Arhitectura de Securitate a Platformei Android: Analiza Mecanismelor de Protecție

Budiul Cristian-Carol
Titianu Maria

27 octombrie 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

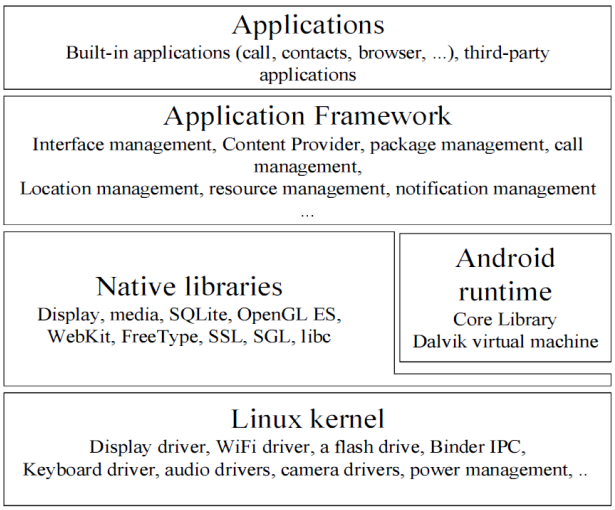

Figura 1: Arhitectura platformei Android

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 Amenințări Specifice

4.1 Vulnerabilități ale Sistemului și Aplicațiilor

4.1.1 Exploatarea Privilegiilor

4.1.2 Vulnerabilități Kernel/Driver

4.2 Malware și Aplicații Malițioase

4.2.1 DroidDream (2011)

4.2.2 GingerMaster (2011)

4.2.3 Masque Attack (2014)

4.3 Riscuri legate de Root/Jailbreak și ROM-uri Terțe

4.4 Pierderea Confidențialității Datelor Personale

4.4.1 Tracking

4.4.2 Acces Neautorizat

5 Studii de Caz – Vulnerabilități Reprezentative

5.1 Stagefright (2015) - Atac Media

5.2 QuadRooter (2016) - Vulnerabilități în Drivers

6 Măsuri de Protecție și Soluții Propuse

6.1 La Nivel de Utilizator: Actualizări, Permisiuni, Igienă Digitală

6.1.1 Actualizări

6.1.2 Permisiuni

6.1.3 Igienă Digitală

6.2 La Nivel de Dezvoltator: Principiul Minimului Privilegiu, Criptare, Protecție IPC

6.2.1 Principiul Minimului Privilegiu

6.2.2 Criptarea Datelor

6.2.3 Protecție IPC

6.3 La Nivel de Sistem: Întărirea Kernel (SELinux), Verificarea Aplicațiilor (Google Play Protect)

6.3.1 Întărirea Kernel

6.3.2 Google Play Protect

7 Concluzii