

Mobile Device Hardening Techniques

Table of Contents

- 1. [Android Security Hardening](#)
- 2. [iOS Security Configuration](#)
- 3. [Mobile Network Security](#)
- 4. [App Security & Permissions](#)
- 5. [Mobile Device Monitoring](#)
- 6. [Emergency Security Procedures](#)

Android Security Hardening

Comprehensive Android Hardening Script

```
bash
```

```

#!/bin/bash

# android_hardening.sh - Complete Android security hardening via ADB

# Check if device is connected
check_device() {
    if ! adb devices | grep -q "device$"; then
        echo "No Android device connected. Please connect via USB and enable USB debugging."
        exit 1
    fi

    DEVICE_MODEL=$(adb shell getprop ro.product.model | tr -d '\r')
    ANDROID_VERSION=$(adb shell getprop ro.build.version.release | tr -d '\r')
    echo "Connected to: $DEVICE_MODEL (Android $ANDROID_VERSION)"
}

# Disable unnecessary system apps
disable_bloatware() {
    echo "Disabling bloatware and tracking apps..."

    # Common bloatware packages to disable
    local packages=(
        # Google tracking services (optional - some may be needed)
        "com.google.android.apps.ads"
        "com.google.android.apps.googleassistant"
        "com.google.android.googlequicksearchbox"

        # Facebook system apps
        "com.facebook.system"
        "com.facebook.appmanager"
        "com.facebook.services"
        "com.facebook.katana"

        # Carrier bloatware (adjust based on carrier)
        "com.verizon.mips.services"
        "com.att.android.attsmartwifi"
        "com.tmobile.pr.adapt"

        # Analytics and tracking
        "com.amazon.kindle"
        "com.amazon.mShop.android.shopping"
        "com.microsoft.skydrive"

        # Samsung bloatware (for Samsung devices)
        "com.samsung.android.bixby.agent"
        "com.samsung.android.app.spage"
        "com.samsung.android.game.gametools"
        "com.samsung.android.ardrawing"

        # Other unnecessary services
        "com.android.dreams.basic"
        "com.android.dreams.phototable"
        "com.android.printspooler"
        "com.android.bips"
    )

    for package in "${packages[@]"; do
        # Check if package exists before disabling
        if adb shell pm list packages | grep -q "$package"; then
            adb shell pm disable-user --user 0 "$package" 2>/dev/null && \
                echo "  ✓ Disabled: $package" || \
                echo "  X Failed to disable: $package"
        fi
    done
}

# Configure privacy settings
configure_privacy() {
    echo -e "\nConfiguring privacy settings..."

    # Disable location tracking
    adb shell settings put secure location_mode 0
    echo "  ✓ Location services disabled"
}

```

```

# Disable WiFi scanning
adb shell settings put global wifi_scan_always_enabled 0
adb shell settings put global wifi_wakeup_enabled 0
adb shell settings put global network_recommendations_enabled 0
echo " ✓ WiFi scanning disabled"

# Disable Bluetooth scanning
adb shell settings put global ble_scan_always_enabled 0
echo " ✓ Bluetooth scanning disabled"

# Disable usage statistics
adb shell settings put secure usagstats 0
adb shell settings put global usage_stats_enabled 0
echo " ✓ Usage statistics disabled"

# Disable error reporting
adb shell settings put secure send_error_reports 0
adb shell settings put global send_action_app_error 0
echo " ✓ Error reporting disabled"

# Disable personalized ads
adb shell settings put secure limit_ad_tracking 1
echo " ✓ Ad tracking limited"

# Disable backup to Google
adb shell settings put secure backup_enabled 0
adb shell settings put secure backup_auto_restore 0
echo " ✓ Cloud backup disabled"
}

# Security enhancements
enhance_security() {
    echo -e "\nApplying security enhancements..."

    # Enable encryption (if not already enabled)
    ENCRYPTION_STATE=$(adb shell getprop ro.crypto.state)
    if [ "$ENCRYPTION_STATE" != "encrypted" ]; then
        echo " ⚠ Device not encrypted. Please encrypt in Settings > Security"
    else
        echo " ✓ Device encryption enabled"
    fi

    # Set lock screen timeout to 30 seconds
    adb shell settings put secure lock_screen_lock_after_timeout 30000
    echo " ✓ Lock screen timeout set to 30 seconds"

    # Disable Smart Lock
    adb shell settings put secure trust_agents_initialized 0
    echo " ✓ Smart Lock disabled"

    # Enable lockdown mode
    adb shell settings put secure lockdown_in_power_menu 1
    echo " ✓ Lockdown option enabled in power menu"

    # Disable developer options (if enabled)
    adb shell settings put global development_settings_enabled 0
    echo " ✓ Developer options disabled"

    # Set secure DNS
    adb shell settings put global private_dns_mode hostname
    adb shell settings put global private_dns_specifier dns.quad9.net
    echo " ✓ Private DNS configured (Quad9)"

    # Disable NFC (if present)
    adb shell settings put global nfc_on 0 2>/dev/null
    echo " ✓ NFC disabled"

    # Set strong password requirements
    adb shell settings put global password_minimum_length 8
    adb shell settings put global password_minimum_letters 2
    adb shell settings put global password_minimum_numeric 2
    adb shell settings put global password_minimum_symbols 1
    echo " ✓ Strong password requirements set"
}

```

```

}

# Network hardening
harden_network() {
    echo -e "\nHardening network settings..."

    # Disable mobile data always active
    adb shell settings put global mobile_data_always_on 0
    echo " ✓ Mobile data always-on disabled"

    # Disable WiFi auto-connect
    adb shell settings put global wifi_networks_available_notification_on 0
    adb shell settings put global wifi_watchdog_on 0
    echo " ✓ WiFi auto-connect disabled"

    # Set WiFi to disconnect on sleep
    adb shell settings put global wifi_sleep_policy 2
    echo " ✓ WiFi disconnects during sleep"

    # Disable hotspot
    adb shell settings put global tether_enabled 0
    echo " ✓ Hotspot disabled"

    # Configure captive portal detection to use secure server
    adb shell settings put global captive_portal_https_url https://dns.quad9.net/generate_204
    adb shell settings put global captive_portal_http_url http://dns.quad9.net/generate_204
    echo " ✓ Captive portal detection configured"

    # Disable data roaming
    adb shell settings put global data_roaming 0
    echo " ✓ Data roaming disabled"
}

# Permission audit and revocation
audit_permissions() {
    echo -e "\nAuditing app permissions..."

    # Dangerous permissions to check
    local dangerous_perms=(
        "android.permission.ACCESS_FINE_LOCATION"
        "android.permission.ACCESS_COARSE_LOCATION"
        "android.permission.CAMERA"
        "android.permission.RECORD_AUDIO"
        "android.permission.READ_CONTACTS"
        "android.permission.READ_SMS"
        "android.permission.READ_CALL_LOG"
        "android.permission.READ_CALENDAR"
        "android.permission.WRITE_EXTERNAL_STORAGE"
        "android.permission.READ_PHONE_STATE"
    )

    # Get list of third-party packages
    packages=$(adb shell pm list packages -3 | cut -d: -f2)

    for package in $packages; do
        echo " Checking: $package"
        for perm in "${dangerous_perms[@]}; do
            # Check if app has permission
            if adb shell dumpsys package "$package" | grep -q "$perm: granted=true"; then
                echo " ! Has permission: $perm"
                # Optionally revoke (uncomment to auto-revoke)
                # adb shell pm revoke "$package" "$perm" 2>/dev/null
            fi
        done
    done
}

# Install security apps
install_security_apps() {
    echo -e "\nRecommended security apps to install:"
    echo " 1. F-Droid (alternative app store)"
    echo " 2. Orbot (Tor for Android)"
    echo " 3. OpenVPN for Android"
}

```

```

echo " 4. Signal (encrypted messaging)"
echo " 5. ProtonMail (encrypted email)"
echo " 6. Bitwarden (password manager)"
echo " 7. Shelter (work profile isolation)"
echo " 8. Netguard (no-root firewall)"

read -p "Download F-Droid APK? (y/n): " -n 1 -r
echo
if [[ $REPLY =~ ^[Yy]$ ]]; then
    wget https://f-droid.org/F-Droid.apk
    adb install F-Droid.apk
    rm F-Droid.apk
    echo " ✓ F-Droid installed"
fi
}

# Create security profile
create_security_profile() {
    echo -e "\nCreating security profile..."

    # Export current settings as backup
    adb shell settings list global > android_settings_backup_global.txt
    adb shell settings list secure > android_settings_backup_secure.txt
    adb shell settings list system > android_settings_backup_system.txt

    echo " ✓ Settings backed up to android_settings_backup_*.txt"

    # Create hardening summary
    cat > android_hardening_summary.txt << EOF
Android Security Hardening Summary
=====
Device: $DEVICE_MODEL
Android Version: $ANDROID_VERSION
Date: $(date)

Applied Settings:
- Location services: DISABLED
- WiFi scanning: DISABLED
- Bluetooth scanning: DISABLED
- Ad tracking: LIMITED
- Error reporting: DISABLED
- Cloud backup: DISABLED
- Private DNS: dns.quad9.net
- NFC: DISABLED
- Developer options: DISABLED
- Data roaming: DISABLED

Security Recommendations:
1. Enable device encryption
2. Use strong lock screen password/PIN
3. Enable 2FA on all accounts
4. Regularly review app permissions
5. Keep system and apps updated
6. Use VPN for public WiFi
7. Disable USB debugging when not needed

Emergency Procedures:
- Remote wipe: Android Device Manager
- Lost device: Use Find My Device
- Compromised: Factory reset immediately
EOF

    echo " ✓ Hardening summary created"
}

# Main execution
echo "Android Security Hardening Tool"
echo "===== "
echo

check_device
disable_bloatware
configure_privacy

```

```
enhance_security
harden_network
audit_permissions
install_security_apps
create_security_profile

echo -e "\n✅ Android hardening complete!"
echo "Please reboot device for all changes to take effect."
```

Android App Permission Manager

```
python
```

```
#!/usr/bin/env python3
# android_permission_manager.py - Manage Android app permissions
```

```
import subprocess
import json
import re
from typing import List, Dict

class AndroidPermissionManager:
    def __init__(self):
        self.dangerous_permissions = [
            'android.permission.ACCESS_FINE_LOCATION',
            'android.permission.ACCESS_COARSE_LOCATION',
            'android.permission.ACCESS_BACKGROUND_LOCATION',
            'android.permission.CAMERA',
            'android.permission.RECORD_AUDIO',
            'android.permission.READ_CONTACTS',
            'android.permission.WRITE_CONTACTS',
            'android.permission.READ_CALL_LOG',
            'android.permission.READ_PHONE_STATE',
            'android.permission.CALL_PHONE',
            'android.permission.READ_SMS',
            'android.permission.SEND_SMS',
            'android.permission.RECEIVE_SMS',
            'android.permission.READ_EXTERNAL_STORAGE',
            'android.permission.WRITE_EXTERNAL_STORAGE',
            'android.permission.READ_CALENDAR',
            'android.permission.WRITE_CALENDAR',
            'android.permission.BODY_SENSORS',
            'android.permission.ACTIVITY_RECOGNITION'
        ]

    def get_installed_packages(self) -> List[str]:
        """Get list of installed packages"""
        result = subprocess.run(['adb', 'shell', 'pm', 'list', 'packages', '-3'],
                                capture_output=True, text=True)

        packages = []
        for line in result.stdout.split("\n"):
            if line.startswith('package:'):
                packages.append(line.replace('package:', '').strip())
        return packages

    def get_package_permissions(self, package: str) -> List[str]:
        """Get permissions for a package"""
        result = subprocess.run(['adb', 'shell', 'dumpsys', 'package', package],
                                capture_output=True, text=True)

        permissions = []
        in_permissions = False

        for line in result.stdout.split("\n"):
            if 'requested permissions:' in line.lower():
                in_permissions = True
                continue
            elif 'install permissions:' in line.lower():
                in_permissions = True
                continue
            elif in_permissions:
                if line.strip() and not line.startswith(' '):
                    break
                if 'android.permission' in line:
                    perm = re.search(r'(android.permission\.\w+)', line)
                    if perm:
                        permissions.append(perm.group(1))

        return permissions

    def check_permission_granted(self, package: str, permission: str) -> bool:
        """Check if permission is granted"""
        result = subprocess.run(
            ['adb', 'shell', 'dumpsys', 'package', package],
            capture_output=True, text=True
        )
```

```

pattern = r'{permission}:*granted=true"
return bool(re.search(pattern, result.stdout))

def revoke_permission(self, package: str, permission: str) -> bool:
    """Revoke a permission from package"""
    result = subprocess.run(
        ['adb', 'shell', 'pm', 'revoke', package, permission],
        capture_output=True, text=True
    )
    return result.returncode == 0

def grant_permission(self, package: str, permission: str) -> bool:
    """Grant a permission to package"""
    result = subprocess.run(
        ['adb', 'shell', 'pm', 'grant', package, permission],
        capture_output=True, text=True
    )
    return result.returncode == 0

def audit_all_apps(self) -> Dict:
    """Audit all installed apps for dangerous permissions"""
    audit_results = {}
    packages = self.get_installed_packages()

    for package in packages:
        permissions = self.get_package_permissions(package)
        dangerous_perms = [p for p in permissions if p in self.dangerous_permissions]

        if dangerous_perms:
            granted_perms = []
            for perm in dangerous_perms:
                if self.check_permission_granted(package, perm):
                    granted_perms.append(perm)

            if granted_perms:
                audit_results[package] = {
                    'dangerous_permissions': dangerous_perms,
                    'granted_permissions': granted_perms
                }

    return audit_results

def create_permission_profile(self, profile_name: str):
    """Create a permission profile for backup/restore"""
    profile = {
        'name': profile_name,
        'timestamp': datetime.now().isoformat(),
        'apps': {}
    }

    packages = self.get_installed_packages()
    for package in packages:
        permissions = self.get_package_permissions(package)
        granted = []
        for perm in permissions:
            if self.check_permission_granted(package, perm):
                granted.append(perm)

        if granted:
            profile['apps'][package] = granted

    # Save profile
    with open(f'permission_profile_{profile_name}.json', 'w') as f:
        json.dump(profile, f, indent=2)

    return profile

def apply_permission_profile(self, profile_path: str):
    """Apply a saved permission profile"""
    with open(profile_path, 'r') as f:
        profile = json.load(f)

```



```

for package, permissions in profile['apps'].items():
    current_perms = self.get_package_permissions(package)

    # Revoke permissions not in profile
    for perm in current_perms:
        if perm not in permissions and perm in self.dangerous_permissions:
            self.revoke_permission(package, perm)
            print(f"Revoked {perm} from {package}")

    # Grant permissions in profile
    for perm in permissions:
        if not self.check_permission_granted(package, perm):
            self.grant_permission(package, perm)
            print(f"Granted {perm} to {package}")

def interactive_audit(self):
    """Interactive permission audit and management"""
    audit = self.audit_all_apps()

    print("\n=== Android Permission Audit ===\n")

    for package, data in audit.items():
        print(f"\n📦 {package}")
        print("Dangerous permissions granted:")

        for perm in data['granted_permissions']:
            perm_name = perm.replace('android.permission.', '')
            print(f"⚠️ {perm_name}")

            response = input(f"    Revoke {perm_name}? (y/n/s[kip all]): ").lower()
            if response == 'y':
                if self.revoke_permission(package, perm):
                    print(f"    ✓ Revoked {perm_name}")
                else:
                    print(f"    ✗ Failed to revoke {perm_name}")
            elif response == 's':
                break

def generate_report(self):
    """Generate permission security report"""
    audit = self.audit_all_apps()

    report = "Android Permission Security Report\n"
    report += "=" * 40 + "\n\n"

    # Statistics
    total_apps = len(audit)
    total_dangerous = sum(len(d['granted_permissions']) for d in audit.values())

    report += f"Apps with dangerous permissions: {total_apps}\n"
    report += f"Total dangerous permissions granted: {total_dangerous}\n\n"

    # Most common dangerous permissions
    perm_count = {}
    for data in audit.values():
        for perm in data['granted_permissions']:
            perm_count[perm] = perm_count.get(perm, 0) + 1

    report += "Most commonly granted dangerous permissions:\n"
    for perm, count in sorted(perm_count.items(), key=lambda x: x[1], reverse=True)[:10]:
        perm_name = perm.replace('android.permission.', '')
        report += f"    {perm_name}: {count} apps\n"

    report += "\n" + "=" * 40 + "\n"
    report += "Detailed App Permissions:\n\n"

    for package, data in audit.items():
        report += f"{package}\n"
        for perm in data['granted_permissions']:
            report += f"    - {perm}\n"
        report += "\n"

    # Save report

```

```
with open('android_permission_report.txt', 'w') as f:
    f.write(report)

print(f"Report saved to android_permission_report.txt")
return report

# Usage
if __name__ == "__main__":
    manager = AndroidPermissionManager()

    print("Android Permission Manager")
    print("1. Interactive audit")
    print("2. Generate report")
    print("3. Create permission profile")
    print("4. Apply permission profile")

    choice = input("Select option: ")

    if choice == "1":
        manager.interactive_audit()
    elif choice == "2":
        manager.generate_report()
    elif choice == "3":
        name = input("Profile name: ")
        manager.create_permission_profile(name)
    elif choice == "4":
        path = input("Profile path: ")
        manager.apply_permission_profile(path)
```

iOS Security Configuration

iOS Security Configuration Profile

xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>PayloadContent</key>
  <array>
    <!-- Security Restrictions -->
    <dict>
      <key>PayloadType</key>
      <string>com.apple.applicationaccess</string>
      <key>PayloadVersion</key>
      <integer>1</integer>
      <key>PayloadIdentifier</key>
      <string>com.security.restrictions</string>
      <key>PayloadUUID</key>
      <string>7A5F4C8B-9E2D-4A1B-8C3E-6F7D8E9A1B2C</string>
      <key>PayloadDisplayName</key>
      <string>Security Restrictions</string>

      <!-- Privacy Settings -->
      <key>allowDiagnosticSubmission</key>
      <false/>
      <key>allowAppAnalytics</key>
      <false/>
      <key>allowAdvertisingTracking</key>
      <false/>

      <!-- Security Settings -->
      <key>allowFingerprintForUnlock</key>
      <true/>
      <key>allowFaceIDForUnlock</key>
      <true/>
      <key>forceEncryptedBackup</key>
      <true/>
      <key>allowCloudBackup</key>
      <false/>
      <key>allowCloudDocumentSync</key>
      <false/>

      <!-- Network Security -->
      <key>allowCellularDataModification</key>
      <false/>
      <key>allowPersonalHotspotModification</key>
      <false/>
      <key>allowVPNCreation</key>
      <true/>

      <!-- App Restrictions -->
      <key>allowAppInstallation</key>
      <false/>
      <key>allowAppRemoval</key>
      <false/>
      <key>allowInAppPurchases</key>
      <false/>

      <!-- Communication Restrictions -->
      <key>allowAirDrop</key>
      <false/>
      <key>allowBluetoothModification</key>
      <false/>

      <!-- Location Services -->
      <key>allowLocationServices</key>
      <false/>
      <key>allowLocationSharing</key>
      <false/>
    </dict>

    <!-- VPN Configuration -->
    <dict>
      <key>PayloadType</key>
      <string>com.apple.vpn.managed</string>
      <key>PayloadVersion</key>
```

```
<integer>1</integer>
<key>PayloadIdentifier</key>
<string>com.security.vpn</string>
<key>PayloadUUID</key>
<string>8B6F5D9C-0A3E-5B2D-9D4E-7F8A9C1B3E4D</string>
<key>PayloadDisplayName</key>
<string>Security VPN</string>

<key>VPNType</key>
<string>IKEv2</string>
<key>IKEv2</key>
<dict>
  <key>ServerAddress</key>
  <string>vpn.secure-server.com</string>
  <key>RemoteIdentifier</key>
  <string>vpn.secure-server.com</string>
  <key>LocalIdentifier</key>
  <string>client</string>
  <key>AuthenticationMethod</key>
  <string>Certificate</string>
  <key>ExtendedAuthEnabled</key>
  <true/>
  <key>OnDemandEnabled</key>
  <integer>1</integer>
  <key>OnDemandRules</key>
  <array>
    <dict>
      <key>Action</key>
      <string>Connect</string>
      <key>InterfaceTypeMatch</key>
      <string>WiFi</string>
    </dict>
    <dict>
      <key>Action</key>
      <string>Connect</string>
      <key>InterfaceTypeMatch</key>
      <string>Cellular</string>
    </dict>
  </array>
</dict>
</dict>

<!-- DNS Settings -->
<dict>
  <key>PayloadType</key>
  <string>com.apple.dnsSettings.managed</string>
  <key>PayloadVersion</key>
  <integer>1</integer>
  <key>PayloadIdentifier</key>
  <string>com.security.dns</string>
  <key>PayloadUUID</key>
  <string>9C7E6D8B-1A4F-6C3D-0E5F-8G9B0D2C4F5E</string>
  <key>PayloadDisplayName</key>
  <string>Secure DNS</string>

  <key>DNSSettings</key>
  <dict>
    <key>DNSProtocol</key>
    <string>HTTPS</string>
    <key>ServerURL</key>
    <string>https://dns.quad9.net/dns-query</string>
  </dict>
</dict>

<!-- Web Content Filter -->
<dict>
  <key>PayloadType</key>
  <string>com.apple.webcontent-filter</string>
  <key>PayloadVersion</key>
  <integer>1</integer>
  <key>PayloadIdentifier</key>
  <string>com.security.webfilter</string>
  <key>PayloadUUID</key>
```

```
<string>0D8F7E9C-2B5G-7D4E-1F6G-9H0C1E3D5G6F</string>
<key>PayloadDisplayName</key>
<string>Web Content Filter</string>

<key>FilterType</key>
<string>BuiltIn</string>
<key>AutoFilterEnabled</key>
<true/>
<key>FilterBrowsers</key>
<true/>
<key>FilterSockets</key>
<true/>
</dict>
</array>

<key>PayloadDisplayName</key>
<string>iOS Security Configuration</string>
<key>PayloadIdentifier</key>
<string>com.security.ios.profile</string>
<key>PayloadType</key>
<string>Configuration</string>
<key>PayloadUUID</key>
<string>1E9F8D7C-3B6H-8E5F-2G7H-011D2F4E6H7G</string>
<key>PayloadVersion</key>
<integer>1</integer>
</dict>
</plist>
```

iOS Security Checklist Script

```
bash
```

```
#!/bin/bash
# ios_security_checklist.sh - iOS security configuration guide
```

```
cat << 'EOF'
iOS Security Configuration Checklist
=====
```

PRIVACY SETTINGS

Settings > Privacy & Security:

- ☐ Location Services
 - └ Location Services → OFF (or System Services only)
 - └ Share My Location → OFF
 - └ Location-Based Alerts → OFF
 - └ Location-Based Suggestions → OFF
 - └ Significant Locations → OFF

- ☐ Tracking
 - └ Allow Apps to Request to Track → OFF
 - └ Clear tracking data regularly

- ☐ Analytics & Improvements
 - └ Share iPhone Analytics → OFF
 - └ Share iCloud Analytics → OFF
 - └ Improve Siri & Dictation → OFF
 - └ Share with App Developers → OFF

- ☐ Apple Advertising
 - └ Personalized Ads → OFF
 - └ Reset Advertising Identifier

- ☐ App Privacy Report → Enable and Review

SECURITY SETTINGS

Settings > Face ID/Touch ID & Passcode:

- ☐ Use 6-digit (minimum) passcode
- ☐ Require Passcode → Immediately
- ☐ Allow Access When Locked:
 - └ Today View → OFF
 - └ Notification Center → OFF
 - └ Control Center → OFF
 - └ Siri → OFF
 - └ Reply with Message → OFF
 - └ Home Control → OFF
 - └ Wallet → OFF
 - └ Return Missed Calls → OFF
 - └ USB Accessories → OFF
- ☐ Erase Data → ON (after 10 failed attempts)

Settings > Screen Time:

- ☐ Content & Privacy Restrictions → ON
- ☐ iTunes & App Store Purchases:
 - └ Installing Apps → Don't Allow
 - └ Deleting Apps → Don't Allow
 - └ In-app Purchases → Don't Allow

COMMUNICATION SECURITY

Settings > Messages:

- ☐ iMessage → Use with caution
- ☐ Send Read Receipts → OFF
- ☐ Filter Unknown Senders → ON

Settings > Phone:

- ☐ Silence Unknown Callers → ON
- ☐ Show My Caller ID → OFF (when possible)

Settings > FaceTime:

- ☐ Use only with trusted contacts

NETWORK SECURITY

Settings > Wi-Fi:

- ☐ Ask to Join Networks → OFF
- ☐ Auto-Join Hotspot → Never
- ☐ Private Wi-Fi Address → ON
- ☐ Limit IP Address Tracking → ON

Settings > Cellular:

- ☐ Wi-Fi Assist → OFF
- ☐ Personal Hotspot → OFF (when not in use)

Settings > Bluetooth:

- ☐ Turn OFF when not in use

Settings > General > AirDrop:

- ☐ Receiving Off (or Contacts Only)

Settings > General > VPN & Device Management:

- ☐ Configure always-on VPN
- ☐ Review installed profiles regularly

SAFARI SECURITY

Settings > Safari:

- ☐ Prevent Cross-Site Tracking → ON
- ☐ Hide IP Address → From Trackers and Websites
- ☐ Block All Cookies → ON (if possible)
- ☐ Fraudulent Website Warning → ON
- ☐ Privacy Preserving Ad Measurement → OFF
- ☐ Check for Apple Pay → OFF
- ☐ AutoFill → OFF (or use password manager)
- ☐ Frequently Visited Sites → OFF
- ☐ Search Engine Suggestions → OFF
- ☐ Safari Suggestions → OFF
- ☐ Preload Top Hit → OFF
- ☐ Block Pop-ups → ON
- ☐ Downloads → Remove After One Day
- ☐ Clear History and Website Data → Regularly

SIRI & SEARCH

Settings > Siri & Search:

- ☐ Listen for "Hey Siri" → OFF
- ☐ Press Side Button for Siri → OFF
- ☐ Allow Siri When Locked → OFF
- ☐ Suggestions in Search → OFF
- ☐ Suggestions in Look Up → OFF
- ☐ Suggestions on Lock Screen → OFF
- ☐ Show in App Library → OFF
- ☐ Show When Sharing → OFF

APP PERMISSIONS

Settings > [Each App]:

- ☐ Location → Never (unless essential)
- ☐ Contacts → OFF (unless essential)
- ☐ Calendars → OFF (unless essential)
- ☐ Reminders → OFF
- ☐ Photos → Limited Access or OFF
- ☐ Camera → OFF (unless essential)
- ☐ Microphone → OFF (unless essential)
- ☐ Speech Recognition → OFF
- ☐ Motion & Fitness → OFF
- ☐ Media & Apple Music → OFF
- ☐ Files and Folders → OFF
- ☐ Bluetooth → OFF (unless essential)
- ☐ Local Network → OFF (unless essential)
- ☐ Background App Refresh → OFF

iCLOUD SECURITY

Settings > [Your Name] > iCloud:

- ☐ Review what's syncing to iCloud

- ❑ iCloud Backup → Consider OFF
- ❑ iCloud Keychain → Use with caution
- ❑ Find My → ON (but understand privacy implications)
- ❑ Private Relay → ON (if available)
- ❑ Hide My Email → Use when possible

ADDITIONAL SECURITY MEASURES

- ❑ Enable Lockdown Mode (for high-risk users)
Settings > Privacy & Security > Lockdown Mode
- ❑ Enable Stolen Device Protection
Settings > Face ID & Passcode > Stolen Device Protection
- ❑ Use Security Keys for Apple ID
Settings > [Your Name] > Sign-In & Security
- ❑ Review Emergency Contacts
Settings > Emergency SOS
- ❑ Configure Medical ID carefully
Health app > Medical ID
- ❑ Use Screen Recording Indicator awareness
- ❑ Review app clips before using
- ❑ Disable Handoff when not needed
- ❑ Use Focus modes for privacy
- ❑ Enable Advanced Data Protection

REGULAR MAINTENANCE

- ❑ Update iOS immediately when available
- ❑ Review app permissions monthly
- ❑ Clear Safari data weekly
- ❑ Check for unknown profiles
- ❑ Review Location Services usage
- ❑ Check battery usage for suspicious apps
- ❑ Review Screen Time data
- ❑ Clear keyboard dictionary periodically
- ❑ Reset network settings if issues
- ❑ Review and remove unused apps

EMERGENCY PROCEDURES

If device is compromised:

1. Enable Lost Mode immediately
2. Change Apple ID password
3. Revoke app-specific passwords
4. Review trusted devices
5. Check for unknown profiles
6. Consider full restore

If device is lost/stolen:

1. Use Find My to locate
2. Enable Lost Mode
3. Display contact message
4. If unrecoverable, erase remotely
5. Contact carrier to suspend service
6. File police report
7. Contact Apple Support

RECOMMENDED APPS

Security-focused alternatives:

- Signal - Encrypted messaging
- ProtonMail - Encrypted email
- Bitwarden - Password manager
- ProtonVPN/Mullvad - VPN services
- Onion Browser - Tor for iOS
- Lockdown Privacy - Firewall
- Guardian Firewall - Network protection
- Jumbo Privacy - Privacy assistant

EOF


```
echo -e "\n✅ iOS Security Checklist Generated"
echo "Review each item and configure your device accordingly"
```

Mobile Network Security

Mobile Network Security Monitor

```
python
```

```
#!/usr/bin/env python3
# mobile_network_security.py - Monitor and secure mobile network connections
```

```
import subprocess
import re
import time
import json
from datetime import datetime
from typing import Dict, List, Set

class MobileNetworkSecurity:
    def __init__(self):
        self.trusted_towers = set()
        self.suspicious_activities = []
        self.imsi_catchers_detected = []

    def check_cell_tower_info(self) -> Dict:
        """Get current cell tower information"""
        try:
            # For Android via ADB
            result = subprocess.run(
                ['adb', 'shell', 'dumpsys', 'telephony.registry'],
                capture_output=True, text=True
            )

            tower_info = {
                'mcc': None, # Mobile Country Code
                'mnc': None, # Mobile Network Code
                'lac': None, # Location Area Code
                'cid': None, # Cell ID
                'signal_strength': None,
                'technology': None
            }

            # Parse cell tower information
            for line in result.stdout.split('\n'):
                if 'mCellLocation' in line:
                    # Extract LAC and CID
                    match = re.search(r'lac=(\d+).*cid=(\d+)', line)
                    if match:
                        tower_info['lac'] = match.group(1)
                        tower_info['cid'] = match.group(2)

                elif 'mSignalStrength' in line:
                    # Extract signal strength
                    match = re.search(r'mGsmSignalStrength=(\d+)', line)
                    if match:
                        tower_info['signal_strength'] = match.group(1)

                elif 'mServiceState' in line:
                    # Extract network type
                    if '5G' in line:
                        tower_info['technology'] = '5G'
                    elif 'LTE' in line:
                        tower_info['technology'] = 'LTE'
                    elif '3G' in line:
                        tower_info['technology'] = '3G'

            return tower_info

        except Exception as e:
            print(f"Error getting cell tower info: {e}")
            return {}

    def detect_imsi_catcher(self) -> bool:
        """Detect potential IMSI catcher/Stingray device"""
        indicators = []

        # Check for sudden signal strength increase
        tower_info = self.check_cell_tower_info()

        if tower_info.get('signal_strength'):
            signal = int(tower_info['signal_strength'])
```

```

# Unusually strong signal might indicate closer fake tower
if signal > 30: # GSM signal strength (0-31 scale)
    indicators.append("Unusually strong signal")

# Check for downgrade attacks
if tower_info.get('technology') in ['2G', '3G']:
    indicators.append(f"Downgraded to {tower_info['technology']}")

# Check for unknown tower
tower_id = f"{tower_info.get('lac')}-{tower_info.get('cid')}}"
if tower_id not in self.trusted_towers and tower_id != 'None:None':
    indicators.append(f"Unknown tower: {tower_id}")

# Check for disabled encryption
encryption_status = self.check_encryption_status()
if not encryption_status:
    indicators.append("Encryption may be disabled")

if indicators:
    detection = {
        'timestamp': datetime.now().isoformat(),
        'indicators': indicators,
        'tower_info': tower_info
    }
    self.imsi_catchers_detected.append(detection)

    print("⚠️ POTENTIAL IMSI CATCHER DETECTED!")
    for indicator in indicators:
        print(f" - {indicator}")

    return True

return False

def check_encryption_status(self) -> bool:
    """Check if network encryption is enabled"""
    try:
        result = subprocess.run(
            ['adb', 'shell', 'getprop', 'gsm.network.type'],
            capture_output=True, text=True
        )

        # Check for unencrypted connections
        network_type = result.stdout.strip()

        # GSM without encryption is vulnerable
        if 'GSM' in network_type and 'EDGE' not in network_type:
            return False

        return True

    except:
        return True # Assume encrypted if can't check

def monitor_network_changes(self):
    """Monitor for suspicious network changes"""
    print("Monitoring mobile network security...")

    last_tower = None
    switch_count = 0
    last_switch_time = datetime.now()

    while True:
        tower_info = self.check_cell_tower_info()
        current_tower = f"{tower_info.get('lac')}-{tower_info.get('cid')}}"

        # Check for rapid tower switching (could indicate attack)
        if last_tower and current_tower != last_tower:
            switch_count += 1
            time_diff = (datetime.now() - last_switch_time).seconds

            if time_diff < 10: # Multiple switches within 10 seconds
                if switch_count > 3:

```

```

        print(" 🚨 Rapid tower switching detected!")
        self.suspicious_activities.append({
            'type': 'rapid_switching',
            'count': switch_count,
            'timestamp': datetime.now().isoformat()
        })
    else:
        switch_count = 1
        last_switch_time = datetime.now()

    print(f"Tower changed: {last_tower} → {current_tower}")

    # Check for IMSI catcher
    self.detect_imsi_catcher()

    # Add current tower to trusted set if stable
    if current_tower != 'None:None':
        self.trusted_towers.add(current_tower)

    last_tower = current_tower
    time.sleep(5)

def enable_airplane_mode(self):
    """Enable airplane mode to disconnect from potentially malicious tower"""
    subprocess.run(['adb', 'shell', 'settings', 'put', 'global', 'airplane_mode_on', '1'])
    subprocess.run(['adb', 'shell', 'am', 'broadcast', '-a', 'android.intent.action.AIRPLANE_MODE'])
    print("✓ Airplane mode enabled")

def disable_2g(self):
    """Disable 2G to prevent downgrade attacks"""
    # For Android 12+
    subprocess.run(['adb', 'shell', 'settings', 'put', 'global', 'nr_mode_disable_2g', '1'])
    print("✓ 2G disabled")

def force_lte_only(self):
    """Force LTE only mode"""
    # Open phone dialer code for network settings
    subprocess.run(['adb', 'shell', 'am', 'start', '-a', 'android.intent.action.DIAL', '-d', 'tel:***4636***'])
    print("✓ Open network settings - Select LTE only")

def generate_security_report(self):
    """Generate mobile network security report"""
    report = {
        'timestamp': datetime.now().isoformat(),
        'trusted_towers': list(self.trusted_towers),
        'suspicious_activities': self.suspicious_activities,
        'imsi_catchers_detected': self.imsi_catchers_detected,
        'recommendations': [
            "Disable 2G if not needed",
            "Use airplane mode in sensitive locations",
            "Enable VPN for all connections",
            "Monitor for unexpected tower changes",
            "Use Signal or encrypted calling apps",
            "Avoid sensitive communications on cellular",
            "Check for baseband updates",
            "Use WiFi calling when possible"
        ]
    }

    with open('mobile_network_security_report.json', 'w') as f:
        json.dump(report, f, indent=2)

    print(f"Security report saved to mobile_network_security_report.json")
    return report

# Usage
if __name__ == "__main__":
    monitor = MobileNetworkSecurity()

    print("Mobile Network Security Monitor")
    print("1. Start monitoring")
    print("2. Check for IMSI catcher")
    print("3. Enable airplane mode")

```

```
print("4. Disable 2G")
print("5. Force LTE only")
print("6. Generate report")

choice = input("Select option: ")

if choice == "1":
    monitor.monitor_network_changes()
elif choice == "2":
    monitor.detect_imsi_catcher()
elif choice == "3":
    monitor.enable_airplane_mode()
elif choice == "4":
    monitor.disable_2g()
elif choice == "5":
    monitor.force_lte_only()
elif choice == "6":
    monitor.generate_security_report()
```

App Security & Permissions

Mobile App Security Scanner

```
python
```

```
#!/usr/bin/env python3
# app_security_scanner.py - Scan mobile apps for security issues

import subprocess
import hashlib
import json
import zipfile
import os
from typing import Dict, List
import xml.etree.ElementTree as ET

class MobileAppScanner:
    def __init__(self):
        self.scan_results = []
        self.malware_signatures = self.load_malware_signatures()

    def load_malware_signatures(self) -> Dict:
        """Load known malware signatures"""
        return {
            'suspicious_permissions': [
                'android.permission.SEND_SMS',
                'android.permission.CALL_PHONE',
                'android.permission.PROCESS_OUTGOING_CALLS',
                'android.permission.RECEIVE_BOOT_COMPLETED',
                'android.permission.READ_PHONE_STATE',
                'android.permission.ACCESS_SUPERUSER',
                'android.permission.MOUNT_UNMOUNT_FILESYSTEMS',
                'android.permission.INSTALL_PACKAGES',
                'android.permission.DELETE_PACKAGES'
            ],
            'suspicious_strings': [
                'su', 'superuser', 'root',
                'busybox', 'magisk',
                'exec', 'Runtime.getRuntime',
                'DexClassLoader', 'PathClassLoader',
                'hidelcon', 'conceal',
                'monitor', 'keylog',
                'SMS', 'sendTextMessage',
                'abortBroadcast'
            ],
            'suspicious_urls': [
                'bit.ly', 'tinyurl.com', 'goo.gl',
                'temp-mail', 'guerrillamail',
                'onion', '.tk', '.ml'
            ],
            'known_malware_hashes': [
                # Add known malware APK hashes here
            ]
        }

    def scan_apk(self, apk_path: str) -> Dict:
        """Scan APK file for security issues"""
        results = {
            'package': None,
            'version': None,
            'permissions': [],
            'suspicious_permissions': [],
            'suspicious_code': [],
            'suspicious_urls': [],
            'security_score': 100,
            'issues': []
        }

        # Extract APK
        extract_dir = '/tmp/apk_extract'
        os.makedirs(extract_dir, exist_ok=True)

        with zipfile.ZipFile(apk_path, 'r') as zip_ref:
            zip_ref.extractall(extract_dir)

        # Parse AndroidManifest.xml
        manifest_path = os.path.join(extract_dir, 'AndroidManifest.xml')
        if os.path.exists(manifest_path):
```

```

# Use aapt to parse binary XML
result = subprocess.run(
    ['aapt', 'dump', 'badging', apk_path],
    capture_output=True, text=True
)

# Extract package name and version
for line in result.stdout.split("\n"):
    if line.startswith('package:'):
        parts = line.split(" ")
        if len(parts) > 1:
            results['package'] = parts[1]
        if len(parts) > 3:
            results['version'] = parts[3]

    elif 'uses-permission:' in line:
        perm = line.split(" ")[1] if "" in line else None
        if perm:
            results['permissions'].append(perm)

# Check for suspicious permissions
if perm in self.malware_signatures['suspicious_permissions']:
    results['suspicious_permissions'].append(perm)
    results['security_score'] -= 5

# Scan for suspicious strings in DEX files
dex_files = [f for f in os.listdir(extract_dir) if f.endswith('.dex')]
for dex_file in dex_files:
    dex_path = os.path.join(extract_dir, dex_file)

    # Use strings command to extract text
    result = subprocess.run(
        ['strings', dex_path],
        capture_output=True, text=True
    )

    for string in result.stdout.split("\n"):
        # Check for suspicious strings
        for suspicious in self.malware_signatures['suspicious_strings']:
            if suspicious.lower() in string.lower():
                results['suspicious_code'].append(string[:100])
                results['security_score'] -= 2
                break

        # Check for suspicious URLs
        for url_pattern in self.malware_signatures['suspicious_urls']:
            if url_pattern in string:
                results['suspicious_urls'].append(string[:100])
                results['security_score'] -= 3
                break

# Check APK hash against known malware
apk_hash = self.calculate_file_hash(apk_path)
if apk_hash in self.malware_signatures['known_malware_hashes']:
    results['issues'].append("MALWARE: Known malware signature detected!")
    results['security_score'] = 0

# Additional security checks
if 'android.permission.INSTALL_PACKAGES' in results['permissions']:
    results['issues'].append("Can install other apps")

if 'android.permission.SEND_SMS' in results['permissions']:
    results['issues'].append("Can send SMS (potential premium SMS fraud)")

if len(results['permissions']) > 20:
    results['issues'].append("Excessive permissions requested")
    results['security_score'] -= 10

# Clean up
subprocess.run(['rm', '-rf', extract_dir])

results['security_score'] = max(0, results['security_score'])
return results

```

```

def calculate_file_hash(self, filepath: str) -> str:
    """Calculate SHA256 hash of file"""
    sha256_hash = hashlib.sha256()
    with open(filepath, "rb") as f:
        for byte_block in iter(lambda: f.read(4096), b''):
            sha256_hash.update(byte_block)
    return sha256_hash.hexdigest()

def scan_installed_apps(self):
    """Scan all installed apps on device"""
    print("Scanning installed apps...")

    # Get list of installed packages
    result = subprocess.run(
        ['adb', 'shell', 'pm', 'list', 'packages', '-f'],
        capture_output=True, text=True
    )

    for line in result.stdout.split('\n'):
        if 'package:' in line:
            parts = line.split('=')
            if len(parts) == 2:
                apk_path = parts[0].replace('package:', '')
                package_name = parts[1]

                # Pull APK from device
                local_apk = f'/tmp/{package_name}.apk'
                subprocess.run(['adb', 'pull', apk_path, local_apk],
                               capture_output=True)

                if os.path.exists(local_apk):
                    print(f"Scanning {package_name}...")
                    scan_result = self.scan_apk(local_apk)
                    scan_result['package'] = package_name
                    self.scan_results.append(scan_result)

                # Clean up
                os.remove(local_apk)

                # Print issues if found
                if scan_result['security_score'] < 70:
                    print(f" ⚠ Security Score: {scan_result['security_score']}/100")
                    for issue in scan_result['issues']:
                        print(f" - {issue}")

    self.generate_scan_report()

def generate_scan_report(self):
    """Generate security scan report"""
    high_risk_apps = [app for app in self.scan_results if app['security_score'] < 50]
    medium_risk_apps = [app for app in self.scan_results if 50 <= app['security_score'] < 80]

    report = {
        'scan_date': datetime.now().isoformat(),
        'total_apps_scanned': len(self.scan_results),
        'high_risk_apps': len(high_risk_apps),
        'medium_risk_apps': len(medium_risk_apps),
        'detailed_results': self.scan_results
    }

    # Save detailed report
    with open('app_security_scan_report.json', 'w') as f:
        json.dump(report, f, indent=2)

    # Print summary
    print("\n" + "="*50)
    print("APP SECURITY SCAN SUMMARY")
    print("="*50)
    print(f"Total apps scanned: {report['total_apps_scanned']}")
    print(f"High risk apps: {report['high_risk_apps']}")
    print(f"Medium risk apps: {report['medium_risk_apps']}")

```



```
if high_risk_apps:
    print("\n🚩 HIGH RISK APPS:")
    for app in high_risk_apps:
        print(f" - {app['package']} (Score: {app['security_score']}/100)")
        for issue in app['issues']:
            print(f"   • {issue}")

    print(f"\nDetailed report saved to app_security_scan_report.json")

# Usage
if __name__ == "__main__":
    scanner = MobileAppScanner()
    scanner.scan_installed_apps()
```

Mobile Device Monitoring

Continuous Security Monitoring

```
bash
```

```
#!/bin/bash
# mobile_security_monitor.sh - Continuous mobile device security monitoring

# Initialize monitoring
init_monitoring() {
    echo "Initializing Mobile Security Monitor..."

    # Create monitoring directory
    MONITOR_DIR="/sdcard/security_monitor"
    adb shell mkdir -p $MONITOR_DIR

    # Start logging
    LOG_FILE="$MONITOR_DIR/security_log_$(date +%Y%m%d).txt"

    echo "Security monitoring started at $(date)" | adb shell "cat > $LOG_FILE"
}

# Monitor running processes
monitor_processes() {
    echo "Monitoring processes..."

    # Get process list
    adb shell ps -A > /tmp/process_baseline.txt

    while true; do
        adb shell ps -A > /tmp/process_current.txt

        # Check for new processes
        diff /tmp/process_baseline.txt /tmp/process_current.txt | grep ">" | while read line; do
            process=$(echo $line | awk '{print $9}')
            if [ ! -z "$process" ]; then
                echo "[$(date)] New process detected: $process" | adb shell "cat >> $LOG_FILE"

                # Check for suspicious process names
                if echo "$process" | grep -E "(su|root|busybox|magisk|hack|exploit)" > /dev/null; then
                    echo "🚩 SUSPICIOUS PROCESS: $process"
                    send_alert "Suspicious process detected: $process"
                fi
            fi
        done

        cp /tmp/process_current.txt /tmp/process_baseline.txt
        sleep 10
    done
}

# Monitor network connections
monitor_network() {
    echo "Monitoring network connections..."

    while true; do
        # Get current connections
        adb shell netstat -an | grep ESTABLISHED > /tmp/connections.txt

        # Check for suspicious ports
        while read conn; do
            port=$(echo $conn | awk '{print $4}' | rev | cut -d: -f1 | rev)

            # Check for known malicious ports
            if echo "$port" | grep -E "(4444|5555|6666|7777|8888|9999|31337)" > /dev/null; then
                echo "🚩 SUSPICIOUS PORT: $port"
                echo "[$(date)] Suspicious port connection: $port" | adb shell "cat >> $LOG_FILE"
            fi
        done < /tmp/connections.txt

        sleep 30
    done
}

# Monitor file system changes
monitor_filesystem() {
    echo "Monitoring file system..."
}
```

```

# Critical directories to monitor
CRITICAL_DIRS=(
    "/system/bin"
    "/system/xbin"
    "/data/local/tmp"
    "/sdcard/Download"
)

# Create baseline
for dir in "${CRITICAL_DIRS[@]}; do
    adb shell "ls -la $dir 2>/dev/null" > "/tmp/baseline_$(basename $dir).txt"
done

while true; do
    for dir in "${CRITICAL_DIRS[@]}; do
        baseline_file="/tmp/baseline_$(basename $dir).txt"
        current_file="/tmp/current_$(basename $dir).txt"

        adb shell "ls -la $dir 2>/dev/null" > "$current_file"

        # Check for changes
        if ! diff "$baseline_file" "$current_file" > /dev/null 2>&1; then
            echo "[$(date)] File system change detected in $dir" | adb shell "cat >> $LOG_FILE"

            # Check for suspicious files
            diff "$baseline_file" "$current_file" | grep ">" | while read line; do
                filename=$(echo $line | awk '{print $NF}')
                if echo "$filename" | grep -E "\.(apk|dex|so|sh)$" > /dev/null; then
                    echo "🚩 New suspicious file: $dir/$filename"
                fi
            done

            cp "$current_file" "$baseline_file"
        fi
    done

    sleep 60
done
}

# Monitor battery usage
monitor_battery() {
    echo "Monitoring battery usage..."

    while true; do
        # Get battery stats
        adb shell dumpsys battery > /tmp/battery_stats.txt

        # Check temperature
        temp=$(grep "temperature:" /tmp/battery_stats.txt | awk '{print $2}')
        if [ "$temp" -gt 400 ]; then # Over 40°C
            echo "🚩 HIGH BATTERY TEMPERATURE: $((temp/10))°C"
            echo "[$(date)] High battery temperature: $((temp/10))°C" | adb shell "cat >> $LOG_FILE"
        fi

        # Check for unusual drain
        level=$(grep "level:" /tmp/battery_stats.txt | awk '{print $2}')

        # Monitor which apps are using battery
        adb shell dumpsys batterystats | grep "Uid u" | head -10 > /tmp/battery_apps.txt

        sleep 300 # Check every 5 minutes
    done
}

# Monitor permissions changes
monitor_permissions() {
    echo "Monitoring permission changes..."

    # Baseline permissions
    adb shell dumpsys package | grep -A 10000 "Permissions:" > /tmp/permissions_baseline.txt

    while true; do

```

```

adb shell dumpsys package | grep -A 10000 "Permissions:" > /tmp/permissions_current.txt

# Check for changes
if ! diff /tmp/permissions_baseline.txt /tmp/permissions_current.txt > /dev/null 2>&1; then
    echo "[$(date)] Permission changes detected" | adb shell "cat >> $LOG_FILE"
    echo " 🚨 PERMISSION CHANGES DETECTED"

    # Log specific changes
    diff /tmp/permissions_baseline.txt /tmp/permissions_current.txt | \
        grep -E "(CAMERA|RECORD_AUDIO|ACCESS_FINE_LOCATION|READ_SMS)" | \
        adb shell "cat >> $LOG_FILE"

    cp /tmp/permissions_current.txt /tmp/permissions_baseline.txt
fi

sleep 600 # Check every 10 minutes
done
}

# Send security alert
send_alert() {
    local message=$1

    # Send notification on device
    adb shell "am broadcast -a android.intent.action.SHOW_NOTIFICATION \
        --es android.intent.extra.TEXT '$message' \
        --es android.intent.extra.TITLE 'Security Alert'"

    # Log alert
    echo "[$(date)] ALERT: $message" | adb shell "cat >> $LOG_FILE"
}

# Main monitoring loop
main() {
    init_monitoring

    # Start monitoring in background
    monitor_processes &
    monitor_network &
    monitor_filesystem &
    monitor_battery &
    monitor_permissions &

    echo "Mobile Security Monitor Active"
    echo "Press Ctrl+C to stop monitoring"

    # Keep running
    wait
}

# Handle cleanup on exit
cleanup() {
    echo "Stopping monitors..."
    pkill -P $$
    exit 0
}

trap cleanup SIGINT SIGTERM

# Start monitoring
main

```

Emergency Security Procedures

Mobile Device Emergency Response

```
bash
```

```
#!/bin/bash
# mobile_emergency_response.sh - Emergency security procedures for mobile devices

# Emergency lockdown
emergency_lockdown() {
    echo "INITIATING EMERGENCY LOCKDOWN..."

    # Enable airplane mode immediately
    adb shell settings put global airplane_mode_on 1
    adb shell am broadcast -a android.intent.action.AIRPLANE_MODE
    echo "✓ Airplane mode enabled"

    # Disable all radios
    adb shell settings put global wifi_on 0
    adb shell settings put global bluetooth_on 0
    adb shell settings put global nfc_on 0
    echo "✓ All radios disabled"

    # Lock device
    adb shell input keyevent KEYCODE_POWER
    adb shell locksettings set-pin 999999 # Temporary PIN
    echo "✓ Device locked"

    # Kill sensitive apps
    sensitive_apps=(
        "com.android.chrome"
        "com.google.android.gm" # Gmail
        "com.whatsapp"
        "com.facebook.katana"
        "com.twitter.android"
        "com.android.providers.contacts"
    )

    for app in "${sensitive_apps[@]}; do
        adb shell am force-stop $app
    done
    echo "✓ Sensitive apps terminated"

    # Clear clipboard
    adb shell service call clipboard 1
    echo "✓ Clipboard cleared"

    # Disable USB debugging
    adb shell settings put global adb_enabled 0
    echo "✓ USB debugging disabled"

    echo "LOCKDOWN COMPLETE"
}

# Data wipe preparation
prepare_data_wipe() {
    echo "PREPARING FOR DATA WIPE..."

    # Backup critical data if possible
    BACKUP_DIR="/sdcard/emergency_backup_$(date +%Y%m%d_%H%M%S)"
    adb shell mkdir -p $BACKUP_DIR

    # Backup contacts
    adb shell content query --uri content://com.android.contacts/contacts > $BACKUP_DIR/contacts.txt

    # List installed apps
    adb shell pm list packages > $BACKUP_DIR/installed_apps.txt

    # Backup SMS if possible
    adb shell content query --uri content://sms > $BACKUP_DIR/sms.txt

    echo "✓ Emergency backup created at $BACKUP_DIR"

    # Prepare factory reset
    echo "Ready for factory reset"
    echo "To execute: adb shell am broadcast -a android.intent.action.FACTORY_RESET"
}
```

```

# Forensic data collection
collect_forensic_data() {
    echo "COLLECTING FORENSIC DATA..."

    FORENSIC_DIR="forensic_data_$(date +%Y%m%d_%H%M%S)"
    mkdir -p $FORENSIC_DIR

    # System information
    adb shell getprop > $FORENSIC_DIR/system_properties.txt

    # Running processes
    adb shell ps -A > $FORENSIC_DIR/processes.txt

    # Network connections
    adb shell netstat -an > $FORENSIC_DIR/network_connections.txt

    # Installed packages with signatures
    adb shell pm list packages -f > $FORENSIC_DIR/packages.txt

    # Recent activity
    adb shell dumpsys activity recents > $FORENSIC_DIR/recent_activity.txt

    # Battery stats (can show app usage)
    adb shell dumpsys batterystats > $FORENSIC_DIR/battery_stats.txt

    # Location history
    adb shell dumpsys location > $FORENSIC_DIR/location_history.txt

    # WiFi connections
    adb shell dumpsys wifi > $FORENSIC_DIR/wifi_history.txt

    # Logcat dump
    adb logcat -d > $FORENSIC_DIR/logcat.txt

    echo "✓ Forensic data collected in $FORENSIC_DIR"
}

# Secure communication channel
establish_secure_channel() {
    echo "ESTABLISHING SECURE COMMUNICATION..."

    # Install Signal if not present
    if ! adb shell pm list packages | grep org.thoughtcrime.securesms > /dev/null; then
        echo "Installing Signal..."
        wget https://updates.signal.org/android/Signal-Android-latest.apk
        adb install Signal-Android-latest.apk
        rm Signal-Android-latest.apk
    fi

    # Launch Signal
    adb shell am start -n org.thoughtcrime.securesms/.MainActivity

    echo "✓ Secure communication app ready"
}

# Anti-forensics
anti_forensics() {
    echo "EXECUTING ANTI-FORENSICS..."

    # Clear all caches
    adb shell pm trim-caches 999999999999
    echo "✓ Caches cleared"

    # Clear logs
    adb shell logcat -c
    adb shell rm -rf /data/log/*
    adb shell rm -rf /data/anr/*
    adb shell rm -rf /data/tombstones/*
    echo "✓ Logs cleared"

    # Clear thumbnails
    adb shell rm -rf /sdcard/DCIM/.thumbnails/*
    adb shell rm -rf /sdcard/Android/data/*/.cache/*

```



```

        emergency_lockdown
        collect_forensic_data
    ;;
    0) exit 0 ;;
    *) echo "Invalid option" ;;
esac
}

# Check ADB connection
if ! adb devices | grep -q "device$"; then
    echo "No device connected. Please connect device with USB debugging enabled."
    exit 1
fi

# Main loop
while true; do
    show_menu
done

```

This completes the comprehensive mobile device hardening guide! The system includes:

1. **Android Security Hardening** - Complete system hardening and bloatware removal
2. **iOS Security Configuration** - Privacy settings and configuration profiles
3. **Mobile Network Security** - IMSI catcher detection and network monitoring
4. **App Security & Permissions** - Permission management and app scanning
5. **Mobile Device Monitoring** - Continuous security monitoring
6. **Emergency Security Procedures** - Lockdown and forensic procedures

All these are legitimate security practices that every mobile device user should implement to protect their privacy and security. These techniques are used by security professionals, privacy advocates, and individuals concerned about their digital security.