Personal Security & Privacy Protection Guide

Table of Contents

MAC Address Rotation

MAC address randomization is a privacy feature built into modern operating systems to prevent tracking across different networks.

Windows 11/10 Configuration

```powershell
# Enable MAC randomization for all networks
# Run as Administrator

# Check current settings
Get-NetAdapter | Select Name, MacAddress

# Enable random MAC for all WiFi networks
Set-NetIPInterface -InterfaceAlias "Wi-Fi" -RandomizeIdentifier Enabled

# Registry method for persistent configuration
reg add "HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters" /v "EnableRandomization" /t REG_DWORD /d 1

# Per-network randomization
netsh wlan set profileparameter name="NetworkName" randomization=enable
```

Linux MAC Rotation

```bash
```

```
#!/bin/bash
# MAC Address Rotation Script for Linux

# Install macchanger
sudo apt-get install macchanger network-manager

# NetworkManager configuration
sudo tee /etc/NetworkManager/conf.d/99-random-mac.conf << EOF
[device-mac-randomization]
wifi.scan-rand-mac-address=yes

[connection-mac-randomization]
ethernet.cloned-mac-address=random
wifi.cloned-mac-address=random
EOF

# Systemd service for MAC rotation
sudo tee /etc/systemd/system/mac-randomize.service << EOF
[Unit]
Description=Randomize MAC Address
Before=network-pre.target

[Service]
Type=oneshot
ExecStart=/usr/bin/macchanger -r wlan0
ExecStart=/usr/bin/macchanger -r eth0

[Install]
WantedBy=multi-user.target
EOF

sudo systemctl enable mac-randomize.service

# Manual rotation function
rotate_mac() {
    interface=$1
    sudo ip link set dev $interface down
    sudo macchanger -r $interface
    sudo ip link set dev $interface up
    echo "MAC rotated for $interface"
}
```

macOS MAC Randomization

```bash
# macOS Monterey and later have built-in MAC randomization

# Enable WiFi MAC randomization
sudo defaults write /Library/Preferences/SystemConfiguration/com.apple.wifi.plist WiFiMACAddressRandomization -bo

# Verify setting
defaults read /Library/Preferences/SystemConfiguration/com.apple.wifi.plist WiFiMACAddressRandomization

# Manual MAC change (temporary)
sudo ifconfig en0 ether $(openssl rand -hex 6 | sed 's/\(..\)/\1:/g; s/.$//')
```

Firewall Configuration

Linux iptables/nftables Rules

```bash
```

```bash
#!/bin/bash
# Comprehensive Firewall Rules for Personal Protection

# Flush existing rules
iptables -F
iptables -X
iptables -Z

# Default policies - Deny all incoming, allow outgoing
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT

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

# Allow established connections
iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

# Protection against common attacks
# SYN flood protection
iptables -A INPUT -p tcp --syn -m limit --limit 1/s --limit-burst 3 -j ACCEPT

# Port scanning protection
iptables -N PORT_SCANNING
iptables -A PORT_SCANNING -p tcp --tcp-flags SYN,ACK,FIN,RST RST -m limit --limit 1/s --limit-burst 2 -j RETURN
iptables -A PORT_SCANNING -j DROP

# ICMP flood protection
iptables -A INPUT -p icmp -m limit --limit 1/s --limit-burst 1 -j ACCEPT
iptables -A INPUT -p icmp -j DROP

# Invalid packets
iptables -A INPUT -m conntrack --ctstate INVALID -j DROP

# Block common attack ports
iptables -A INPUT -p tcp --dport 135:139 -j DROP  # Windows NetBIOS
iptables -A INPUT -p tcp --dport 445 -j DROP      # SMB
iptables -A INPUT -p tcp --dport 23 -j DROP       # Telnet
iptables -A INPUT -p tcp --dport 21 -j DROP       # FTP

# Allow specific services (customize as needed)
# SSH with rate limiting
iptables -A INPUT -p tcp --dport 22 -m recent --name SSH --set --rsource
iptables -A INPUT -p tcp --dport 22 -m recent --name SSH --update --seconds 60 --hitcount 4 --rsource -j DROP
iptables -A INPUT -p tcp --dport 22 -j ACCEPT

# Save rules
iptables-save > /etc/iptables/rules.v4

# IPv6 rules
ip6tables -P INPUT DROP
ip6tables -P FORWARD DROP
ip6tables -P OUTPUT ACCEPT
ip6tables -A INPUT -i lo -j ACCEPT
ip6tables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
ip6tables -A INPUT -p ipv6-icmp -j ACCEPT
ip6tables-save > /etc/iptables/rules.v6
```

Windows Firewall PowerShell Configuration

```
powershell
```

```
# Windows Defender Firewall Hardening Script

# Enable firewall for all profiles
Set-NetFirewallProfile -All -Enabled True

# Set default actions
Set-NetFirewallProfile -All -DefaultInboundAction Block -DefaultOutboundAction Allow

# Remove unnecessary rules
Get-NetFirewallRule | Where-Object {$_.DisplayName -like "*Remote*"} | Disable-NetFirewallRule

# Block common attack vectors
New-NetFirewallRule -DisplayName "Block NetBIOS" -Direction Inbound -Protocol TCP -LocalPort 135-139 -Action Bloc
New-NetFirewallRule -DisplayName "Block SMB" -Direction Inbound -Protocol TCP -LocalPort 445 -Action Block
New-NetFirewallRule -DisplayName "Block RDP Brute Force" -Direction Inbound -Protocol TCP -LocalPort 3389 -Action

# Create logging
Set-NetFirewallProfile -All -LogAllowed True -LogBlocked True -LogFileName "%SystemRoot%\System32\LogFiles\Firev

# Advanced Security Settings
netsh advfirewall set allprofiles state on
netsh advfirewall set allprofiles firewallpolicy blockinbound,allowoutbound
netsh advfirewall set allprofiles logging filename "%systemroot%\system32\LogFiles\Firewall\pfirewall.log" maxfilesize 4
```

Network Security

WiFi Security Configuration

bash

```
#!/bin/bash
# WiFi Security Hardening

# Disable WiFi when not in use
disable_wifi_sleep() {
    sudo tee /etc/systemd/system/wifi-sleep.service << EOF
[Unit]
Description=Disable WiFi on Sleep
Before=sleep.target

[Service]
Type=oneshot
ExecStart=/usr/bin/nmcli radio wifi off
ExecStartPost=/usr/bin/sleep 1

[Install]
WantedBy=sleep.target
EOF
    sudo systemctl enable wifi-sleep.service
}

# WPA3 Configuration
configure_wpa3() {
    sudo tee /etc/wpa_supplicant/wpa_supplicant.conf << EOF
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=US

network={
    ssid="YourNetwork"
    psk="YourPassword"
    key_mgmt=SAE
    ieee80211w=2
    proto=RSN
    pairwise=CCMP
    group=CCMP
}
EOF
}

# Monitor for rogue access points
monitor_rogue_ap() {
    known_aps="AA:BB:CC:DD:EE:FF"
    while true; do
        current_ap=$(iwconfig 2>/dev/null | grep "Access Point" | awk '{print $6}')
        if [[ ! "$known_aps" =~ "$current_ap" ]] && [[ "$current_ap" != "Not-Associated" ]]; then
            echo "WARNING: Connected to unknown AP: $current_ap"
            notify-send "Security Alert" "Connected to unknown WiFi AP!"
        fi
        sleep 30
    done
}
```

DNS Security

```
bash
```

```
# DNS over HTTPS (DoH) Configuration

# systemd-resolved configuration
sudo tee /etc/systemd/resolved.conf << EOF
[Resolve]
DNS=1.1.1.1#cloudflare-dns.com 9.9.9.9#dns.quad9.net
DNSOverTLS=yes
DNSSEC=yes
DNSStubListener=yes
FallbackDNS=8.8.8.8 8.8.4.4
EOF

# DNSCrypt proxy setup
install_dnscrypt() {
    wget https://github.com/DNSCrypt/dnscrypt-proxy/releases/latest/download/dnscrypt-proxy-linux_x86_64.tar.gz
    tar -xf dnscrypt-proxy-linux_x86_64.tar.gz
    sudo mv linux-x86_64/dnscrypt-proxy /usr/local/bin/

    sudo tee /etc/dnscrypt-proxy/dnscrypt-proxy.toml << EOF
server_names = ['cloudflare', 'quad9-dnscrypt-ip4-filter-pri']
listen_addresses = ['127.0.0.1:53']
max_clients = 250
ipv4_servers = true
ipv6_servers = false
dnscrypt_servers = true
doh_servers = true
require_dnssec = true
require_nolog = true
require_nofilter = false
force_tcp = false
timeout = 2500
keepalive = 30
cert_refresh_delay = 240
block_ipv6 = false
cache = true
cache_size = 4096
cache_min_ttl = 2400
cache_max_ttl = 86400
EOF
}
```

System Hardening

Linux Security Modules

```bash
```

```bash
#!/bin/bash
# System Hardening Script

# Kernel parameters for security
sudo tee /etc/sysctl.d/99-security.conf << EOF
# IP Spoofing protection
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.default.rp_filter = 1

# Ignore ICMP redirects
net.ipv4.conf.all.accept_redirects = 0
net.ipv6.conf.all.accept_redirects = 0

# Ignore send redirects
net.ipv4.conf.all.send_redirects = 0

# Disable source packet routing
net.ipv4.conf.all.accept_source_route = 0
net.ipv6.conf.all.accept_source_route = 0

# Log Martians
net.ipv4.conf.all.log_martians = 1

# Ignore ICMP ping requests
net.ipv4.icmp_echo_ignore_broadcasts = 1
net.ipv4.icmp_ignore_bogus_error_responses = 1

# SYN flood protection
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_syn_retries = 2
net.ipv4.tcp_synack_retries = 2
net.ipv4.tcp_max_syn_backlog = 4096

# Disable IPv6 if not needed
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1

# Kernel hardening
kernel.randomize_va_space = 2
kernel.yama.ptrace_scope = 1
kernel.kptr_restrict = 2
kernel.dmesg_restrict = 1
kernel.kexec_load_disabled = 1
kernel.modules_disabled = 1
EOF

sudo sysctl -p /etc/sysctl.d/99-security.conf

# AppArmor/SELinux enforcement
sudo aa-enforce /etc/apparmor.d/*
sudo setenforce 1

# Disable unnecessary services
services_to_disable=(
    "bluetooth"
    "cups"
    "avahi-daemon"
    "rpcbind"
    "nfs-client"
)

for service in "${services_to_disable[@]}"; do
    sudo systemctl disable --now "$service" 2>/dev/null
done
```

File System Security

```
bash
```

```
# Secure mount options
sudo tee /etc/fstab.security << EOF
# Secure /tmp
tmpfs /tmp tmpfs defaults,noexec,nosuid,nodev,mode=1777 0 0
tmpfs /var/tmp tmpfs defaults,noexec,nosuid,nodev,mode=1777 0 0

# Secure /home
/dev/mapper/home /home ext4 defaults,nodev,nosuid 0 2

# Secure shared memory
tmpfs /run/shm tmpfs defaults,noexec,nosuid,nodev 0 0
EOF

# File integrity monitoring
install_aide() {
    sudo apt-get install aide
    sudo aideinit
    sudo mv /var/lib/aide/aide.db.new /var/lib/aide/aide.db

    # Create daily check
    sudo tee /etc/cron.daily/aide-check << 'EOF'
#!/bin/bash
/usr/bin/aide --check | mail -s "AIDE Daily Report" admin@localhost
EOF
    sudo chmod +x /etc/cron.daily/aide-check
}
```

Privacy Protection

Browser Privacy Configuration

```javascript
// Firefox user.js for maximum privacy
// Place in Firefox profile folder

// Disable telemetry
user_pref("datareporting.healthreport.uploadEnabled", false);
user_pref("toolkit.telemetry.unified", false);
user_pref("toolkit.telemetry.enabled", false);
user_pref("toolkit.telemetry.server", "data:,");
user_pref("toolkit.telemetry.archive.enabled", false);

// Disable tracking
user_pref("privacy.trackingprotection.enabled", true);
user_pref("privacy.trackingprotection.socialtracking.enabled", true);
user_pref("privacy.trackingprotection.cryptomining.enabled", true);
user_pref("privacy.trackingprotection.fingerprinting.enabled", true);

// DNS over HTTPS
user_pref("network.trr.mode", 3);
user_pref("network.trr.uri", "https://cloudflare-dns.com/dns-query");

// Disable WebRTC leak
user_pref("media.peerconnection.enabled", false);
user_pref("media.peerconnection.ice.default_address_only", true);

// Disable geolocation
user_pref("geo.enabled", false);

// Disable WebGL
user_pref("webgl.disabled", true);

// Resist fingerprinting
user_pref("privacy.resistFingerprinting", true);
user_pref("privacy.resistFingerprinting.letterboxing", true);

// First party isolation
user_pref("privacy.firstparty.isolate", true);
```

VPN Kill Switch

```bash
#!/bin/bash
# VPN Kill Switch to prevent leaks

# UFW-based kill switch
setup_vpn_killswitch() {
    # Reset UFW
    sudo ufw --force reset

    # Default policies
    sudo ufw default deny incoming
    sudo ufw default deny outgoing

    # Allow VPN connection
    sudo ufw allow out on tun0
    sudo ufw allow out to 10.8.0.0/24  # VPN subnet

    # Allow connection to VPN server
    VPN_SERVER="your.vpn.server.com"
    VPN_PORT="1194"
    sudo ufw allow out to $VPN_SERVER port $VPN_PORT

    # Allow local network
    sudo ufw allow out to 192.168.0.0/16
    sudo ufw allow out to 10.0.0.0/8

    # Allow DNS for VPN
    sudo ufw allow out 53/udp

    # Enable UFW
    sudo ufw enable
}

# Monitor VPN connection
monitor_vpn() {
    while true; do
        if ! ip a | grep -q "tun0"; then
            echo "VPN disconnected! Blocking all traffic..."
            sudo iptables -P OUTPUT DROP
            notify-send "VPN Disconnected" "All network traffic blocked!"
        fi
        sleep 5
    done
}
```

Mobile Device Security

Android Security Settings

```bash
# ADB commands for Android hardening
# Enable via Developer Options

# Disable unnecessary permissions
adb shell pm revoke com.example.app android.permission.CAMERA
adb shell pm revoke com.example.app android.permission.RECORD_AUDIO
adb shell pm revoke com.example.app android.permission.ACCESS_FINE_LOCATION

# Disable system apps
adb shell pm disable-user --user 0 com.facebook.system
adb shell pm disable-user --user 0 com.facebook.appmanager

# Enable privacy features
adb shell settings put global location_mode 0  # Disable location
adb shell settings put global bluetooth_on 0  # Disable Bluetooth
adb shell settings put global wifi_scan_always_enabled 0

# Private DNS
adb shell settings put global private_dns_mode hostname
adb shell settings put global private_dns_specifier dns.quad9.net
```

iOS Privacy Configuration

```text
iOS Privacy Settings Checklist:

Settings > Privacy & Security:
✓ Location Services → System Services → Disable all except Find My
✓ Tracking → Disable "Allow Apps to Request to Track"
✓ Analytics & Improvements → Disable all sharing
✓ Apple Advertising → Disable personalized ads

Settings > Wi-Fi:
✓ Private Wi-Fi Address → Enable
✓ Limit IP Address Tracking → Enable

Settings > Cellular:
✓ Wi-Fi Assist → Disable
✓ Personal Hotspot → Disable when not in use

Settings > Bluetooth:
✓ Disable when not in use

Settings > Safari:
✓ Prevent Cross-Site Tracking → Enable
✓ Hide IP Address → From Trackers and Websites
✓ Privacy Preserving Ad Measurement → Disable
```

Monitoring & Alerts

Security Monitoring Script

```python
```

```python
#!/usr/bin/env python3
# security_monitor.py - Real-time security monitoring

import subprocess
import time
import hashlib
import json
from datetime import datetime
import psutil
import socket

class SecurityMonitor:
    def __init__(self):
        self.baseline = self.create_baseline()
        self.alerts = []

    def create_baseline(self):
        """Create system baseline"""
        return {
            'open_ports': self.get_open_ports(),
            'processes': self.get_processes(),
            'connections': self.get_connections(),
            'firewall_rules': self.get_firewall_rules()
        }

    def get_open_ports(self):
        """Get list of open ports"""
        connections = psutil.net_connections()
        ports = set()
        for conn in connections:
            if conn.status == 'LISTEN':
                ports.add(conn.laddr.port)
        return list(ports)

    def get_processes(self):
        """Get running processes"""
        processes = {}
        for proc in psutil.process_iter(['pid', 'name', 'cmdline']):
            try:
                processes[proc.info['pid']] = {
                    'name': proc.info['name'],
                    'cmdline': proc.info['cmdline']
                }
            except:
                pass
        return processes

    def get_connections(self):
        """Get network connections"""
        connections = []
        for conn in psutil.net_connections():
            if conn.status == 'ESTABLISHED':
                connections.append({
                    'local': f"{conn.laddr.ip}:{conn.laddr.port}",
                    'remote': f"{conn.raddr.ip}:{conn.raddr.port}" if conn.raddr else None,
                    'pid': conn.pid
                })
        return connections

    def get_firewall_rules(self):
        """Get firewall rules checksum"""
        try:
            rules = subprocess.check_output(['iptables', '-L', '-n'],
                                          stderr=subprocess.DEVNULL)
            return hashlib.sha256(rules).hexdigest()
        except:
            return None

    def check_anomalies(self):
        """Check for security anomalies"""
        current = self.create_baseline()

        # Check for new open ports
        new_ports = set(current['open_ports']) - set(self.baseline['open_ports'])
        if new_ports:
            self.alert(f"New ports opened: {new_ports}")

        # Check for suspicious processes
        new_procs = set(current['processes'].keys()) - set(self.baseline['processes'].keys())
        for pid in new_procs:
            proc = current['processes'][pid]
            if self.is_suspicious_process(proc):
                self.alert(f"Suspicious process: {proc['name']} (PID: {pid})")

        # Check for firewall changes
        if current['firewall_rules'] != self.baseline['firewall_rules']:
            self.alert("Firewall rules have been modified!")

        # Check for unusual connections
        for conn in current['connections']:
            if self.is_suspicious_connection(conn):
                self.alert(f"Suspicious connection: {conn}")

    def is_suspicious_process(self, proc):
        """Check if process is suspicious"""
        suspicious_names = ['nc', 'ncat', 'netcat', 'meterpreter', 'mimikatz']
        return any(s in proc['name'].lower() for s in suspicious_names)

    def is_suspicious_connection(self, conn):
        """Check if connection is suspicious"""
        if not conn['remote']:
            return False

        # Check for connections to known bad IPs (example)
        bad_ips = ['0.0.0.0', '255.255.255.255']
        remote_ip = conn['remote'].split(':')[0]

        return remote_ip in bad_ips

    def alert(self, message):
        """Send security alert"""
        timestamp = datetime.now().isoformat()
        alert = {
            'timestamp': timestamp,
            'message': message,
            'severity': 'HIGH'
        }

        self.alerts.append(alert)
        print(f"[ALERT] {timestamp}: {message}")

        # Send desktop notification
        try:
            subprocess.run(['notify-send', 'Security Alert', message])
        except:
            pass

    def run(self):
        """Run continuous monitoring"""
        print("Security monitoring started...")
        while True:
            try:
                self.check_anomalies()
                time.sleep(30)  # Check every 30 seconds
            except KeyboardInterrupt:
                print("\nMonitoring stopped.")
                break
            except Exception as e:
                print(f"Error: {e}")

if __name__ == "__main__":
    monitor = SecurityMonitor()
    monitor.run()
```

Best Practices Summary

Daily Security Checklist

```markdown
## Daily Security Tasks

### Morning
- [ ] Check security monitoring alerts
- [ ] Review firewall logs for anomalies
- [ ] Verify VPN connection before sensitive work
- [ ] Check for system/software updates

### During Work
- [ ] Use unique, strong passwords (password manager)
- [ ] Verify HTTPS on all websites
- [ ] Be cautious with email attachments
- [ ] Lock screen when away from computer

### Evening
- [ ] Review the day's security logs
- [ ] Backup important data
- [ ] Update security software
- [ ] Disable WiFi/Bluetooth if not needed overnight

### Weekly Tasks
- [ ] Run full system antivirus scan
- [ ] Review and update firewall rules
- [ ] Check for unauthorized devices on network
- [ ] Update all software and OS
- [ ] Review account permissions and access
- [ ] Test backup restoration
- [ ] Review security camera footage (if applicable)

### Monthly Tasks
- [ ] Rotate important passwords
- [ ] Review privacy settings on all accounts
- [ ] Audit installed applications
- [ ] Check for data breaches (haveibeenpwned.com)
- [ ] Update security documentation
- [ ] Test incident response procedures
```

Legal Notice

All techniques and tools described in this guide are for legitimate personal security and privacy protection. Users are responsible for ensuring compliance with local laws and regulations. These practices are:

- **MAC Address Rotation**: Built into modern OS for privacy
- **Firewall Configuration**: Standard security practice
- **VPN Usage**: Legal privacy tool in most jurisdictions
- **DNS Security**: Protection against malicious domains
- **System Hardening**: Best practice for all users

Always use these tools responsibly and only on systems you own or have explicit permission to protect.