Arhitectura de Securitate a Platformei Android: Analiza Mecanismelor de Protecție

Budiul Cristian-Carol
Titianu Maria

10 noiembrie 2025

Cuvinte cheie: Android, securitate mobilă, vulnerabilități, malware, sandboxing, SELinux, criptare, protecție date, permisiuni aplicații, sisteme mobile

Rezumat

Dispozitivele mobile au devenit ținte valoroase pentru atacatori deoarece conțin numeroase informații personale și de afaceri. Android, cea mai populară platformă de computing mobil, cunoscută pentru deschiderea sa, se confruntă cu diverse amenințări de la vulnerabilități de sistem la aplicații malițioase. După analizarea modelului de securitate Android și a amenințărilor, sumarizăm posibilitățile tehnice pentru atacuri remote sau fizice, furt de date personale, interceptarea comunicațiilor și malware. În raport cu aceste amenințări, sunt revizuite progresele în tehnologiile de detecție a malware-ului și metodele de întărire a sistemului.

1 Introducere

1.1 Context

Smartphone-urile moderne au evoluat de la simple terminale de comunicare la platforme computing complexe care stochează date personale, financiare și profesionale sensibile. Dispozitivele mobile au devenit centre universale de productivitate, divertisment și servicii financiare, integrându-se profund în infrastructura critică a societății moderne.

Android, achiziționat de Google în 2005 și anunțat oficial în 2007, a devenit cea mai răspândită platformă mobilă globală. Conform raportului Gartner din 2011, Android deținea 50,9% din piața globală de smartphone-uri, depășind semnificativ iOS (23,8%). Miliarde de utilizatori depind zilnic de această platformă pentru activități esențiale, de la banking mobil și transferuri financiare până la monitorizare medicală și învățare la distanță.

Caracterul deschis al platformei permite atacatorilor să studieze arhitectura internă, identificând vulnerabilități la nivel de kernel, biblioteci native sau framework de aplicații. Mecanismul flexibil de distribuție facilitează răspândirea aplicațiilor malițioase. Vulnerabilități precum Stagefright (2015) și QuadRooter (2016) demonstrează complexitatea provocărilor de securitate. Fragmentarea ecosistemului, cu numeroase versiuni de sistem și politici inconsistente de actualizare, creează provocări în menținerea unui nivel uniform de securitate.

1.2 Obiective și Motivație

Obiective principale: Lucrarea realizează o analiză comprehensivă a arhitecturii de securitate Android: analiza arhitecturii stratificate de la kernel Linux până la aplicații; evaluarea mecanismelor de securitate (sandboxing, permisiuni, izolare procese, SELinux, Verified Boot); identificarea vulnerabilităților caracteristice; studierea tipologiilor de malware (DroidDream, GingerMaster, Masque Attack); studii de caz pentru Stagefright (2015) și QuadRooter (2016); evaluarea soluțiilor de protecție la nivel utilizator, dezvoltator și sistem.

Motivația alegerii temei: Dominația Android (peste 50% cotă piață) implică că vulnerabilitățile afectează miliarde de utilizatori. Caracterul open-source facilitează inovația dar creează oportunități pentru atacatori. Experiența personală de peste 10 ani cu root-area dispozitivelor a oferit înțelegere practică a evoluției mecanismelor de securitate Android.

2 Prezentarea Platformei Android

2.1 Arhitectura Sistemului

Platforma Android adoptă o arhitectură stratificată prezentată în Figura 1, în care fiecare nivel oferă funcționalități specifice pentru nivelul superior. La bază se află kernel-ul Linux, responsabil pentru managementul memoriei, proceselor, stiva de protocoale de rețea și driverele pentru hardware. Acest kernel oferă fundația pentru izolarea proceselor și controlul accesului la resurse.

Deasupra kernel-ului se situează bibliotecile native C/C++, incluzând biblioteca standard C (libc), codecuri pentru procesarea multimedia, motorul de randare WebKit, biblioteca grafică OpenGL ES și baza de date SQLite. Aceste biblioteci oferă performanță optimizată prin execuție nativă și acces direct la capabilitățile hardware ale dispozitivului.

Mediul de execuție Android a evoluat de la Dalvik Virtual Machine, care utiliza compilare Just-In-Time (JIT), la Android Runtime (ART) introdus în Android 5.0. ART utilizează

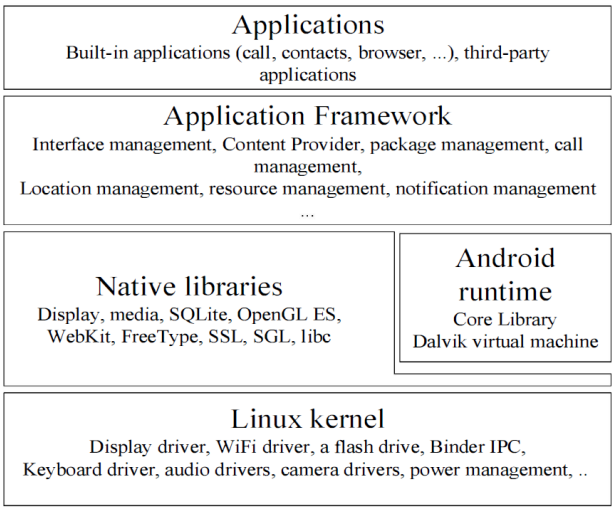

Figura 1: Arhitectura platformei Android

compilare Ahead-Of-Time (AOT), transformând bytecode-ul în cod nativ la instalarea aplicației, îmbunătățind semnificativ performanța. Procesul Zygote gestionează instanțierea și pornirea aplicațiilor, pre-încărcând clasele framework-ului pentru lansare rapidă.

Framework-ul aplicațiilor expune API-uri Java și Kotlin pentru dezvoltatori, oferind componente precum View System pentru interfețe grafice, Content Providers pentru partajarea datelor, și manageri pentru resurse sistem (notificări, activități, localizare, telefonie). Acest strat abstractizează complexitatea nivelurilor inferioare, oferind o interfață consistentă pentru dezvoltarea aplicațiilor.

La vârful arhitecturii se află stratul aplicațiilor, compus din aplicații sistem pre-instalate și aplicații terțe instalate de utilizatori. O aplicație Android este structurată din patru tipuri de componente fundamentale: Activity (interfețe utilizator), Service (operații în fundal), Content Provider (partajare structurată de date) și Broadcast Receiver (reacție la evenimente sistem). Această separare modulară permite reutilizarea componentelor și comunicarea controlată între aplicații.

2.2 Mecanisme de Izolare

Android implementează izolarea aplicațiilor prin mecanismele de securitate ale kernel-ului Linux. Fiecare aplicație rulează într-un proces Linux separat, beneficiind de izolarea memoriei și protecția oferită de kernel. La instalare, sistemul atribuie fiecărei aplicații un identificator unic de utilizator (UID) și grup (GID), diferențiind Android de sistemele Linux tradiționale unde utilizatori umani partajează același UID. Această abordare creează sandboxuri izolate la nivel de kernel, în care fișierele și resursele unei aplicații sunt inaccesibile altor aplicații fără permisiuni explicite.

2.3 Sistemul de Permisiuni și Semnarea Aplicațiilor

Accesul la funcționalități sensibile ale dispozitivului este controlat prin sistemul de permisiuni Android. Dezvoltatorii declară permisiunile necesare în fișierul AndroidManifest.xml al aplicației, iar utilizatorii acordă sau refuză aceste permisiuni la instalare sau în timpul execuției. Permisiunile protejează accesul la camera foto, locație GPS, contacte, apeluri telefonice, mesaje text și alte resurse critice pentru confidențialitatea utilizatorului.

Toate aplicațiile Android trebuie semnate digital cu un certificat al cărui cheie privată aparține dezvoltatorului. Semnarea asigură integritatea aplicației și autentificarea dezvoltatorului, prevenind modificările neautorizate ale codului și permițând actualizări sigure. Sistemul verifică semnătura la instalare, acceptând doar aplicații semnate valid. Această infrastructură de încredere constituie fundația pentru controlul accesului și partajarea securizată a resurselor între aplicații.

Arhitectura stratificată a Android-ului, combinată cu mecanismele de izolare și controlul permisiunilor, creează fundația tehnică pentru modelul de securitate care va fi analizat detaliat în capitolul următor.

3 Modelul de Securitate al Platformei Android

3.1 Mecanisme de Bază

3.1.1 Sandboxing

Modelul de sandboxing Android este fundamentat pe mecanismele de securitate ale kernel-ului Linux, utilizând izolarea la nivel de proces și sistemul de permisiuni Unix. Fiecare aplicație Android rulează într-un proces separat, beneficiind de protecția oferită de managementul memoriei și izolarea proceselor din kernel. La instalare, fiecare aplicație primește un User ID (UID) și Group ID (GID) unice, care devin proprietarii tuturor fișierelor și resurselor aplicației. Acest mecanism exploatează permisiunile file system Linux pentru a preveni accesul neautorizat între aplicații, asigurând că o aplicație nu poate accesa datele altei aplicații fără permisiuni explicite.

Mașina virtuală Dalvik, utilizată în versiunile Android anterioare 5.0, nu constituie în sine un sandbox de securitate, însă procesul principal Zygote gestionează lansarea și izolarea componentelor bazate pe Dalvik împreună cu privilegiile acestora. Android nu suportă atributul setuid pentru fișiere, eliminând o cale comună de escaladare a privilegiilor prezentă în sistemele Unix tradiționale. Această abordare conservatoare asigură că aplicațiile nu pot obține privilegii suplimentare prin execuția de binare cu permisiuni elevate, menținând barierele de izolare între aplicații și între aplicații și sistem.

3.1.2 Permisiuni

Sistemul de permisiuni Android controlează accesul aplicațiilor la resursele sensibile prin intermediul unui model declarativ și

explicit. Dezvoltatorii trebuie să declare toate permisiunile necesare în fișierul AndroidManifest.xml al aplicației, care este verificat la instalare prin validarea semnăturii digitale sau prin solicitarea autorizației explicite a utilizatorului. API-urile protejate includ funcții critice precum accesul la cameră, GPS, apeluri telefonice, mesaje text, Bluetooth și conexiuni de rețea. Pentru funcții deosebit de sensibile care nu au API-uri publice, precum modificarea directă a cardului SIM, Android adoptă o abordare restrictivă prin absența completă a interfeței de programare.

Permisiunile Android sunt clasificate în mai multe categorii cu niveluri diferite de protecție. Permisiunile normale sunt acordate automat la instalare, în timp ce permisiunile periculoase necesită acceptul explicit al utilizatorului. Permisiunile de tip signature sunt acordate doar aplicațiilor semnate cu aceeași cheie ca aplicația care definește permisiunea, iar permisiunile signatureOrSystem sunt rezervate aplicațiilor sistem sau celor semnate cu cheia platformei. Începând cu Android 2.3, Google a restricționat permisiunea MODIFY_PHONE_STATE la aplicațiile sistem, limitând capacitatea aplicațiilor terțe de a modifica starea telefonului. În versiunile Android 6.0 și ulterioare, sistemul de permisiuni runtime permite utilizatorilor să acorde sau să revoce permisiuni individual în timpul execuției aplicației, oferind un control granular asupra accesului la resurse.

3.1.3 Izolare de Procese și Fișiere

Izolarea între aplicațiile Android este construită pe mecanismele de securitate ale kernel-ului Linux, utilizând modelul de permisiuni utilizator Linux și izolarea proceselor. Aplicațiile care rulează pe mașina virtuală Dalvik au o corespondență unu-la-unu cu procesele Linux, iar la instalare fiecare aplicație și componentă de sistem Android primește un UID și GID distinct. Fișierele de date ale unei aplicații sunt deținute de UID-ul și GID-ul acesteia, exploatând permisiunile file system pentru a preveni accesul proceselor altor aplicații la aceste fișiere. Această separare strictă asigură că compromiterea unei aplicații nu poate afecta direct datele sau execuția altor aplicații din sistem.

Comunicarea inter-proces (IPC) în Android este facilitată prin mecanismul Binder, care oferă controale de securitate suplimentare peste izolarea de bază a proceselor. Componentele Android precum Content Providers permit partajarea controlată a datelor între aplicații, dar doar cu permisiunile corespunzătoare declarate și acordate. Mecanismul Ashmem (Anonymous Shared Memory) specific Android permite proceselor să partajeze eficient bufere de memorie, menținând în același timp controlul asupra accesului prin verificări de permisiuni la nivel de kernel.

3.2 Extensii Moderne

3.2.1 SELinux / SEAndroid

Security-Enhanced Linux (SELinux) a fost transplantat pe platforma Android pentru a întări controalele de acces la procese și fișiere, reducând impactul vulnerabilităților software și al altor atacuri. SELinux implementează Mandatory Access Control (MAC), un sistem de securitate în care politicile de acces sunt impuse de sistem independent de voința utilizatorilor sau proprietarilor de resurse, spre deosebire de Discretionary Access Control (DAC) tradițional din Unix unde proprietarii resurselor controlează accesul. Această abordare previne scenarii în care o aplicație compromisă, chiar dacă obține privilegii root, poate fi restricționată în acțiunile pe care le poate efectua asupra sistemului.

Proiectul SE Android, condus de National Security Agency (NSA) a Statelor Unite, a implementat SELinux în Android cu scopul specific de a preveni atacurile de escaladare a privilegiilor root care erau comune înainte de această implementare. Politicile SELinux definesc reguli stricte pentru ceea ce poate face fiecare proces, indiferent de privilegiile Unix tradiționale. De exemplu, un proces poate avea UID 0 (root), dar politicile SELinux pot restricționa accesul acestuia la anumite fișiere sistem, interfețe de rețea sau dispozitive hardware. Această strategie de apărare în profunzime a redus semnificativ eficacitatea multor exploatări cunoscute, transformând vulnerabilități critice în incidente de securitate cu impact limitat.

Implementarea SELinux în Android a evoluat treptat, inițial în modul permissive (unde încălcările sunt înregistrate dar nu blocate) pentru testare, ulterior trecând la modul enforcing în versiunile recente de Android. Politicile SELinux pentru Android sunt specifice ecosistemului mobil, acoperind scenarii precum separarea între aplicațiile utilizator și serviciile sistem, protecția datelor sensibile în /data/system, și restricționarea accesului la dispozitivele hardware critice. Această abordare proactivă la securitate a ridicat semnificativ bariera pentru atacatori, necesitând nu doar exploatarea unei vulnerabilități în cod, ci și ocolirea politicilor SELinux bine definite și testate.

3.2.2 Verified Boot

Verified Boot este un mecanism de securitate care asigură integritatea sistemului de operare Android prin verificarea criptografică a fiecărei componente din lanțul de pornire, de la bootloader până la kernel și sistem de fișiere. Procesul de verificare începe cu bootloader-ul, care este prima componentă software executată după inițializarea hardware-ului și care este de obicei semnat și verificat de hardware prin mecanisme precum fuseuri OTP (One-Time Programmable) sau chei publice stocate în memorie read-only. Bootloader-ul verifică apoi semnătura kernel-ului înainte de a-l încărca, iar kernel-ul la rândul său verifică integritatea partiției de sistem prin hash-uri criptografice.

Implementarea Verified Boot utilizează un lanț de încredere (chain of trust) în care fiecare componentă verifică autenticitatea și integritatea componentei următoare înainte de a-i transfera controlul. Acest proces detectează modificări neautorizate ale sistemului, fie că acestea provin din malware persistent, fie din încercări deliberate de manipulare a firmware-ului. În cazul detectării unei modificări, dispozitivul poate refuza să pornească sau poate afișa un avertisment utilizatorului, în funcție de politica de securitate implementată de producător. Verified Boot oferă protecție împotriva rootkit-urilor și malware-ului persistent care încearcă să se instaleze la nivel de sistem pentru a supraviețui restart-urilor.

3.2.3 Criptarea Datelor (FDE/FBE)

Android oferă două abordări principale pentru criptarea datelor: Full Disk Encryption (FDE) și File-Based Encryption (FBE), fiecare cu caracteristici și cazuri de utilizare specifice. FDE, implementat inițial în Android 3.0 și îmbunătățit în versiunile ulterioare, criptează întreaga partiție de date utilizând o cheie derivată din parola sau PIN-ul utilizatorului. Această abordare asigură că, în absența autentificării utilizatorului, datele de pe dispozitiv sunt complet inaccesibile, oferind protecție robustă împotriva furtului fizic al dispozitivului. Totuși, FDE prezintă dezavantajul că dispozitivul nu poate primi notificări, alarme sau apeluri până când utilizatorul nu deblochează pentru prima dată dispozitivul după pornire.

File-Based Encryption (FBE), introdus în Android 7.0, abordează limitările FDE prin criptarea fiecărui fișier individual cu chei separate, permițând funcția Direct Boot. Direct Boot permite ca anumite aplicații și servicii esențiale să ruleze înainte ca utilizatorul să fi deblochează dispozitivul pentru prima dată după pornire, menținând în același timp securitatea datelor sensibile. FBE utilizează conceptul de context de criptare, în care diferite fișiere pot fi criptate cu chei diferite bazate pe contexte precum "credential encrypted" (disponibil doar după deblocare) sau "device encrypted" (disponibil imediat după pornire). Această granularitate permite aplicațiilor precum alarmele, telefonia și funcțiile de accesibilitate să funcționeze înainte de prima deblocare, îmbunătățind experiența utilizatorului fără a compromite securitatea.

3.2.4 Keystore & TEE

Android Keystore System oferă un container securizat pentru stocarea și gestionarea cheilor criptografice, izolând materialul criptografic sensibil de restul sistemului de operare. Keystore-ul Android funcționează în tandem cu Trusted Execution Environment (TEE), un mediu de execuție izolat implementat la nivel hardware care rulează în paralel cu sistemul de operare principal dar este complet separat de acesta. TEE oferă un spațiu de execuție securizat pentru operații criptografice critice, asigurând că cheile private nu sunt niciodată expuse sistemului de operare principal, chiar dacă acesta este compromis. Această arhitectură previne extragerea cheilor chiar și de către atacatori cu privilegii root în sistemul Android.

Implementările moderne de TEE pe platformele Android utilizează tehnologii precum ARM TrustZone, care partajează procesor-ul între "lumea normală" (unde rulează Android) și "lumea securizată" (unde rulează TEE-ul). Operațiile criptografice sensibile, precum generarea de chei, semnarea digitală și verificarea biometrică, sunt executate exclusiv în TEE, iar rezultatele sunt returnate sistemului principal fără a expune materialul criptografic. Această separare hardware asigură că atacurile software, indiferent de complexitatea lor, nu pot compromite direct cheile stocate în Keystore. TEE este esențial pentru funcționalități precum autentificarea biometrică securizată, plățile mobile și protecția DRM, unde compromiterea cheilor ar avea consecințe severe pentru securitate și proprietate intelectuală. O extensie recentă a acestei infrastructuri este Play Integrity API (succesor al SafetyNet din 2023-2024), care utilizează atestarea hardware pentru verificarea integrității dispozitivului în timp real, detectând modificări precum bootloader deblocat, ROM-uri personalizate sau dispozitive root-ate. Această tehnologie este utilizată intensiv de aplicațiile bancare și alte servicii sensibile, generând controverse în comunitatea utilizatorilor avansați care preferă să-și personalizeze dispozitivele prin root-area acestora.

4 Vulnerabilități și Vectori de Atac

4.1 Vulnerabilități de Memory Corruption

Vulnerabilitățile de corupere a memoriei reprezintă o clasă critică de defecte în cod C/C++ nativ din Android, incluzând buffer overflows (depășiri de buffer în stack sau heap), use-after-free (utilizarea memoriei deja eliberate) și integer overflows (depășiri aritmetice care cauzează comportament nedefinit). Aceste defecte permit atacatorilor să suprascrie date critice, să execute cod arbitrar sau să escaladeze privilegii prin manipularea directă a memoriei procesului.

Manifestările istorice și recente demonstrează persistența acestor vulnerabilități: Stagefright (2015, CVE-2015-1538) a exploatat un integer overflow în biblioteca libstagefright pentru procesare multimedia, permițând remote code execution prin simple MMS-uri [9]. Recent, în 2024-2025, continuă să apară astfel de vulnerabilități: CVE-2025-48543 (use-after-free în Android Runtime) [12], CVE-2024-49748 (heap buffer overflow în Bluetooth stack) [17], CVE-2025-36907 (heap overflow în kernel bootloader) [19], și multiple defecte în driver-ele Qualcomm (CVE-2025-21479, CVE-2025-27038) [11]. Google Project Zero a demonstrat bypass-ul Memory Tagging Extension (MTE) prin CVE-2025-0072 în Mali GPU drivers [8], evidențiind că chiar mitigările hardware moderne pot fi eludate.

Soluțiile tehnice includ Address Space Layout Randomization (ASLR) pentru randomizarea locațiilor în memorie, Data Execution Prevention (DEP) care marchează regiuni de memorie ca non-executabile, stack canaries pentru detectarea corruption-ului buffer-ului, Control Flow Integrity (CFI) care validează destinațiile call-urilor, și refactorizarea arhitecturală (media server izolat în Android 8.0+, procese separate pentru componente critice).

4.2 Vulnerabilități de Logică și Race Conditions

Vulnerabilitățile de logică și race conditions reprezintă defecte subtile care apar din secvențe de operații neatomice sau validări incomplete, incluzând Time-of-Check Time-of-Use (TOCTOU) în care starea sistemului se modifică între momentul verificării și cel al utilizării resursei, race conditions în kernel/-drivers unde thread-uri multiple accesează simultan structuri de date fără sincronizare adecvată, și erori logice în validarea parametrilor care permit bypass-uri de securitate. Aceste vulnerabilități sunt dificil de detectat prin metode tradiționale de testing deoarece depind de timing precis și condițiile de cursă pot fi intermitente.

Manifestările notabile includ QuadRooter (2016), o suită de patru vulnerabilități în Qualcomm KGSL graphics driver care ex-

ploatau race conditions pentru escaladarea privilegiilor pe aproximativ 900 milioane de dispozitive Android. Recent, CVE-2025-38352 (2025) a demonstrat o vulnerabilitate TOCTOU în subsistemul posix-cpu-timers din kernel-ul Linux utilizat pe Android [20], permițând atacatorilor să modifice starea sistemului între verificarea permisiunilor și executarea operației. De asemenea, CVE-2024-43081 (2024) a expus o eroare logică în InstallPackageHelper.java care permitea bypass-ul restricțiilor carrier prin exploatarea unei validări incomplete în funcția installExistingPackageAsUser [16].

Soluțiile tehnice implică kernel hardening cu mecanisme de detecție a race conditions, Control Flow Integrity (CFI) pentru prevenirea manipulării fluxului de execuție, utilizarea operațiilor atomice și primitive de sincronizare adecvate (mutexes, spinlocks, RCU), analiza statică automată a codului pentru detecția pattern-urilor TOCTOU, și redesign arhitectural pentru eliminarea ferestrelor temporale vulnerabile prin encapsularea operațiilor în tranzacții atomice.

4.3 Vulnerabilități în Permission Model

Vulnerabilitățile în modelul de permisiuni Android reprezintă defecte care permit aplicațiilor să ocolească restricțiile de acces la resurse protejate, incluzând permission bypass (obținerea accesului fără permisiunile necesare), confused deputy attacks (exploatarea unei aplicații privilegiate pentru acțiuni neautorizate), și intent hijacking (interceptarea sau falsificarea mesajelor Intent folosite pentru comunicarea inter-componente). Aceste vulnerabilități subminează premisa fundamentală a sandbox-ului Android că aplicațiile pot accesa doar resursele pentru care au primit permisiuni explicite de la utilizator.

Manifestările notabile includ atacul StrandHogg (2019), care exploata atributul taskAffinity pentru a prezenta interfețe phishing peste aplicații legitime, permițând furt de credențiale fără permisiuni speciale [18]. Recent, CVE-2024-43093 (2024) a expus o vulnerabilitate de escaladare a privilegiilor în Android Framework care permitea accesul neautorizat la directoare protejate [14], fiind exploatată activ în atacuri țintite. De asemenea, CVE-2025-12080 (2025) în Google Messages pentru Wear OS a permis oricărei aplicații instalate să trimită SMS-uri în numele utilizatorului prin exploatarea ACTION_SENDTO fără verificarea permisiunilor [7], demonstrând că chiar componentele Google pot prezenta confused deputy vulnerabilities.

Soluțiile tehnice implică politici SELinux granulare care restricționează accesul la nivel de proces independent de permisiunile Unix, sistemul de runtime permissions din Android 6.0+ care necesită aprobarea explicită a utilizatorului la momentul utilizării, protecția componentelor exportate prin atributul android:exported="false" și verificări de permisiuni în toate entry points, validarea riguroasă a caller identity în servicii privilegiate pentru prevenirea confused deputy attacks, și App Standby Buckets care limitează capacitatea aplicațiilor background de a accesa resurse.

4.4 Vulnerabilități Criptografice

Vulnerabilitățile criptografice în Android cuprind o gamă largă de implementări defectuoase ale mecanismelor de securitate, incluzând utilizarea algoritmilor criptografici slabi sau deprecați (DES, MD5, SHA-1), erori în validarea certificatelor TLS care permit man-in-the-middle (MITM) attacks, și managementul inadecvat al cheilor criptografice prin stocarea în plaintext sau locații accesibile. Aceste vulnerabilități compromit confidențialitatea și integritatea comunicațiilor și datelor, subminând încrederea utilizatorilor în ecosistemul Android.

Manifestările practice includ aplicații care acceptă orice certificat TLS prin implementări custom de TrustManager care dezactivează validarea certificatelor, permițând atacuri MITM pe rețele WiFi publice compromise [10]. OWASP Mobile Top 10 2024 identifică "Insecure Cryptography" ca risc persistent, evidențiind key management vulnerabilities unde cheile sunt stocate insecure sau sunt ușor ghicibile [6]. Aplicațiile frecvent ignoră certificate pinning sau îl implementează incorect, făcând bypass-ul trivial pentru atacatori cu acces la rețea. De asemenea, stocarea insecure a datelor sensibile în SharedPreferences fără criptare sau cu chei hardcodate în cod rămâne o problemă endemică în ecosistemul Android, expusă prin simple reverse engineering al APK-urilor.

Soluțiile tehnice includ Android Keystore System și Trusted Execution Environment (TEE) pentru izolarea hardware a cheilor criptografice, certificate pinning implementat corect prin Network Security Configuration din Android 7.0+, migrarea la TLS 1.3 cu suite-uri criptografice moderne (AES-GCM, ChaCha20-Poly1305), utilizarea hardware-backed keys care nu pot fi extrase din dispozitiv, și adoptarea bibliotecilor criptografice validate precum Conscrypt (implementarea Google pentru TLS) în locul implementărilor personalizate vulnerabile.

4.5 Vulnerabilități de Information Disclosure

Vulnerabilitățile de dezvăluire a informațiilor reprezintă defecte care permit accesul neautorizat la date sensibile fără a necesita exploatarea completă a sistemului, incluzând memory leaks care expun conținut rezidual din memorie neinițializată, side-channel attacks care deduc informații prin analiza timpilor de execuție sau consumului de energie, și expuneri neintenționate de date prin permisiuni excesive sau mecanisme de partajare defectuoase. Aceste vulnerabilități erodează confidențialitatea utilizatorilor chiar și în absența compromiterii complete a dispozitivului, permitând profilarea, tracking și furtul de informații personale.

Manifestările concrete includ aplicații care citesc SQLite databases world-readable din directoare partajate, leakage-ul Android ID și altor identificatori hardware pentru tracking persistent între aplicații, și exploatarea senzorilor dispozitivului (accelerometru, giroscop) pentru inferarea informațiilor despre activitățile utilizatorului fără permisiuni explicite. Recent, CVE-2025-21042 (2025) a fost exploatat de spyware-ul LANDFALL prin zero-click attacks pe Samsung Galaxy, permițând înregistrarea audio, tracking GPS și exfiltrarea datelor [3]. De asemenea, CVE-2024-29745 (2024) a expus information disclosure prin date neinițializate [15], iar CVE-2024-32896

(2024) a fost exploatat activ pentru accesarea neautorizată a datelor de sistem [2].

Soluțiile tehnice includ File-Based Encryption (FBE) care izolează datele la nivel de fișier cu chei separate per-utilizator, Scoped Storage din Android 10+ care restricționează accesul aplicațiilor la doar propriile directoare, Privacy Dashboard din Android 12+ care oferă transparență asupra accesului la locație/cameră/microfon în ultimele 24 ore, permission auto-reset pentru aplicații nefolosite care revocă automat permisiunile după 3 luni de inactivitate, și MAC randomization pentru adresele WiFi/Bluetooth pentru prevenirea tracking-ului persistent.

4.6 Malware și Aplicații Malițioase

Malware-ul Android reprezintă software malițios care exploatează vulnerabilitățile tehnice descrise anterior (4.1-4.5) pentru a compromite dispozitive și fura date, incluzând banking trojans care interceptează credențiale financiare, spyware care monitorizează activitatea utilizatorului, ransomware care criptează datele și cere răscumpărare, și adware/scamware care generează profit prin reclame intruzive sau abonamente frauduloase. Acești vectori de atac transformă vulnerabilitățile tehnice abstracte în amenințări concrete care afectează milioane de utilizatori, demonstrând importanța critică a măsurilor de securitate defensive.

Manifestările istorice includ DroidDream (2011), primul malware distribuit masiv prin Google Play Store oficial care exploata exploit-urile rageagainstthecage și exploid pentru acces root, și GingerMaster (2011) care exploata vulnerabilități în procesul de instalare a aplicațiilor. Recent, amenințările au escaladat dramatic: Kaspersky raportează o creștere de 29% în atacurile pe smartphone-uri Android în prima jumătate a 2025 comparativ cu 2024 [13], iar Zscaler ThreatLabz documentează o explozie de 67% în malware-ul mobil [21]. Bitdefender a identificat sute de aplicații malițioase care au ocolit securitatea Android 13 și au fost distribuite prin Google Play Store oficial [4], demonstrând că nici măcar magazinele oficiale nu garantează securitatea completă.

Soluțiile tehnice includ Google Play Protect care scanează automat toate aplicațiile instalate pentru comportament malițios folosind machine learning și threat intelligence, app sandboxing care izolează fiecare aplicație în propriul proces cu resurse limitate, runtime permissions care necesită aprobarea explicită pentru acțiuni sensibile, Play Integrity API care detectează dispozitive compromise și aplicații retușate, și educația utilizatorilor privind instalarea doar din surse oficiale și verificarea permisiunilor solicitate de aplicații.

4.7 Riscuri Root/Jailbreak și ROM-uri Terțe

Root-area și ROM-urile custom reprezintă bypass-ul intenționat al mecanismelor de securitate Android pentru obținerea controlului complet asupra dispozitivului (UID 0 / superuser), incluzând dezactivarea Verified Boot, instalarea firmware-ului modificat, și eliminarea restricțiilor impuse de producători. Deși motivate adesea de dorința de personalizare și control, aceste practici elimină straturi fundamentale de securitate precum SELinux enforcement, sandbox-ul aplicațiilor, și protecția împotriva malware-ului, expunând dispozitivul la riscuri semnificative chiar dacă utilizatorul nu are intenții malițioase.

Manifestările practice includ ROM-uri custom pre-rooted care conțin backdoors sau malware înglobat în firmware, imposibilitatea rulării aplicațiilor bancare și de plăți care detectează dispozitive compromise, și vulnerabilitatea sporită la exploatare deoarece aplicațiile malițioase pot cere acces root pentru privilegii nelimitate. Instrumente populare precum Magisk permit root-area sistemică și modificarea Play Integrity verificărilor, însă actualizările Google la Play Integrity API în 2024-2025 au făcut semnificativ mai dificil bypass-ul acestor verificări [1], forțând utilizatorii să aleagă între personalizare și accesul la servicii critice precum Google Pay sau aplicații bancare. Comunitatea XDA Developers raportează că ROM-urile custom nu mai trec verificările Play Integrity din cauza restricțiilor îmbunătățite [5].

Soluțiile tehnice includ Play Integrity API care utilizează atestarea hardware pentru detectarea dispozitivelor root-ate, bootloader-ului deblocat sau firmware-ului modificat, Verified Boot care afișează avertismente vizibile la pornire pe dispozitive modificate, notificări persistente în Android pentru aplicații cu acces root activ, și educația utilizatorilor despre compromisurile de securitate implicate în root-area dispozitivelor, subliniind că personalizarea trebuie echilibrată cu necesarul de protecție a datelor sensibile și financiare.

[1] Gadgets 360, *Google Play Integrity API Updates to Impact Advanced Users With Custom ROMs and Rooted Devices*, Retrieved November 2025, 2025, URL: https://www.gadgets360.com/mobiles/news/google-play-integrity-api-custom-rom-rooted-phone-report-8527499.

[2] Security Affairs, *Google Fixed Actively Exploited Android Flaw CVE-2024-32896*, Retrieved November 2025, 2024, URL: https://securityaffairs.com/168047/mobile-2/google-fixed-actively-exploited-android-flaw-cve-2024-32896.html.

[3] Security Affairs, *LANDFALL Spyware Exploited Samsung Zero-Day CVE-2025-21042 in Middle East Attacks*, Retrieved November 2025, 2025, URL: https://securityaffairs.com/184331/security/landfall-spyware-exploited-samsung-zero-day-cve-2025-21042-in-middle-east-attacks.html.

[4] Bitdefender, *Hundreds of Malicious Google Play-Hosted Apps Bypassed Android 13 Security With Ease*, Retrieved November 2025, 2025, URL: https://www.bitdefender.com/en-us/blog/labs/malicious-google-play-apps-bypassed-android-security.

[5] XDA Developers, *Custom ROMs Are No Longer Passing Play Integrity Validations*, Retrieved November 2025, 2025, URL: https://xdaforums.com/t/custom-roms-are-no-longer-passing-play-integrity-validations.4738655/.

[6] OWASP Foundation, *OWASP Mobile Top 10 Vulnerabilities 2024*, Retrieved November 2025, 2024, URL: https://strobes.co/blog/owasp-mobile-top-10-vulnerabilities-2024-updated/.

[7] GBHackers, *Google Wear OS Flaw Lets Any App Send Texts on Behalf of Users: CVE-2025-12080*, Retrieved November 2025, 2025, URL: https://gbhackers.com/google-wear-os-flaw/.

[8] GitHub Security Blog, *Bypassing MTE with CVE-2025-0072*, Retrieved November 2025, 2025, URL: https://github.blog/security/vulnerability-research/bypassing-mte-with-cve-2025-0072/.

[9] Google, *Android Security Bulletin - August 2015*, CVE-2015-1538, 2015, URL: https://source.android.com/security/bulletin/2015-08-01.

[10] Guardsquare, *Insecure TLS Certificate Checking in Android Apps*, Retrieved November 2025, 2024, URL: https://www.guardsquare.com/blog/insecure-tls-certificate-checking-in-android-apps.

[11] Qualcomm Technologies Inc., *Qualcomm Product Security Bulletin - June 2025*, CVE-2025-21479, CVE-2025-27038, 2025, URL: https://docs.qualcomm.com/product/publicresources/securitybulletin/june-2025-bulletin.html.

[12] Lookout Threat Intelligence, *CVE-2025-38352 and CVE-2025-48543: Two New Privilege Escalation Vulnerabilities Affecting Android OS*, Retrieved November 2025, 2025, URL: https://www.lookout.com/threat-intelligence/article/cve-2025-38352-cve-2025-48543.

[13] Kaspersky, *Attacks on Smartphones Increased in the First Half of 2025*, Retrieved November 2025, 2025, URL: https://www.kaspersky.com/about/press-releases/kaspersky-report-attacks-on-smartphones-increased-in-the-first-half-of-2025.

[14] SC Media, *Active Exploitation of Android Vulnerabilities: CVE-2024-43093 Framework Privilege Escalation*, Retrieved November 2025, 2024, URL: https://www.scworld.com/brief/active-exploitation-of-android-vulnerabilities-ongoing.

[15] National Vulnerability Database, *CVE-2024-29745 Detail: Information Disclosure via Uninitialized Data*, Retrieved November 2025, 2024, URL: https://nvd.nist.gov/vuln/detail/cve-2024-29745.

[16] National Vulnerability Database, *CVE-2024-43081 Detail: Logic Error in Android InstallPackageHelper*, Retrieved November 2025, 2024, URL: https://nvd.nist.gov/vuln/detail/CVE-2024-43081.

[17] National Vulnerability Database, *CVE-2024-49748 Detail*, Retrieved November 2025, 2025, URL: https://nvd.nist.gov/vuln/detail/CVE-2024-49748.

[18] Promon, *StrandHogg: Task Affinity Vulnerability in Android*, Retrieved November 2025, 2019, URL: https://promon.io/resources/downloads/strandhogg-2-0-new-serious-android-vulnerability.

[19] OffSeq Radar, *CVE-2025-36907: Elevation of Privilege in Google Android*, Retrieved November 2025, 2025, URL: https://radar.offseq.com/threat/cve-2025-36907.

[20] StreyPaws, *Race Against Time in the Kernel's Clockwork: CVE-2025-38352 TOCTOU Analysis*, Retrieved November 2025, 2025, URL: https://streypaws.github.io/posts/Race-Against-Time-in-the-Kernel-Clockwork/.

[21] Zscaler ThreatLabz, *67% Jump in Android Malware - 2025 Mobile, IoT, and OT Threat Report*, Retrieved November 2025, 2025, URL: https://ir.zscaler.com/news-releases/news-release-details/zscaler-threatlabz-reveals-67-jump-android-malware-and-40-iot.

A Detalii Tehnice Suplimentare

A.1 Exemplu de Politică SELinux

B Glosar de Termeni