', 'openvpn'], check=False)

        self.active_connections = []

# Usage
if __name__ == "__main__":
    vpn_manager = MultiHopVPN()

    # Create 3-hop VPN chain
    if vpn_manager.create_vpn_chain(hops=3):
        print("Multi-hop VPN chain established")

        # Test the connection
        print("Testing connection through chain...")
        result = subprocess.run(['curl', '-s', 'https://api.ipify.org'],
                                capture_output=True, text=True)
        print(f"Final exit IP: {result.stdout}")
    else:
        print("Failed to create VPN chain")
```

Network Segmentation

VLAN and Network Isolation

bash

```bash
#!/bin/bash
# network_segmentation.sh - Create isolated network segments

# Create VLANs for different security zones
create_vlans() {
    # Load 8021q module
    sudo modprobe 8021q
    echo "8021q" | sudo tee -a /etc/modules

    # Define VLANs
    declare -A vlans=(
        [10]="management"
        [20]="trusted"
        [30]="guest"
        [40]="iot"
        [50]="dmz"
        [99]="quarantine"
    )

    # Create VLAN interfaces
    for vlan_id in "${!vlans[@]}"; do
        vlan_name=${vlans[$vlan_id]}

        # Create VLAN interface
        sudo ip link add link eth0 name eth0.$vlan_id type vlan id $vlan_id

        # Assign IP address
        case $vlan_id in
            10) ip_addr="192.168.10.1/24" ;;
            20) ip_addr="192.168.20.1/24" ;;
            30) ip_addr="192.168.30.1/24" ;;
            40) ip_addr="192.168.40.1/24" ;;
            50) ip_addr="192.168.50.1/24" ;;
            99) ip_addr="192.168.99.1/24" ;;
        esac

        sudo ip addr add $ip_addr dev eth0.$vlan_id
        sudo ip link set eth0.$vlan_id up

        echo "Created VLAN $vlan_id ($vlan_name) with IP $ip_addr"
    done
}

# Configure firewall rules for VLANs
configure_vlan_firewall() {
    # Flush existing rules
    sudo iptables -F
    sudo iptables -X
    sudo iptables -t nat -F
    sudo iptables -t nat -X

    # Default policies
    sudo iptables -P INPUT DROP
    sudo iptables -P FORWARD DROP
    sudo iptables -P OUTPUT ACCEPT

    # Allow established connections
    sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
    sudo iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

    # Management VLAN (10) - Full access
    sudo iptables -A INPUT -i eth0.10 -j ACCEPT
    sudo iptables -A FORWARD -i eth0.10 -j ACCEPT

    # Trusted VLAN (20) - Limited access
    sudo iptables -A FORWARD -i eth0.20 -o eth0 -j ACCEPT
    sudo iptables -A FORWARD -i eth0.20 -o eth0.10 -j DROP
    sudo iptables -A FORWARD -i eth0.20 -o eth0.30 -j DROP
    sudo iptables -A FORWARD -i eth0.20 -o eth0.40 -j DROP

    # Guest VLAN (30) - Internet only
    sudo iptables -A FORWARD -i eth0.30 -o eth0 -j ACCEPT
    sudo iptables -A FORWARD -i eth0.30 -d 192.168.0.0/16 -j DROP
```

```
    sudo iptables -A FORWARD -i eth0.30 -d 10.0.0.0/8 -j DROP

    # IoT VLAN (40) - Isolated, limited internet
    sudo iptables -A FORWARD -i eth0.40 -o eth0 -p tcp --dport 443 -j ACCEPT
    sudo iptables -A FORWARD -i eth0.40 -o eth0 -p tcp --dport 80 -j ACCEPT
    sudo iptables -A FORWARD -i eth0.40 -o eth0 -p udp --dport 123 -j ACCEPT
    sudo iptables -A FORWARD -i eth0.40 -j DROP

    # DMZ VLAN (50) - Public services
    sudo iptables -A FORWARD -i eth0 -o eth0.50 -p tcp --dport 80 -j ACCEPT
    sudo iptables -A FORWARD -i eth0 -o eth0.50 -p tcp --dport 443 -j ACCEPT
    sudo iptables -A FORWARD -i eth0.50 -o eth0.20 -j DROP

    # Quarantine VLAN (99) - No access
    sudo iptables -A FORWARD -i eth0.99 -j DROP
    sudo iptables -A FORWARD -o eth0.99 -j DROP

    # Enable NAT for internet access
    sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

    # Enable routing
    echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
}

# Setup network namespaces for container isolation
create_network_namespaces() {
    # Create namespaces for different security contexts
    namespaces=("secure" "sandbox" "untrusted")

    for ns in "${namespaces[@]}"; do
        # Create namespace
        sudo ip netns add $ns

        # Create veth pair
        sudo ip link add veth-$ns type veth peer name veth-$ns-peer

        # Move one end to namespace
        sudo ip link set veth-$ns-peer netns $ns

        # Configure IPs
        case $ns in
            secure)
                host_ip="10.200.1.1/24"
                ns_ip="10.200.1.2/24"
                ;;
            sandbox)
                host_ip="10.200.2.1/24"
                ns_ip="10.200.2.2/24"
                ;;
            untrusted)
                host_ip="10.200.3.1/24"
                ns_ip="10.200.3.2/24"
                ;;
        esac

        # Configure host side
        sudo ip addr add $host_ip dev veth-$ns
        sudo ip link set veth-$ns up

        # Configure namespace side
        sudo ip netns exec $ns ip addr add $ns_ip dev veth-$ns-peer
        sudo ip netns exec $ns ip link set veth-$ns-peer up
        sudo ip netns exec $ns ip link set lo up

        # Add default route in namespace
        sudo ip netns exec $ns ip route add default via ${host_ip%/*}

        # Configure firewall for namespace
        configure_namespace_firewall $ns

        echo "Created network namespace: $ns"
    done
}
```

```
configure_namespace_firewall() {
    local ns=$1

    case $ns in
        secure)
            # Allow only specific services
            sudo iptables -A FORWARD -i veth-$ns -p tcp --dport 443 -j ACCEPT
            sudo iptables -A FORWARD -i veth-$ns -p udp --dport 53 -j ACCEPT
            sudo iptables -A FORWARD -i veth-$ns -j DROP
            ;;
        sandbox)
            # Limited access
            sudo iptables -A FORWARD -i veth-$ns -p tcp --dport 80 -j ACCEPT
            sudo iptables -A FORWARD -i veth-$ns -p tcp --dport 443 -j ACCEPT
            sudo iptables -A FORWARD -i veth-$ns -p udp --dport 53 -j ACCEPT
            sudo iptables -A FORWARD -i veth-$ns -d 192.168.0.0/16 -j DROP
            ;;
        untrusted)
            # No network access
            sudo iptables -A FORWARD -i veth-$ns -j DROP
            ;;
    esac
}

# Main execution
create_vlans
configure_vlan_firewall
create_network_namespaces
```

Traffic Obfuscation

Protocol Obfuscation and Steganography

```
python
```

```python
#!/usr/bin/env python3
# traffic_obfuscation.py - Network traffic obfuscation techniques

import socket
import ssl
import struct
import random
import hashlib
import base64
from cryptography.fernet import Fernet
from typing import bytes, Tuple
import asyncio

class TrafficObfuscator:
    def __init__(self):
        self.key = Fernet.generate_key()
        self.cipher = Fernet(self.key)
        self.obfuscation_methods = {
            'http': self.obfuscate_as_http,
            'https': self.obfuscate_as_https,
            'dns': self.obfuscate_as_dns,
            'ntp': self.obfuscate_as_ntp,
            'random': self.random_obfuscation
        }

    def obfuscate_traffic(self, data: bytes, method: str = 'https') -> bytes:
        """Obfuscate traffic to look like legitimate protocol"""
        if method in self.obfuscation_methods:
            return self.obfuscation_methods[method](data)
        return data

    def obfuscate_as_http(self, data: bytes) -> bytes:
        """Make traffic look like HTTP"""
        # Encrypt the actual data
        encrypted = self.cipher.encrypt(data)
        encoded = base64.b64encode(encrypted).decode()

        # Create fake HTTP request
        http_request = f"""GET /api/v1/data HTTP/1.1\r
Host: api.legitimate-site.com\r
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36\r
Accept: application/json\r
Accept-Language: en-US,en;q=0.9\r
Accept-Encoding: gzip, deflate, br\r
Connection: keep-alive\r
Cookie: session={encoded[:32]}\r
X-Request-ID: {encoded[32:64]}\r
X-Data: {encoded[64:]}\r
\r
"""
        return http_request.encode()

    def obfuscate_as_https(self, data: bytes) -> bytes:
        """Make traffic look like HTTPS"""
        # Add TLS-like header
        tls_header = struct.pack('!BBH',
                                 0x16,  # Content Type: Application Data
                                 0x03, 0x03,  # TLS Version 1.2
                                 len(data))  # Length

        # Encrypt and pad data
        encrypted = self.cipher.encrypt(data)

        # Add random padding to mimic TLS padding
        padding_length = random.randint(1, 255)
        padding = bytes([padding_length] * padding_length)

        return tls_header + encrypted + padding

    def obfuscate_as_dns(self, data: bytes) -> bytes:
        """Tunnel data through DNS queries"""
        # Encrypt and encode data
        encrypted = self.cipher.encrypt(data)
```

```
        encoded = base64.b32encode(encrypted).decode().lower().replace('=', '')

        # Split into DNS labels (max 63 chars each)
        labels = [encoded[i:i+63] for i in range(0, len(encoded), 63)]

        # Create DNS query
        dns_query = b''

        # DNS Header
        transaction_id = random.randint(0, 65535)
        flags = 0x0100  # Standard query
        dns_query += struct.pack('!HHHHHH',
                                 transaction_id, flags,
                                 1, 0, 0, 0)  # 1 question, 0 answers

        # DNS Question
        for label in labels[:3]:  # Limit to 3 labels for subdomain
            dns_query += bytes([len(label)]) + label.encode()

        # Add base domain
        dns_query += b'\x07example\x03com\x00'

        # Query type and class
        dns_query += struct.pack('!HH', 1, 1)  # Type A, Class IN

        return dns_query

    def obfuscate_as_ntp(self, data: bytes) -> bytes:
        """Make traffic look like NTP"""
        # NTP packet structure
        ntp_packet = bytearray(48)

        # LI, Version, Mode
        ntp_packet[0] = 0x1b  # LI=0, Version=3, Mode=3 (client)

        # Hide data in timestamps
        encrypted = self.cipher.encrypt(data)

        # Embed encrypted data in timestamp fields
        for i, byte in enumerate(encrypted[:32]):
            ntp_packet[16 + i] = byte

        # Set reference timestamp
        ntp_packet[16:24] = struct.pack('!Q', int(time.time() * 2**32))

        return bytes(ntp_packet)

    def random_obfuscation(self, data: bytes) -> bytes:
        """Apply random obfuscation patterns"""
        methods = ['chaff', 'timing', 'size_padding', 'fragmentation']
        method = random.choice(methods)

        if method == 'chaff':
            # Add random chaff packets
            chaff_size = random.randint(100, 1000)
            chaff = os.urandom(chaff_size)
            marker = b'\x00\xff\x00\xff'  # Marker to identify real data
            return marker + data + marker + chaff

        elif method == 'timing':
            # Add timing delays between bytes
            obfuscated = b''
            for byte in data:
                obfuscated += bytes([byte])
                delay = random.randint(0, 255)
                obfuscated += bytes([delay])
            return obfuscated

        elif method == 'size_padding':
            # Pad to standard sizes to prevent traffic analysis
            standard_sizes = [512, 1024, 1500, 2048, 4096]
            target_size = min(s for s in standard_sizes if s >= len(data))
            padding_size = target_size - len(data) - 2
```

```
            return struct.pack('!H', len(data)) + data + os.urandom(padding_size)

        elif method == 'fragmentation':
            # Fragment data into random sized chunks
            fragments = []
            pos = 0
            while pos < len(data):
                chunk_size = random.randint(10, 100)
                fragment = data[pos:pos + chunk_size]
                fragments.append(struct.pack('!H', len(fragment)) + fragment)
                pos += chunk_size

            # Shuffle fragments
            random.shuffle(fragments)
            return b''.join(fragments)

        return data

class StealthProxy:
    """Proxy server with traffic obfuscation"""

    def __init__(self, listen_port: int = 8888):
        self.listen_port = listen_port
        self.obfuscator = TrafficObfuscator()

    async def handle_client(self, reader, writer):
        """Handle client connection with obfuscation"""
        try:
            # Read client request
            data = await reader.read(4096)

            if not data:
                return

            # Parse destination
            dest_host, dest_port = self.parse_socks_request(data)

            # Connect to destination through obfuscated channel
            dest_reader, dest_writer = await self.connect_obfuscated(dest_host, dest_port)

            # Relay traffic with obfuscation
            await asyncio.gather(
                self.relay_traffic(reader, dest_writer, obfuscate=True),
                self.relay_traffic(dest_reader, writer, obfuscate=False)
            )

        except Exception as e:
            print(f"Error: {e}")
        finally:
            writer.close()
            await writer.wait_closed()

    def parse_socks_request(self, data: bytes) -> Tuple[str, int]:
        """Parse SOCKS5 request"""
        # Simplified SOCKS5 parsing
        if data[0] == 0x05:  # SOCKS5
            # Skip authentication
            addr_type = data[3]

            if addr_type == 0x01:  # IPv4
                addr = socket.inet_ntoa(data[4:8])
                port = struct.unpack('!H', data[8:10])[0]
                return addr, port
            elif addr_type == 0x03:  # Domain
                addr_len = data[4]
                addr = data[5:5 + addr_len].decode()
                port = struct.unpack('!H', data[5 + addr_len:7 + addr_len])[0]
                return addr, port

        raise ValueError("Invalid SOCKS request")

    async def connect_obfuscated(self, host: str, port: int):
        """Connect to destination with obfuscation"""
```

```
        # Use obfuscated connection
        reader, writer = await asyncio.open_connection(host, port)

        # Wrap in obfuscation layer
        return reader, writer

    async def relay_traffic(self, reader, writer, obfuscate: bool):
        """Relay traffic between client and server"""
        try:
            while True:
                data = await reader.read(4096)
                if not data:
                    break

                if obfuscate:
                    # Apply random obfuscation method
                    methods = ['http', 'https', 'dns', 'ntp', 'random']
                    method = random.choice(methods)
                    data = self.obfuscator.obfuscate_traffic(data, method)

                writer.write(data)
                await writer.drain()
        except:
            pass
        finally:
            writer.close()

    async def start(self):
        """Start the stealth proxy server"""
        server = await asyncio.start_server(
            self.handle_client,
            '127.0.0.1',
            self.listen_port
        )

        print(f"Stealth proxy listening on port {self.listen_port}")

        async with server:
            await server.serve_forever()

# Usage
if __name__ == "__main__":
    import asyncio

    proxy = StealthProxy(listen_port=8888)
    asyncio.run(proxy.start())
```

Mesh Networking

Decentralized Mesh Network Setup

```
bash
```

```bash
#!/bin/bash
# mesh_network.sh - Create decentralized mesh network

# Install mesh networking tools
install_mesh_tools() {
    sudo apt-get update
    sudo apt-get install -y \
        batman-adv \
        babeld \
        olsrd \
        bmx7 \
        yggdrasil \
        cjdns \
        wireguard-tools
}

# Setup Batman-adv mesh network
setup_batman_mesh() {
    # Load batman-adv module
    sudo modprobe batman-adv

    # Create mesh interface
    sudo ip link add name mesh0 type batadv

    # Add wireless interface to batman
    sudo iw dev wlan0 interface add mesh0 type mesh
    sudo ip link set mesh0 master bat0
    sudo ip link set mesh0 up
    sudo ip link set bat0 up

    # Assign IP address
    sudo ip addr add 10.0.0.1/24 dev bat0

    # Configure wireless interface for mesh
    sudo iw dev mesh0 set type mesh
    sudo iw dev mesh0 mesh join my-mesh-network
    sudo ip link set mesh0 up
}

# Setup Yggdrasil overlay network
setup_yggdrasil() {
    # Generate configuration
    sudo yggdrasil -genconf > /etc/yggdrasil.conf

    # Modify configuration for privacy
    sudo tee /etc/yggdrasil.conf << 'EOF'
{
  # Your public key (auto-generated)
  PublicKey: "",

  # Your private key (auto-generated)
  PrivateKey: "",

  # Listen for connections
  Listen: [
    "tcp://0.0.0.0:9001",
    "tls://0.0.0.0:9002"
  ],

  # Peer connections
  Peers: [
    "tcp://ygg-peer1.example.com:9001",
    "tls://ygg-peer2.example.com:9002"
  ],

  # Network interface configuration
  IfName: "ygg0",
  IfTAPMode: false,
  IfMTU: 1280,

  # Session firewall
  SessionFirewall: {
    Enable: true,
```

```
    AllowFromDirect: true,
    AllowFromRemote: true,
    AlwaysAllowOutbound: true,
    WhitelistEncryptionPublicKeys: [],
    BlacklistEncryptionPublicKeys: []
  },

  # Tunnel routing
  TunnelRouting: {
    Enable: true,
    IPv6RemoteSubnets: {},
    IPv6LocalSubnets: [],
    IPv4RemoteSubnets: {},
    IPv4LocalSubnets: []
  },

  # Switch options
  SwitchOptions: {
    MaxTotalQueueSize: 4194304
  }
}
EOF

    # Start Yggdrasil
    sudo systemctl enable yggdrasil
    sudo systemctl start yggdrasil
}

# Setup WireGuard mesh
setup_wireguard_mesh() {
    # Generate keys
    wg genkey | tee privatekey | wg pubkey > publickey

    # Create WireGuard interface
    sudo ip link add wg-mesh type wireguard

    # Configure interface
    sudo tee /etc/wireguard/wg-mesh.conf << EOF
[Interface]
PrivateKey = $(cat privatekey)
Address = 10.100.0.1/24
ListenPort = 51820
PostUp = iptables -A FORWARD -i wg-mesh -j ACCEPT; iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
PostDown = iptables -D FORWARD -i wg-mesh -j ACCEPT; iptables -t nat -D POSTROUTING -o eth0 -j MASQUERADE

# Peer 1
[Peer]
PublicKey = [PEER1_PUBLIC_KEY]
AllowedIPs = 10.100.0.2/32
Endpoint = peer1.example.com:51820
PersistentKeepalive = 25

# Peer 2
[Peer]
PublicKey = [PEER2_PUBLIC_KEY]
AllowedIPs = 10.100.0.3/32
Endpoint = peer2.example.com:51820
PersistentKeepalive = 25
EOF

    # Start WireGuard
    sudo wg-quick up wg-mesh
    sudo systemctl enable wg-quick@wg-mesh
}

# Setup CJDNS mesh network
setup_cjdns() {
    # Generate configuration
    sudo cjdroute --genconf > /etc/cjdroute.conf

    # Start CJDNS
    sudo systemctl enable cjdns
    sudo systemctl start cjdns
```

```
    # Get CJDNS IPv6 address
    sudo cjdns-online -w
    CJDNS_IP=$(sudo nodejs -e "console.log(require('/etc/cjdroute.conf').ipv6)")
    echo "CJDNS IPv6: $CJDNS_IP"
}

# Configure mesh routing protocols
configure_mesh_routing() {
    # Babel configuration
    sudo tee /etc/babeld.conf << EOF
# Babel mesh routing configuration
interface wlan0
interface eth0

# Redistribute local routes
redistribute local deny
redistribute ip 10.0.0.0/24 metric 128

# Security
key id 1 type sha256 value "your-secret-key-here"
default key 1

# Diversity routing
diversity true
diversity-factor 3
EOF

    # Start Babel
    sudo systemctl enable babeld
    sudo systemctl start babeld
}

# Setup distributed DNS for mesh
setup_mesh_dns() {
    # Use Namecoin for decentralized DNS
    sudo apt-get install -y namecoin

    # Configure Namecoin
    mkdir -p ~/.namecoin
    cat > ~/.namecoin/namecoin.conf << EOF
rpcuser=mesh_user
rpcpassword=$(openssl rand -hex 32)
rpcport=8336
server=1
listen=1
EOF

    # Start Namecoin
    namecoind -daemon

    # Configure DNS resolver
    sudo tee /etc/unbound/unbound.conf.d/mesh-dns.conf << EOF
server:
    # Namecoin integration
    local-zone: "bit." redirect
    local-data: "bit. IN A 127.0.0.1"

    # Mesh TLD
    local-zone: "mesh." static
    local-data: "node1.mesh. IN A 10.0.0.1"
    local-data: "node2.mesh. IN A 10.0.0.2"
EOF

    sudo systemctl restart unbound
}

# Main execution
install_mesh_tools
setup_batman_mesh
setup_yggdrasil
setup_wireguard_mesh
setup_cjdns
```

```
configure_mesh_routing
setup_mesh_dns

echo "Mesh network configuration complete"
```