HAAGA-HELIA
ammattikorkeakoulu

# Android - Arkkitehtuuri ja pääkäyttäjäoikeuksien hallinnointi

Lauri Vihonen

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2015

HAAGA-HELIA
ammattikorkeakoulu

**Tiivistelmä**

Tietojenkäsittelyn koulutusohjelma 29.5.2015

| **Tekijä tai tekijät**<br>Lauri Vihonen | **Ryhmätunnus tai aloitusvuosi**<br>TN7PB |
|---|---|
| **Raportin nimi**<br>Android - Arkkitehtuuri ja pääkäyttäjäoikeuksien hallinnointi | **Sivu- ja liitesivumäärä**<br>**40** |
| **Opettajat tai ohjaajat**<br>Jukka Juslin | |


Useat isot teknologia-toimijat ovat lähteneet mukaan älylaite-markkinoille viime vuosina, kuten esimerkiksi Microsoft ja Apple. Kaikki ovat saavuttaneet menestystä, mutta yksi on ylitse muiden. Vuoden 2014 kolmannella vuosikvartaalilla Android-pohjaisia laitteita oli myyty maailmanlaajuisesti yli 268 miljoonaa kappaletta, joka oli roimasti enemmän kuin heidän suurimmalla kilpailijallaan iPhonella.

Android-käyttöjärjestelmän avoimuus antaa sen käyttäjille lukuisia mahdollisuuksia muokata laitteensa toiminnallisuutta sekä ulkoasua. Käyttöjärjestelmän muokkaaminen on mahdollista myös ilman tutkimukseni aiheena olevaa prosessia, mutta sen tehtyään vaihtoehdot ovat käytännössä rajoittamattomat. Tämä pääkäyttäjäoikeuksien- hankkimiseen vaadittava prosessi, ”roottaus”, on valikoitunut tutkimukseni aiheeksi.

Projektin aikana tutkitaan lukuisista eri mobiilikäyttöjärjestelmistä ainoastaan Androidia, eikä sen aikana perehdytä sen kilpailijoihin. Tiedonperustana käytetään pääasiallisesti Android Hackers Handbook – eKirjaa. Projekti käynnistettiin 1.4.2015 ja se päätetään 31.5.2015



**Avainsanat**

Android, Roottaus, älypuhelin, Open Source, Arkkitehtuuri, SuperUser

HAAGA-HELIA
ammattikorkeakoulu

**Abstract**

Business Information Technology 29.5.2015

| **Authors** | **Group or year of entry** |
|---|---|
| Lauri Vihonen | TN7PB |
| **The title of thesis** | **Number of pages and appendices** |
| Android - Architecture and management of SuperUser-rights | **40** |
| **Supervisor(s)** | |
| Jukka Juslin | |


Several large technology organizations have joined the smart device markets such as Microsoft, as well as Apple. All have achieved success, but one stands above the rest. In the 2014 third quarter Android-based devices were sold worldwide in more than 268 million pieces, which had tumbled more than their biggest competitor iPhone or iOS.

The openness of the Android operating system grants it´s users a number of options for customizing it´s functionality and design. Customization possibilities are significant even without rooting-process, but afte concluding it the options are virtually unlimited. This is the reason why this project was started.

This research concerns only Android and it´s competitors are not the subject. The main source of information is an eBook called Android Hacker´s Handbook. This Project was initialized on 1.4.2015 and concludes on 31.5.2015.



**Key words**

Android, Rooting, Smartphones, Open Source, Architecture, SuperUser

# Sisällys

# 1 Johdanto

Jo syntymästään asti älypuhelimet ovat mullistaneet maailmaa. Monet olivat skeptisiä niiden luomisprosessin alkuvaiheissa eivätkä uskoneet niiden mahdollisuuksiin. Mutta toisin kävi. Ne ovat ottaneet oman paikkansa kuluttajien päivittäisessä elämässä. Ihmiset käyttävät niitä kuin tietokoneitaan ennen ja niitähän ne ovatkin. Taskuun mahtuvia tietokoneita, joilla voi soittaa. Näitä maagisia laitteita käytetään työtehtävien suorittamiseen, sosiaaliseen kanssakäymiseen, videopelaamiseen, valokuvaamiseen, ja musiikin kuunteluun. Lähes ainoana erona on se, että tämän tapainen laite löytyy melkein jokaisen ihmisen taskusta tänä päivänä. Tämä mahdollistaa tietokoneen jo olemassa olevien ominaisuuksien erilaisen hyödyntämisen, sekä tarjoaa alustan täysin uusien luomiseen.

Useat isot teknologia-toimijat ovat lähteneet mukaan älylaite-markkinoille kuten Microsoft sekä Apple. Kaikki ovat saavuttaneet menestystä, mutta yksi on ylitse muiden. Vuoden 2014 kolmannella vuosikvartaalilla Android-pohjaisia laitteita oli myyty maailmanlaajuisesti yli 268 miljoonaa kappaletta, joka oli roimasti enemmän kuin heidän suurimmalla kilpailijalla iPhonella. Saman vuoden toisella kvartaalilla Android-tuotteilla oli hallussaan jopa 84,7-prosentin markkinaosuus maailmanlaajuisessa älypuhelin-teollisuudessa. Monet niiden käyttäjät saattavat kuitenkin miettiä miten tämä taskussa oleva laitteeni toimii ja mitä niillä voisi tehdä. Siinä on pääsyy miksi tämä projekti käynnistettiin.

Opinnäytetyössä perehdytään kaikista suosituimpaan älylaite-käyttöjärjestelmään, Androidiin. Siinä tutkitaan sen arkkitehtuuria, eri komponentteja sekä niiden toiminnallisuutta joista koostuu kattava dokumentaatio. Projektin aikana toteutetaan lisäksi roottaus-toimenpide Samsung Galaxy S4 LTE-mallille joka dokumentoidaan. Androidin toimintaperiaate ei ole tuttu tutkijalle, joten siihen perehtyminen ja sen analysointi toimivat oppimistavoitteena. Projektissa ei käsitellä muita älypuhelin-käyttöjärjestelmiä kuten iOS:tä tai Windows Phonea, vaan vain ja ainoastaan Androidia.

# 2 Android - Ekosysteemi

Sanaa Android voidaan käyttää monissa asiayhteyksissä. Vaikka sanalla voidaan viitata humanoidi robottiin, viimeisen vuosikymmenen aikana sana Android on saanut paljon muitakin merkityksiä. Sillä voidaan tarkoittaa organisaatiota, käyttöjärjestelmää, vapaan lähdekoodin projektia tai kehitysyhteisöä. Jotkut kutsuvat mobiililaitteitaan Androideiksi. Tämä on tavallaan ymmärrettävää sillä kokonainen ekosysteemi perustuu tähän suosittuun käyttöjärjestelmään.
(Drake.J, Ym…2014)

## 2.1 Historia

Vuoden 2003 lokakuussa neljä teknologian asiantuntijaa löysivät toisensa ja päättivät yhdistää voimansa perustaakseen organisaation joka tunnetaan nimellä Android Inc. of Palo Alto,CA. Tämän nelikon muodostivat herrat nimeltään Andy Rubin, Rich Miner, Nick Sears sekä Chris White. Perustamishetkellä heillä oli vain heidän ammattitaitonsa sekä visio matkapuhelimesta joka mullistaisi silloisen käsityksen puhelimista. Heidän visiostaan kehittyi myöhemmin tuote joka sai uskomattoman jalansijan mobiilin tietotekniikan alalla. Nelikon primäärisenä tavoitteena oli kehittää käyttöjärjestelmä mobiililaitteille joka samanaikaisesti olisi tietoinen käyttäjänsä reaaliaikaisesta sijainnista ja tiedostaisi heidän henkilökohtaiset mieltymyksensä.

Käyttöjärjestelmää suunniteltiin alun perin implementoitavaksi digitaalisiin kameroihin, pääajatuksena luoda ns. äly-kameroita. Kameroilla olisi pääsy tietokonemaisiin palveluihin, mutta toisin kuitenkin kävi. Organisaatio tunnisti että sellaiselle laitteelle ei olisi kysyntää kuluttajapuolelta, joten he päättivät kohdistaa kehityspanoksensa matkapuhelimiin.

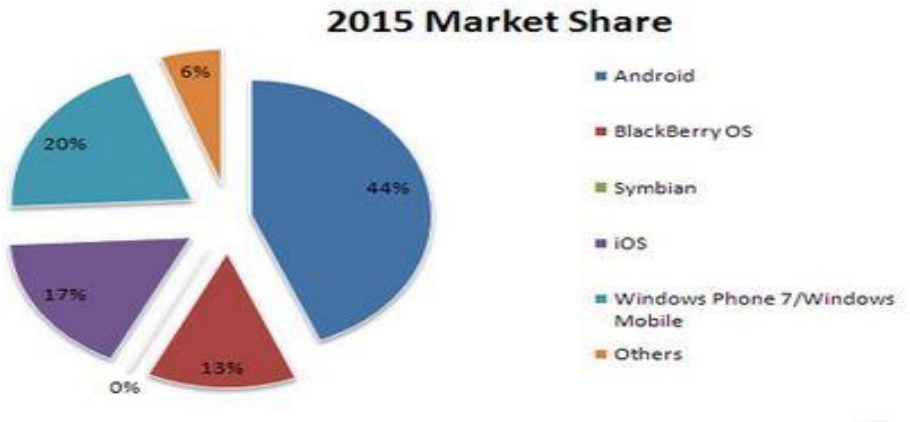
Elokuussa 2005 Androidista tuli osa teknologiajätti Googlea, joka oli kiinnostunut uudesta teknologiasta sen omaperäisyyden ja viehätyksen ansiosta. Tämän kasvavan startup-yrityksen lopullista myyntihintaa ei ole julkisesti kerrottu mutta sen on arvioitu olevan noin 50 miljoonan dollarin suuruinen. Kourallinen ihmisiä kyseenalaisti kauppaa silloin, mutta suurin osa asiantuntijoista kuitenkin tiedostivat kaupasta seuraavan Googlen hallinta mobiili käyttöjärjestelmämarkkinoilla.

Suurin virstanpylväs Androidin kehityskaaressa tapahtui Lokakuun 5. päivä vuonna 2007. Tuona kyseisenä päivänä Google julkisti Open Handset Alliance (OHA) nimisen teknologiakonsortion, jonka päämääränä olisi tuottaa kuluttajille avoimia standardeja mobiililaitteille. Lopputuloksena syntyi 34:n eri yrityksen muodostama liitto. Liitto sisälsi eri teknologian alojen toimijoita kuten T-mobil, Motorola, HTC, Texas Instruments sekä Qualcomm. Tästä seurasi vääjäämättä se että kehitteillä olevalla käyttöjärjestelmällä olisi yksi merkittävä etu kilpailijaansa iPhoneen verrattuna. Käyttöjärjestelmää ei rajoitettu vain yhdelle laitevalmistajalle vaan muutkin saivat mahdollisuuden valmistaa niitä. Tämä oli Googlen kannalta hyvä asia, sillä ensimmäinen heidän valmistama laite ilmestyi vasta vuosien päästä. Kaiken kaikkiaan ensimmäisen Android-pohjaisen älypuhelimen valmisti HTC vuoden 2008 Lokakuussa. Laite sai nimekseen HTC Dream.

Kuten yllä mainittiin, vaikka Android julkaistiin kaupallisesti älypuhelinmarkkinoille vuonna 2008, Googlen ensimmäinen oma laite julkaistiin kuluttajille vasta vuona 2010. Ennen Google Nexuksen julkaisua muut laitevalmistajat pitivät yllä Androidin kiinnostusta ja markkina-arvoa.

Tärkein toimija Android-laitemarkkinoilla on kuitenkin korealainen teknologiajätti Samsung, joka on saavuttanut suuren jalansijansa julkaisemalla useampia laitemalleja kuin kilpailijansa Sony, HTC tai LG. Samsungin Galaxy-mallisto erityisesti tunnetaan organisaationsa lippulaivana. Tämä voidaan todeta suoraan myyntiluvuista, joista esimerkkinä vuoden 2014 ensimmäisen vuosikvartaalin 85 miljoonaa myytyä älypuhelinta. Nämä luvut ylittivät Applen, LG:n, Huawein sekä Lenovon yhdistetyt saman kvartaalin myyntiluvut.

Vuoden 2014 kolmannella vuosikvartaalilla Android-pohjaisia laitteita oli myyty maailmanlaajuisesti yli 268 miljoonaa kappaletta, joka oli roimasti enemmän kuin esimerkiksi iPhonella. Saman vuoden toisella kvartaalilla heidän tuotteella oli hallussaan jopa 84,7-prosentin markkinaosuus maailmanlaajuisessa älypuhelin-teollisuudessa. Tilanne on kuitenkin ajan kuluessa tasoittunut, sillä heidän ja muiden toimijoiden välinen kuilu on kaventunut huomattavasti. Tämä voidaan todeta vuoden 2015 markkinaosuuksista, jotka näkyvät kaaviossa numero 1.




Kaavio 1 - Markkinaosuudet 2015 (fiercemobileit.com 2015)

(Drake.J, Ym…2014; Brachmann.S 2014)

## 2.2 Versiot

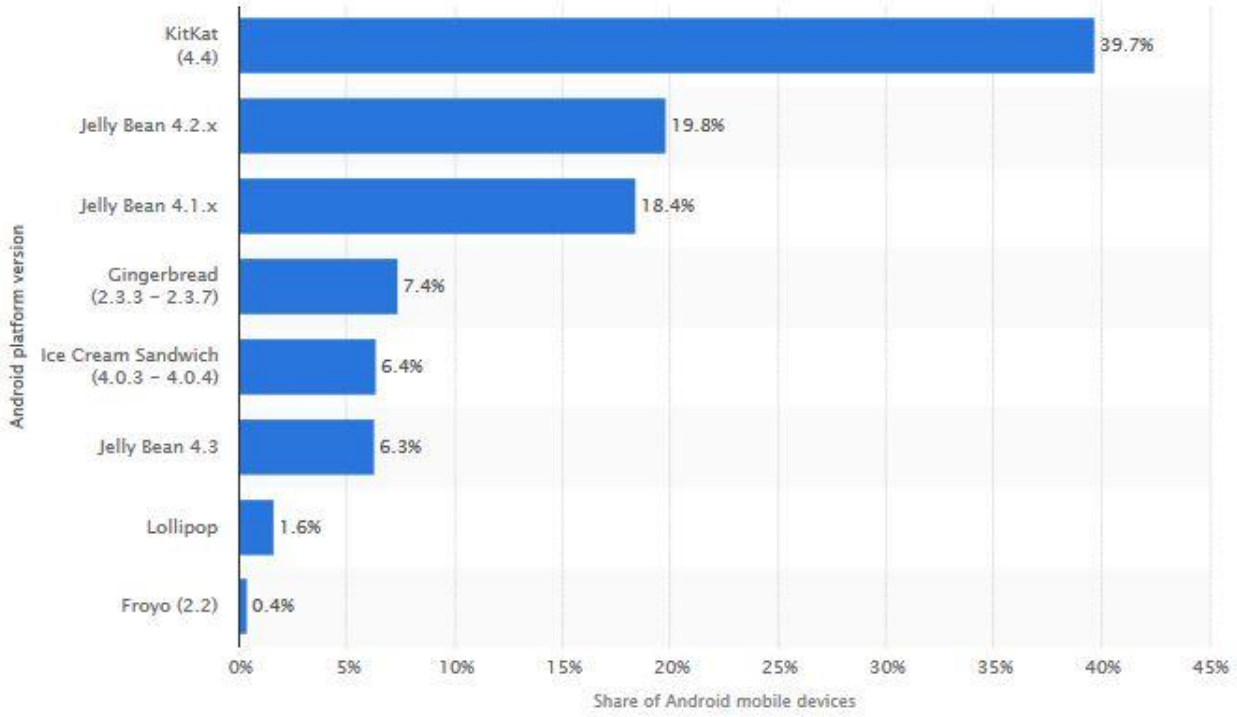
Androidista oli kaksi julkaisua ennen ensimmäistä kaupallista versiota, Alpha ja Beta. Niiden nimiksi varmistui kuitenkin myöhemmin Astro ja Bender, suosittujen robottien mukaan. Nämä olivat kuitenkin vain Googlen ja OHA-liiton jäsenien käytössä. Alla olevassa taulukossa 1 on eriteltynä kaikki tähän asti julkaistut versiot.

Taulukko 1 - Android versiot (Drake.J, Ym 2014)

| **Versio** | **Koodinimi** | **Julkaisupäivä (1.version mukaan)** |
|---|---|---|
| **0.x** | Alpha | Marraskuu 2007 |
| **0.x** | Beta | 5.11.2007 |
| **1.0** | Astro | 23.9.2008 |
| **1.1** | Bender | 9.2.2009 |
| **1.5** | Cupcake | 27.4.2009 |
| **1.6** | Donut | 15.9.2009 |
| **2.0 - 2.1** | Eclair | 26.10.2009 |
| **2.2 – 2.2.2** | Froyo | 20.5.2010 |
| **2.3 – 2.3.7** | Gingerbread | 6.12.2010 |
| **3.0 – 3.2.6** | HoneyComb | 22.2.2011 |
| **4.0 – 4.0.4** | Icecream Sandwich | 18.10.2011 |
| **4.1 – 4.2.2** | JellyBean | 9.7.2012 |
| **4.4 – 4.4.4** | KitKat | 31.10.2013 |

Ensimmäinen kaupallinen versio Android 1.0 julkaistiin vuoden 2008 syyskuun 23. päivä ja seuraava 1.1-versio helmikuun 9. päivä vuonna 2009. Nämä kaksi ensimmäistä olivat ainoat ns. nimettömät julkaisut. Versiosta 1.5 eteenpäin isojen julkaisujen nimet ovat muodostuneet erinäisten herkkujen mukaan aakkosjärjestyksessä. Taulukossa 2 kuvataan Androidin omaa sisäistä eri versioiden käyttöasteiden välistä ”kilpailua”.

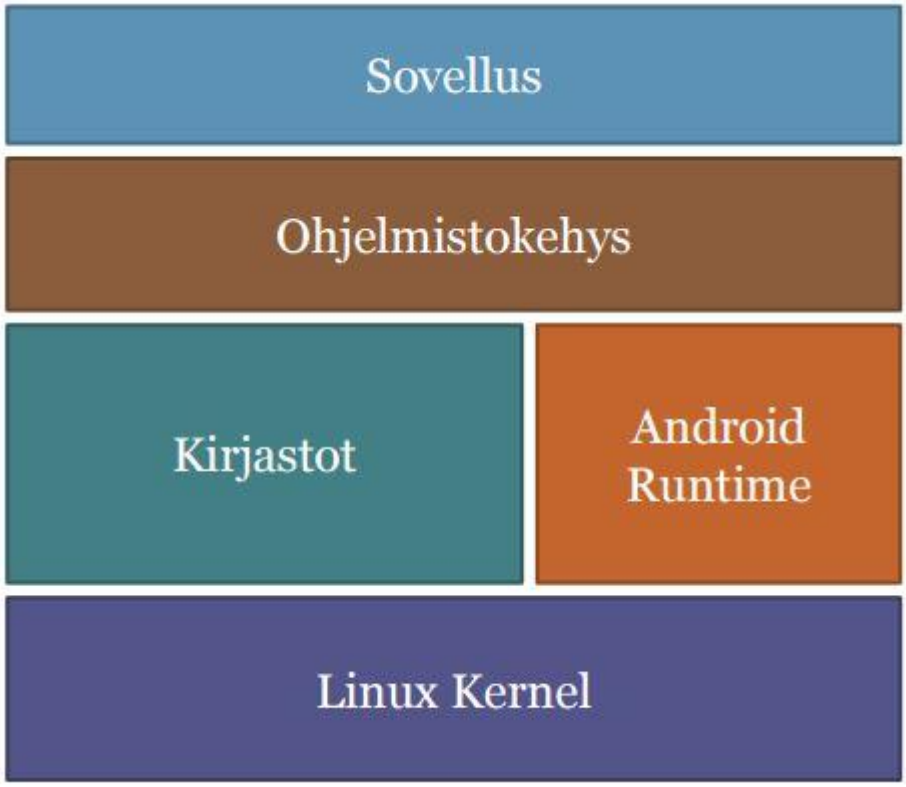
Taulukko 2 - Versioiden käyttöasteet 2.2.2015 (statista.com)




(Drake.J, Ym…2014)

# 3 Android - Arkkitehtuuri

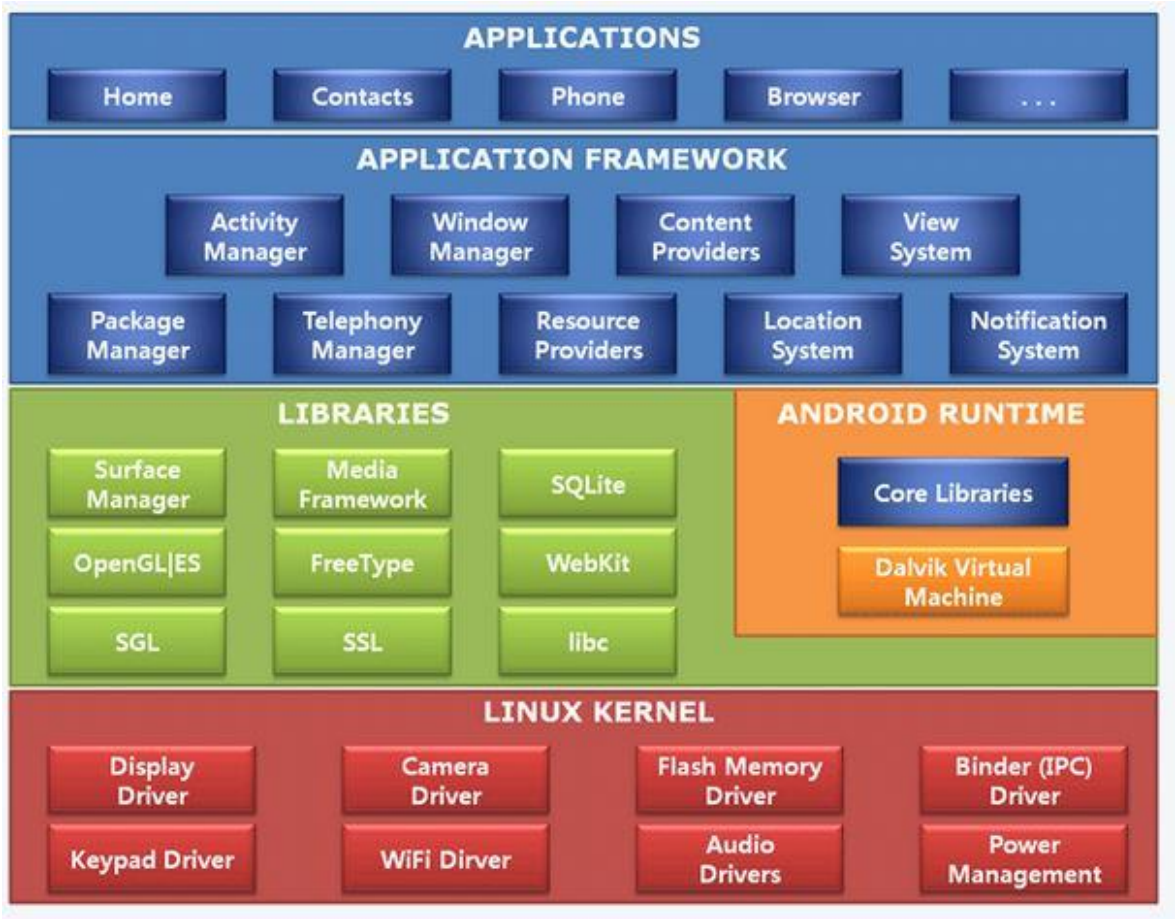
Androidin yleisarkkitehtuuri on useasti luonnehdittu ”Javaksi Linuxin päällä” (Java on Linux). Tämän tapainen luonnehdinta saattaa kuitenkin olla hieman harhaanjohtava eikä välttämättä ole täysin oikeudenmukainen Androidin arkkitehtuurin monimutkaisuutta kohtaan. Se voidaan käytännössä jakaa viiteen osa-alueeseen tai toisin sanoen kehykseen, jotka ovat sovellus(Applications), ohjelmisto(Application Framework), kirjasto(Libraries), Android Runtime sekä Linux Kernel. Kuva 1 selventää arkkitehtuurin hierarkian selvemmin.




Kuva 1 - Android arkkitehtuuri 1 (Android Ohjelmointi PDF)

Sovelluskehys antaa kehittäjille mahdollisuuden laajentaa ja parantaa laitteen toiminnallisuutta ilman alempien tasojen muokkausta. Ohjelmistokehys puolestaan tarjoaa kehittäjilleen paremman ohjelmointirajapinnan(API). Tämä toimii ns. ”liimana” ohjelmistokehyksen ja Android Runtime(ts.Dalvik) – kehyksen välillä. Sen ansiosta kehittäjät pystyvät suorittamaan yleisiä toimenpiteitä, kuten käyttöliittymän(UI) elementtien hallinnointia, jaettujen datavarastojen hallinnointia. He pystyvät myös kommunikoimaan eri sovelluskomponenttien välillä.

Sekä Sovellus ja Ohjelmisto-kehykset on toteutettu Java-ohjelmointikielellä ja niitä suoritetaan Dalvik-virtuaalikoneessa. Dalvik on rekisteripohjainen virtuaalikone joka tulkitsee DEX(Dalvik Executable)-bittikoodia. Toiminnallisuuden varmistamiseksi se käyttää Kirjasto-kehykseltä löytyviä natiiveja tuki-koodeja. Kuvassa 2 käsitellään Androidin arkkitehtuuria hieman syvällisemmin.




Kuva 2 - Android arkkitehtuuri 2
(Drake.J, Ym…2014)

### 3.1 Sovelluskehys

Tämäkin kehys voidaan jakaa vielä kahteen eri ala osa-alueeseen: esiasennetut sovellukset sekä käyttäjän asentamat sovellukset. Esiasennetut (Bloatware) pitävät yleisesti ottaen Googlen erinäisiä sovelluksia ja laitevalmistajasta riippuen heidän omia sovelluksiaan. Muita esiasennettuja sovelluksia ovat esimerkiksi kalenteri, sähköposti, web-selain sekä yhteystietojen hallinnointisovellus. Nämä paketit ovat esiasennettuna /system/app – polulla, vaikka ne saattavatkin olla piilotettuina jos laitevalmistaja ei ole halunnut päästää kuluttajaa käsiksi niihin. Tässä kohtaa tutkimuksen aihe tulee ajankohtaiseksi. Käyttäjän asentamat sovellukset ovat kirjaimellisesti ohjelmistoja jotka laitteen käyttäjä on asentanut itse sovelluskaupasta, lataamalla APK – paketin verkosta tai manuaalisesti tiettyjä komentoja hyödyntäen. Nämä sovellukset asentuvat /data/app – polulle.
(Drake.J, Ym…2014)

Vaikka Sovelluskehys koostuu useista eri palasista, Android-versiosta riippuamatta, se pitää sisällään muutaman pääsovelluksen. Näitä ovat AndroidManifest, Intentit, BroadcasReceiverit, Servicet ja Content Provider. Neljä viimeiseksi mainittua ovat IPC-päätepisteitä joilla on turvallisuuden kannalta oleellisia ominaisuuksia.

AndroidManifest on xml-tiedosto joka pitää sisällään kaikki Androidissa olevat APK-paketit sekä niiden tiedot. AndroidManifest-tiedosto generoituu yleensä automaattisesti kehitysympäristön(esim.Eclipse tai Android-studio) toimesta. Tietoja joita AndroidManifest sisältää ovat:

- Yksilöivä paketin nimi ja versioinformaatio.
- Activitien, Servicien, BroadcastReceiverin sekä Instrumentaation määritelmät.
- Lupamääritelmät.
- Ulkoisten kirjastojen tietoja joita sovellus käyttää tai jotka tulevat sen mukana.
- Tukidirektiivejä esim. UID-informaatio.

(Drake.J, Ym…2014)

Intentit ovat oleellinen osa sovelluksen sisäistä kommunikaatiota. Nämä ovat viesti objekteja jotka pitävä sisällään informaatiota suoritettavasta operaatiosta, vaihtoehtoisesta kohde komponentista jota käsitellään sekä muuta tukidataa joka voi olla merkityksellistä vastaanottajalle. Intentit ovat oleellisessa roolissa useissa yleisissä järjestelmätapahtumissa, kuten esimerkiksi sovelluksien asennuksissa tai niiden poistossa, viestintäsovelluksen SMS-viestin havaitsemissa ja linkin avautumisessa selaimeen sähköpostista.
(Drake.J, Ym…2014)

Broadcast Receiver on toisenlainen IPC-päätepiste. Näitä käytetään kun sovellukset haluavat vastaanottaa Intentin joka vastaa tietyntyyppistä kriteeriä. Esimerkiksi sovellus, joka haluaa vastaanottaa Intentin joka on yhdistetty SMS-viestin vastaanottamiseen. Tässä tapauksessa sovellus rekisteröisi receiverin sen manifestiin joka on soveltuva Intent-suodattimen mukaan. Alla olevassa kuvassa 3 tapahtuman koodi:

```
<receiver android:name=".MySMSReceiver">
  <intent-filter android:priority="999">
    <action android:name="android.provider.Telephony.SMS_RECEIVED" />
  </intent-filter>
</receiver>
```

Kuva 3 – Broadcast Receiver

Broadcast Receiver voidaan myöskin rekisteröidä ohjelmoinnin avulla Dalvikissa käyttämällä registerReceiver-metodia. Tätä metodia voidaan hyödyntää myös lupavaatimuksien määrittelyssä. Lupavaatimuksien määrittelyn avulla voidaan rajoittaa mitkä sovellukset voivat lähettää Intenttejä Broadcast Receiver-päätepisteeseen.

Servicet tai palvelut ovat applikaation osia, joilla ei ole käyttöliittymää, ja jotka pyörivät taustalla ilman että käyttäjä suoraan käyttäisi kyseisiä palveluja. Näitä ovat esimerkiksi SmsReceiverService sekä BluetoothOppService-palvelut. Nämä molemmat voivat hyödyntää IPC:n tarjoamia mahdollisuuksia kuten Intentien vastaanottoa ja lähettämistä. Jotta Service toimisi, se on pakko julistaa(declare) sovelluksen manifestiin. Alla olevassa kuvassa 4 on esimerkki yksinkertaisesta palvelun määritelmästä joka hyödyntää Intent-suodatinta.

```
        <service
android:name="com.yougetitback.androidapplication.FindLocationService">
            <intent-filter>
                <action
android:name="com.yougetitback.androidapplication.FindLocationService" />
            </intent-filter>
        </service>
```

Kuva 4 – Service definition (Drake.J, Ym…2014)

Content Providerit,ts. sisällön tarjoajat, hallinnoivat pääsyä data-rakennelmiin. Ne kapseloivat datan sekä tarjoavat mekanismit joiden avulla datan turvallisuus määritellään. Content provider on rajapinta joka yhdistää datan toisesta prosessista koodiin mitä ajetaan toisessa prosessissa.

(Drake.J, Ym…2014)

### 3.2 Ohjelmistokehys (Android Framework)

”Sovelluskehyksen(Android Framework) ja Dalvikin(Android Runtime) välinen silta” hallinnoi paketteja ja niiden käyttämiä eri luokkia(Class). Näiden ansiosta kehittäjät kykenevät suorittamaan yleisiä toimenpiteitä kuten modifioimaan käyttöliittymiä. Kehys pitää sisällään kaiken koodin joka ei ole sovelluskohtaista ja jota ajetaan DalvikinVM:ssä.

Android Framework käsittää myöskin palveluja jotka hallinnoivat ja helpottavat luokkien toiminnallisuutta. Nämä ns. managerit käynnistyvät järjestelmän käynnistysvaiheessa ja ne ovat esitelty taulukossa 3.

Taulukko 3 – Kehyksen managerit

| Kehyksen Palvelu (Framework Service) | Kuvaus |
|---|---|
| **Activity Manager** | Hallinnoi Intenttien päämääriä sekä sovellusten aktiivisuutta |
| **View System** | Hallinnoi aktiviteettien näkymiä |
| **Package Manager** | Hallinnoi jonossa olevien asennettavien pakettien informaatiota ja tehtäviä |
| **Telephony Manager** | Hallinnoi puhelin, radio ja verkko tehtäviä ja niiden informaatiota |
| **Resurssi Manageri (Resource Manager)** | Hallinnoi pääsyä koodittomiin sovellusresursseihin kuten grafiikkaan ja käyttöliittymään |
| **Sijainti Manageri (Location Manager)** | Tarjoaa rajapinnan sijaintipalveluihin(GPS,WiFi) |
| **Ilmoitus Manageri (Notification Manager)** | Hallinnoi erinäisiä tapahtumailmoituksia kuten äänten toisto, tärinä, välkkyviä LEDejä sekä näyttää ikoneita tilapalkissa |

(Drake.J, Ym…2014)

### 3.3 Android Runtime (The Dalvik Virtual Machine)

Rekisteripohjaisen Dalvikin sanotaan pohjautuvan Java-ohjelmointikieleen, mutta se ei ole puhdasta Javaa. Tämä voidaan päätellä siitä että Google ei käytä Javan logoja eikä sen sovellus-mallin ja JSR:n(Java Specification Requirements) välillä ole minkäänlaista yhteyttä. Android kehittäjän mielestä DalvikVM voi näyttää ja tuntua siltä mutta sitä se ei ole. Alla kuvattuna Androidin kehitysprosessi:

1. Kehittäjä kirjoittaa koodin joka muistuttaa Javaa
2. Lähdekoodi käännetään .class-tiedostoiksi (muistuttaa Javaa)
3. Tuloksena syntyvät class-tiedostot käännetään Dalvik-koodiksi
4. Kaikki tiedostot yhdistetään yhdeksi DEX(Dalvik executable)-tiedostoksi
5. Bitti-koodi ladataan ja tulkataan DalvikVM:ssä

Koska Dalvik on rekisteripohjainen virtuaalikone, se pitää sisällään suurin piirtein 64 000 erinäistä virtuaalista rekisteriä. Näistä kuitenkin käytetään tyypillisesti vain 16 ensimmäistä tai harvemmin 256. Nämä rekisterit ovat osoitettuja muistipaikkoja VM:n muistissa, jotka

simuloivat rekisterin toiminnallisuutta mikroprosessoreissa. DalvikVM käyttää näitä rekistereitä järjestelmän monitoroinnissa ja ajaa samanaikaisesti bitti-koodia, aivan kuten oikea mikroprosessori.

DalvikVM suunniteltiin sulautetulle järjestelmälle(embedded system) ja sen rajoituksille kuten alhaiselle muistin määrälle ja prosessorin nopeudelle. Näin ollen sen suunnittelu vaiheessa kiinnitettiin erityisesti huomiota nopeuteen ja tehokkuuteen. Jotta rajoituksista saataisiin kaikki hyöty irti, DEX-tiedostot optimoidaan ennen kuin ne otetaan käyttöön virtuaalikoneessa. Tämä tapahtuu vain kerran kun Android-sovellus käynnistetään ensimmäisen kerran. Tuloksena syntyy optimoitu DEX-tiedosto(ODEX). On huomioitavaa että että nämä tiedostot eivät ole siirrettävissä Dalvikin eri versioihin tai laitteisiin. Kuten JavaVM:ssä, DalvikVM liittyy alempitasoiseen natiivi-koodiin käyttäen Java Native Interfacea(JNI). Tämä toiminnallisuus mahdollistaa Dalvik-koodin kutsumisen natiivista ja päinvastoin.

Zygote on yksi ensimmäisistä käynnistyvistä prosesseista kun Android-laitetta käynnistetään. Se vastaa lisäpalveluiden sekä kirjastojen lataamisesta joita Ohjelmistokehys käyttää. Lisäksi se toimii latausohjelmana jokaiselle Dalvikin prosessille tekemällä kopion itsestään. Optimoinnin ansiosta Android Frameworkia tai sen riippuvuuksia ei tarvitse ladata uudelleen kun käynnistetään Dalvik-prosessia. Tästä johtuen ydinkirjastot ja luokat jaetaan kaikille eri DalvikVM instansseille.

Zygote vastaa system_server-prosessin käynnistyksestä joka käsittää kaikki ydin palvelut jotka toimivat korotetuilla oikeuksilla. System_server käynnistää myöskin kaikki managerit joitka käsiteltiin taulukossa 3. Tämä prosessi on niin tärkeä, että sen lopettamisesta vaikuttaisi seuraavan laitteen uudelleenkäynnistäminen. Todellisuudessa kuitenkin laitteen Dalvikin alajärjestelmä käynnistyy uudestaan. Zygoten ensimmäisen käynnistyksen jälkeen se tarjoaa muille Dalvik-prosesseille pääsyn kirjastoihin RPC:n ja IPC:n avulla. Tämä on se mekanismi joka oikeasti käynnistää sovellukset.
(Drake.J, Ym…2014)

### 3.4 User-Space Native Code (Kirjastot)

Iso osa Androidin käyttöjärjestelmästä koostuu käyttäjä-avaruuskoodista. Tämä kehys voidaan jakaa kahteen pääryhmään: kirjastoihin sekä ydinpalveluihin.
(Drake.J, Ym…2014)

### 3.4.1 Kirjastot

Monet alhaiset toiminnallisuudet nojaavat Android Frameworkin korkeampiin luokkiin jotka toteutetaan jaetuilla kirjastoilla ja joihin otetaan yhteyttä JNI:n (Java Native Interface) avulla. Monet näistä kirjastoista ovat samoja tunnettuja avoimen lähdekoodin projekteja joita hyödynnetään Unixmaisissa käyttöjärjestelmissä. Esimerkiksi SQLite tuottaa tietokannan toiminnallisuuden lokaalisti, WebKitin avulla saadaan sulautettu web-selain ja FreeType mahdollistaa bitmap ja vector renderöinnin.

Vendor-tyyppiset kirjastot tarjoavat hardware tuen uniikeille laitemalleille ja löytyvät polulta /vendor/lib tai laitteesta riippuen /system/vendor/lib. Esimerkkeinä näistä kirjastot jotka tuottavat alhaisen tuen graafisille laitteille, GPS-lähettimille tai radiolle. Kirjastot sisältävät myöskin näiden vastakohtia, epäVendoreita, jotka löytyvät polulta /system/lib. Tyypillisiä esimerkkejä ovat:

- Libexif: JPEG prosessointi.
- Libexpat: Expat XML parserointi.
- Libaudioalsa:ALSA ääni.
- Libbluetooth: BlueZ Linux Bluetooth.
- Libdbus: D-Bus IPC.

Android-järjestelmä pitää sisällään useita muitakin. Android 4.3-versio esimerkiksi käsittää yli 200 jaettua kirjastoa. Koska nämä kirjastot kehitetään natiivilla koodilla, ne ovat taipuvaisia muistin korruptoitumiseen. Turvallisuuden näkökulmasta tämä on mielenkiintoinen kehys.
(Drake.J, Ym…2014)

### 3.4.2 Ydinpalvelut

Init on ensimmäinen käyttäjä-avaruuspalvelu jonka kernel käynnistää. Tämä on täysin verrattavissa Linuxin vastaavaan palveluun. Sen vastuulla on monia palveluita, mutta laitteen toiminnallisuuden kannalta muutamat ovat ylitse muiden:

- Käynnistykseen vaadittavien palveluiden käynnistäminen.
- Käyttäjien ja ryhmien määrittely palvelulle käytettäväksi.
- Järjestelmänlaajuisten asetuksien ja määritelmien asetus joita hyödynnetään Property Servicen kautta.

Property Service, joka on palvelu Initin sisällä, vastaa muistikartoitetuista avainarvoisista konfiguraatioista. Monet käyttöjärjestelmän ja kehyksien komponentit ovat riippuvaisia

näistä. Esimerkiksi verkkorajapinnan konfiguraatiot, radiovaihtoehdot ja jopa turvallisuuteen liittyvät asetukset.

Radio Interface Layer (RIL) tekee älypuhelimesta puhelimen. Ilman tätä komponenttia laitteella ei voi soittaa, lähettää tai vastaanottaa SMS-viestejä. Laitteella ei voisi myöskään päästä internettiin ilman WiFi-yhteyttä.

Debuggerin avulla Android-laite on kykeneväinen suorittamaan vianselvitystä. Se kerää raportin jokaisesta järjestelmän kaatumisesta. Näin ollen järjestelmänkehittäjä saa arvokasta informaatiota ongelmien ratkaisemiseksi.

ADB(Android Debugging Bridge) koostuu adbd-demonista, adb-serveristä isäntä työasemalla sekä adb-komentokehotteesta clientillä. Palvelin hallinnoi clientin ja demonin välisiä yhteyksiä sekä sekä ajaa tehtäviä kuten shell, sovellusten debuggaus, ohjaa portteja ja socketteja, tiedostojen siirtoja sekä asentaa ja poistaa sovellus-paketteja.

Volume Daemonin(ts.vold) avulla Android-järjestelmään voidaan liittää(mount) ja poistaa(unmount) erinäisiä tiedostojärjestelmiä. Esimerkiksi kun laitteeseen asennetaan SD-kortti vold-prosessi tarkistaa sen tiedostojärjestelmän virheiden varalta että siltä voidaan ajaa tiedostoja, ja että se asentuu oikealle polulle. Kun kortti poistetaan laitteesta void-prosessi poistaa sen laitteen tietojärjestelmästä.
(Drake.J, Ym…2014)

### 3.5 Linux Kernel – The Android Fork

Androidin kehityksen alusta alkaen Google on käyttänyt sen pohjana Linux-kerneliä josta se muokkasi ”forkin” joka tunnetaan nykyään nimellä Android. Vaikka Linux-kernelin toiminta on dokumentoitu ja ymmärretty kattavasti, kaikki muokkaukset ja uudet lisäykset eivät olleet yhteensopivia uuden järjestelmän kanssa. Kaiken kaikkiaan nämä muutokset ilmenivät noin 250 patchin muodossa jotka paikkasivat ongelmia joita ilmeni tiedostojärjestelmässä, verkkoyhteyksissä sekä prosessien ja muistin hallinnassa. Erään kernel-insinöörin mukaan nämä paikat tai patchit kuvastivat rajoituksia jotka Android kehittäjät paikansivat Linuxin kernelistä.

Lopullinen Android ominaisuuksien yhdistäminen Linuxiin tapahtui vuoden 2012 maaliskuussa. Taulukossa 4 on eriteltynä tärkeimpiä modifikaatioita.

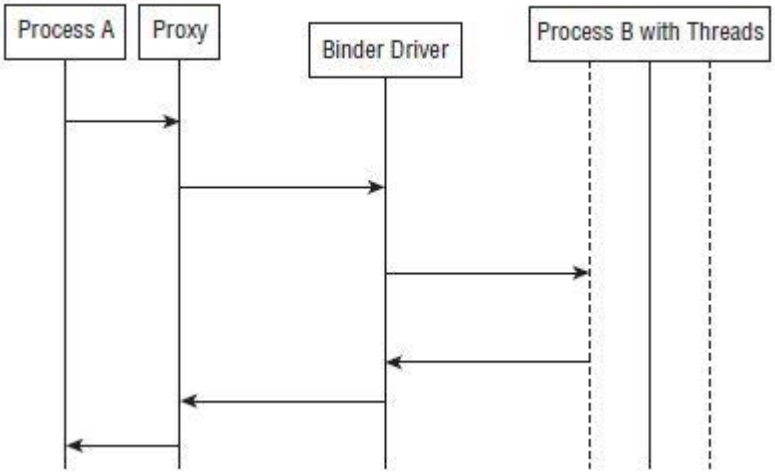
Taulukko 4 - Androidin muutokset Linuxin Kerneliin (Drake.J, Ym…2014)

| Kernelin muutos | Kuvaus |
|---|---|
| **Binder** | IPC-mekanismi lisäominaisuuksilla kuten soittajan turvallisuus validointi |
| **ashmem(Anonymous Shared Memory)** | Tiedostojen jaetun muistin jakaja. |
| **pmem(Process Memory Allocator)** | Käytetään isojen jaettujen muistien hallinnointiin |
| **logger** | Järjestelmän laajuinen loki-palvelu |
| **RAM_CONSOLE** | Säilyttää Kernelin loki viestit RAM:ssa Kernelin kaatumisen jälkeen |
| **“oom” modifications** | ”Out of memory”- tappaja. Tappaa sovellukset kun muisti loppuu |
| **wakelocks** | Virranhallinta ominaisuus joka estää laitetta ajautumasta virransäästötilaan ja auttaa sitä pysymään responsiivisenä |
| **Alarm Timers** | Kertoo kernelille million herätä |
| **Paranoid Networking** | Rajoittaa tiettyjä verkon operaatioita ja ominaisuuksia tiettyihin ryhmätunnisteisiin |
| **timed output / gpio** | Muuttaa ja palauttaa GPIO-rekisterejä tietyn ajan jälkeen |
| **yaffs2** | Tuki yaffs2 flash-tiedostojärjestelmälle |

### 3.5.1 Binder

Yksi tärkeimmistä näistä muutoksista on Binder. Binder on IPC-mekanismi joka perustuu OpenBinderin modifioituun versioon. Binder on suhteellisen pieni sovellus, joka käsittää n. 4000 riviä koodia, mutta on järjestelmän toiminnallisuuden kannalta äärimmäisen tärkeä.

Binderin ajuri hallinnoi sen koko arkkitehtuuria. Se toimii asiakas – palvelin mallin mukaan. Sen ansiosta prosessi kykenee kutsumaan metodeja ”etänä” synkronoidusti. Kuvassa 5 on kuvattu Binderin kommunikaatiota.




Kuva 5 – Binderin toiminta (Drake.J, Ym…2014)

Binder hyödyntää prosessin ID:tä(Process ID) sekä yksilöllistä ID:tä(Unique Identification Number) kutsunnan tunnistamiseen. Tämän ansiosta kutsuttuvalla on mahdollisuus kontrolloida pääsyä. Tähän käytetään tyypillisesti kahta metodia:

- Binder.getCallingUid.
- Binder.getCallingPid

ACCESS_SURFACE_FLINGER-lupa toimii hyvänä käytännön esimerkkinä tästä. Tämä lupa myönnetään tyypillisesti vain graafiselle järjestelmäkäyttäjälle (Graphics System user) joka myöntää pääsyn Binderin IPC-rajapinnalle. Alla olevassa kuvassa 6 on esimerkki metodien toiminnasta.

```
const int pid = ipc->getCallingPid();
const int uid = ipc->getCallingUid();
    if ((uid != AID_GRAPHICS) &&
            !PermissionCache::checkPermission(sReadFramebuffer,
                pid, uid)) {
        ALOGE("Permission Denial: "
                "can't read framebuffer pid=%d, uid=%d", pid, uid);
        return PERMISSION_DENIED;
}
```

Kuva 6 – Esimerkki Calling-metodista (Drake.J, Ym…2014)

### 3.5.2 Ashmem

Anonymous Shared Memory tai lyhennettynä ashmem, tarkoittaa periaatteessa tiedostopohjaista jaetun muistin rajapintaa. Sitä käytetään laajasti Androidin ydin komponenteissa kuten Surface Flingerissä, Audio Flingerissä, System Serverissä ja DalvikVM:ssä. Se on suunniteltu pienentämään ja kasvattamaan välimuisteja aina kun järjestelmämuisti on vähissä, joten se sopii loistavasti pienimuistisiin ympäristöihin.
(Drake.J, Ym…2014)

### 3.5.3 Pmem

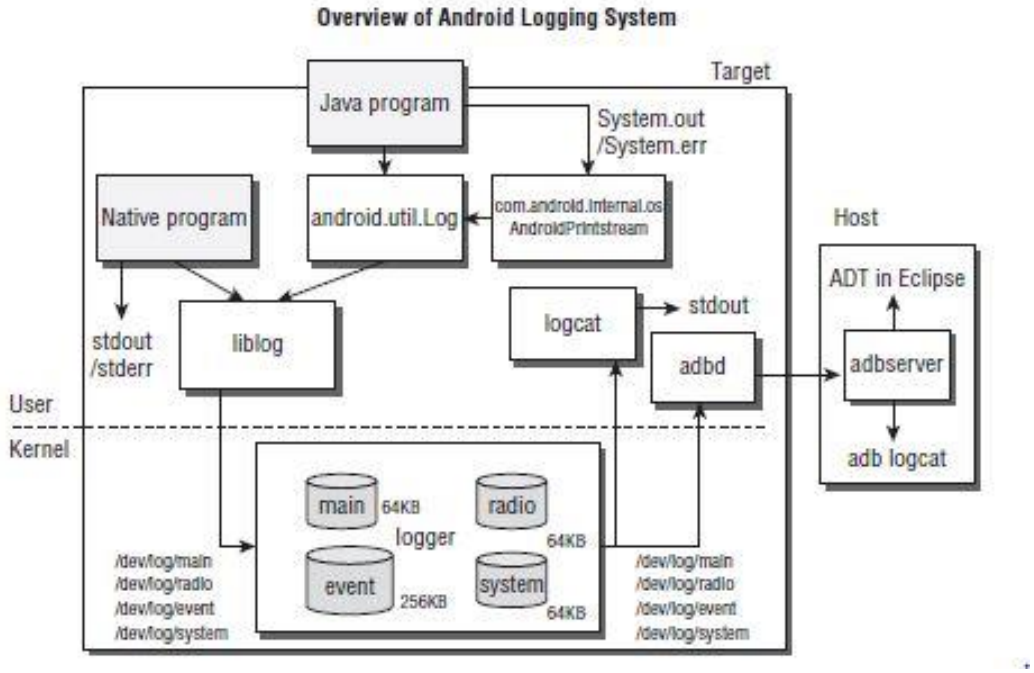
Pmem on erityisesti Androidille kehitetty ajuri, jonka vastuulla on suurten muistien hallinnointi. Implementoinnista riippuen nämä koot vaihtelevat tyypillisesti 1 – 16MB välillä, mutta voivat olla suurempiakin.
(Drake.J, Ym…2014)

### 3.5.4 Logger

Vaikka Androidin kernelissä on Linux-pohjainen kernel-kirjautumismekanismi, se käyttää myöskin toista alamekanismia jota kutsutaan loggeriksi. Tämä ajuri toimii logcat-komennon tukena jota käytetään lokien puskurien katsomiseen. Se sisältää tyypillisesti neljä erilaista puskuria:

- Main.
- Radio.
- Event.
- System.

Kuvassa 7 on esitelty Androidin kirjautumisjärjestelmän arkkitehtuuria sekä toiminnallisuutta.




Kuva 7 – Loggerin toiminta (Drake.J, Ym…2014)

### 3.5.5 Paranoid Networking

Paranoid Networkingin avulla Android Kernel rajoittaa kutsumisprosessin aikana käytettäviä verkkofunktiota. Tärkeimmät funktiot on eritelty alla olevassa taulukossa 5.

Taulukko 5 – Verkkofunktiot ryhmittäin (Drake.J, Ym…2014)

| AID Määritelmä | Ryhmä ID / Nimi | Funktio |
|---|---|---|
| **AID_NET_BT_ADMIN** | 3001 / net_bt_admin | Sallii Bluetooth-socketin luomisen. Diagnosoi ja hallitsee Bluetooth-yhteyksiä |
| **AID_NET_BT** | 3002 / net_bt | Sallii SCO, RFCOMM ja L2CAP-socketien luomisen |
| **AID_INET** | 3003 / inet | Sallii AF_INET and AF_INET6-socketien luomisen |
| **AID_NET_RAW** | 3004 / net_raw | Sallii RAW ja PACKET-socketien käytön |
| **AID_NET_ADMIN** | 3005 / net_admin | Myöntää CAP_NET_ADMIN kyvyn. Mahdollistaa verkkorajapinnan, reititystaulun sekä socketin manipulaation |

(Drake.J, Ym…2014)

# 4 Pääkäyttäjä-oikeuksien hallinnointi

Roottaus(eng.rooting) on prosessi jonka aikana Android-laitteen ns. super-käyttäjäoikeudet, toisin sanoen pääkäyttäjäoikeudet, valjastetaan laitteen omistajan käyttöön. Termi ”roottaus” tulee suoraan pääkäyttäjä-tilin nimestä root, joka on tismalleen sama kuin Linuxin pääkäyttäjä. Androidhan on UNIX-pohjainen ympäristö, joten sen termistössä on hyvinkin paljon samankaltaisuuksia samoin kuin useissa Linux-variaatioissa.
(Drake.J, Ym…2014)

### 4.1 Hyödyt vs haitat

Monet miettivät että kannattaako laitetta alkaa ”roottamaan”. Mitä se tekee, ja meneekö jotain rikki? Nämä ovat aivan normaaleja kysymyksiä, sillä onhan kyseessä rahanarvoisia tuotteita jotka sisältävät henkilökohtaisia tietojamme aina sähköposteista valokuviin. Mutta miksi ”rootata” sitten?

Monet esimerkiksi iPhonen käyttäjät valittavat että heidän käyttöjärjestelmän ulkoasuun kyllästyy, se on tylsä ja että he haluaisivat muokata sitä. Androidissa tällaista ongelmaa ei ole, koska siihen on tarjolla jo valmiiksi massiivinen määrä erinäisiä teemoja ja ikoneja. ”Roottauksen” avulla käyttäjät voivat asentaa laitteeseensa esimerkiksi heidän omatekoisia teemoja tai vaikkapa käynnistysanimaatioita. Monet haluavat mahdollisesti syventyä laitteensa järjestelmätietoihin enemmän. Jotta käyttäjä pääsisi tarkastelemaan tai esimerkiksi muokkaamaan niitä, on ”roottaus” välttämätöntä. Se mahdollistaa myös laitteeseen valmiiksi asennettujen ohjelmien tai ”bloatwaren” poistamisen. Näiden ohjelmistojen poisto ilman ”roottausta” on mahdotonta ja niiden käyttö on hyvinkin vähäistä, ainakin omien kokemuksien perusteella.

Käyttöjärjestelmässä on myöskin eräs itsenäinen ohjelmistoluokka nimeltään ”root apps”. Nimestäkin voisi jo melkein päätellä että kyseessä on ohjelmistoja jotka vaativat pääkäyttäjäoikeudet toimiakseen oikein. Tällaisia ovat esimerkiksi IP-tauluihin perustuvat palomuurit, mainostenesto-ohjelmat, tai vaikkapa erinäiset kellotusohjelmat. Nämä saattavat kiinnostaa käyttäjää mikäli hän on ns. tehokäyttäjä.

Sitten vastaavasti miksi ei kannata ”rootata” laitetta. Ensinnäkin jos toimenpiteen suorittaja ei tiedä mitä on tekemässä, on prosessissa iso mahdollisuus että se epäonnistuu. Tästä voi pahimmassa aiheutua jopa laitteen rikkoutuminen esimerkiksi ylikuumentumisen tai jonkun muun laiterikon johdosta. Toisin sanoen laite ”brikkaantuu”, eli muuttuu suhteellisen kalliiksi tiiliskiveksi. Se on ikävä asia etenkin siksi, että kun suoritat ”roottaus”– prosessin, laitteesi takuu raukeaa välittömästi. Tämä saattaa siis olla pahimmassa tapauksessa kallis kokeilu.

Oikeastaan tärkein syy perustuu turvallisuuteen. Harva käyttäjä jaksaa oikeasti paneutua laitteeseensa asentamiin ohjelmiin niin perusteellisesti että tietää varmasti mitä ne tekevät. Tämä aiheuttaa erittäin suuria tietoturvariskejä laitteellesi. Kun myönnät erinäisille ohjelmille täysiä root-oikeuksia, altistat omat henkilökohtaiset tietosi vaaraan. Lisäksi laitteen kadotessa, on sen anastajalla helppo työ varastaa kaikki siinä olevat tiedot.
(Drake.J, Ym…2014)

### 4.2 Laillisuus

Prosessin laillisuudesta on käyty paljon keskusteluja ja siitä on useita eri mielipiteitä. Jotkut maat ovat sen kannalla, ja jotkut sitä vastaan. Kaikki ovat kuitenkin yhtä mieltä siitä, että mikäli prosessin jälkeen laitteeseen asennetaan jokin ohjelma tai sovellus joka rikkoo tekijänoikeuslakeja, tuote on laiton. Tällaisessa tilanteessa kyseessä on piratismi, joka on laitonta kaikkialla maailmassa. Vastaavasti Applen iPhonen vastaava toimenpide ”Jailbreikkaus” on laitonta.
(Hoffmann C.2014)

## 5 Roottaus

Tässä kappaleessa käsitellään varsinaista roottaus-prosessia ja sen aikana käytetyt teknologiat.

Liitteissä 2 ja 3 suoritetaan varsinainen roottaus Android-laitteelle.

### 5.1 Partitio-layout

Partitiot eli osiot ovat loogisia varastokappaleita, tai divisioita, jotka on tehty laitteen pysyvään muistiin. Layout määrittelee erinäisten osioiden järjestyksen ja koon. Useimmissa laitteissa layoutin hallinnointi on Boot Loaderin vastuulla, tosin joissain laitteissa se on kernelin vastuulla. Yleisesti ottaen kaksi erinäistä Android-laitetta eivät käytä tismalleen samoja osioita tai yhtenäistä layouttia. Kaikilla Android-laitteilla on kuitenkin muutamat yhteiset osiot jotka ovat hyvin tärkeitä laitteen toiminnallisuudelle. Näitä ovat boot, system, data, recovery ja cache. Laitteen NAND-muisti osioidaan tyypillisesti taulukon 6 mukaan.

Taulukko 6 – Android-laitteen tyypillinen osiointi (Drake.J, Ym…2014)

| Osioin nimi | Tehtävä |
|---|---|
| **Boot Loader** | Varastoi laitteen boot loader-sovelluksen jonka vastuulla on laitteen hardwaren herätys käynnistyksessä, Android kernelin boottaus sekä muiden käynnistystilojen käyttö kuten Download-tila. |
| **Splash** | Varastoi ensimmäisen splash-kuvan laitetta käynnistettäessä.Tyypillisesti laitteen valmistajan logo. |
| **Boot** | Varastoi Android boot imagen, joka sisältää Linux kernelin(zImage) sekä root-tiedostojen RAM-levyn (initrd). |
| **Recovery** | Varastoi minimi Android-imagen joka tarjoaa huoltotoimintoja sekä toimii varmuuskopiona. |
| **System** | Sisältää Android-frameworkin, kirjastot, järjestelmäbinäärit sekä esiasennetut ohjelmistot. Polku: /system |
| **Userdata** | Kutsutaan myös data-osioksi. Laitteen sisäinen varasto johon kaikki käyttäjän oma data tallentuu, esim. kuvat, videot, äänet ja lataukset. Polku: /data |
| **Cache** | Käytetään palautuslokien sekä päivityspakettien varastointiin. Sisältää myöskin dalvik-cache kansion laitteissa joissa hyödynnetään SD-korttia ohjelmien säilytyksessä. |

### 5.2 Käynnistys-prosessi

Android-laitetta käynnistettäessä tyypillisesti ensimmäinen käynnistyvä osio on boot loader. Se sisältää laitevalmistajan omistuksessa olevan koodin joka vastaa hardwaren initiasaatiosta, herätyksestä, (eng. initialization) sekä mahdollistaa varmuuskopion palautuksen ja pääsyn Download-modeen. Jae lataa kernelin ja initrd:n boot osiosta RAM:iin initialisaation, jolloin kernel kykenee jatkamaan käynnistys-prosessia.

Androidin kernel ja sen toiminta ovat erittäin olennaisia laitteen käynnistysprosessia. Se vastaa muistin herätyksestä, input/output (I/O)alueista, muistin suojauksista, keskeytyksien käsittelijöistä (interrupt handlers), CPU:n aikataulutuksesta sekä laitteen ajureista. Toimintansa päätteeksi se käynnistää ensimmäisen käyttäjä-avaruus(User-Space) prosessin Initin.

Käyttäjä-avaruusprosesseista Init-prosessi on tärkein ja /init.rc-skripti toimii sen konfiguraatio-tiedostona. Se määrittelee mitä toimintoja mitä käytetään kun käyttöjärjestelmän käyttäjä-avaruus komponentteja herätetään. Niitä ovat muun muassa muutamia Androidin ydin-palveluja kuten rild(puhelin-ominaisuus), mtpd(VPN-yhteydet) sekä adbd (Android Debugger Bridge daemon). Tämän jälkeen Zygote-palvelu luo DalvikVM-instanssin ja käynnistää ensimmäisen Java-komponentin, System Serverin. Lopuksi muut Android Frameworkin-palvelut käynnistyvät. Kuva 8 esittelee init.rc-skriptin.

```
[...]
service adbd /sbin/adbd
    disabled
[...]
service ril-daemon /system/bin/rild
    socket rild stream 660 root radio
    socket rild-debug stream 660 radio system
    user root
    group radio cache inet misc audio sdcard_rw qcom_oncrpc diag
[...]
service zygote /system/bin/app_process -Xzygote
/system/bin --zygote --start-system-server
    socket zygote stream 660 root system
    onrestart write /sys/android_power/request_state wake
    onrestart write /sys/power/state on
    onrestart restart media
    onrestart restart netd
[...]
```

Kuva 8 – Init.rc-skripti (Drake.J, Ym…2014)

Käynnistyksen viimeistelee ACTION_BOOT_COMPLETED-tapahtuma joka lähetetään kaikille sovelluksille joiden manifestiin on rekisteröity tämän intentin vastaanotto. Tämän jälkeen järjestelmän käynnistys on valmis.
(Drake.J, Ym…2014)

### 5.3 Boot Loader

Koko roottaus-prosessi kulminoituu su-binääriin jolla on oikein määritelty UID-luvat järjestelmä-osiossa. Tämä mahdollistaa oikeuksien noston silloin kun niitä tarvitaan. su-binääriä

käytetään yleisesti ottaen jonkun Android-sovelluksen kanssa joka sisältää graafisen käyttöliittymän. Näitä ovat esimerkiksi SuperUser tai SuperSU. Nämä sovellukset kysyvät käyttäjältä lupaa aina kun jokin vaatii root-oikeuksia. Näiden ohjelmien avulla pystytään hallinnoimaan mitkä ohjelmat ja mitkä käyttäjät saavat root-oikeudet. Mikäli oikeudet myönnetään su-sovellus ajaa vaaditut komennot root-oikeuksin.

Root-oikeuksien haltuunotto on helppoa laitteissa joissa on avattavissa tai avattu oleva boot loader. Tämä sen takia että tällöin ei tarvitse hyödyntää paikkaamatonta turvallisuus aukkoa joka lukituissa laitteissa on. Käyttäjä kykenee tekemään kustomointeja täysin vapaasti boot loaderin ollessa auki. Tällöin käyttäjä voi modifioida tehdas-imagea ja lisätä sinne su-binäärin. Toimenpidettä voisi kutsua binääripatchaukseksi. Kuvassa 9 on esimerkki, jossa puretaan ext4-formaattinen järjestelmäimage, mountataan se, lisätään su-binääri sekä pakataan uudestaan. Laite roottaantuisi mikäli tämä ajettaisiin laitteeseen.

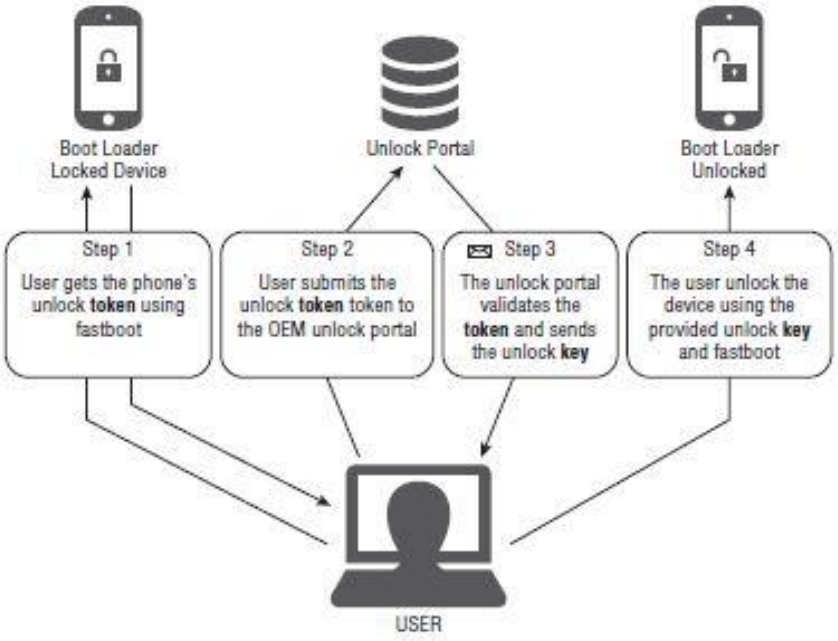
```
mkdir systemdir
simg2img system.img system.raw
mount -t ext4 -o loop system.raw systemdir
cp su systemdir/xbin/su
chown 0:0 systemdir/xbin/su
chmod 6755 systemdir/xbin/su
make_ext4fs -s -l 512M -a system custom-system.img systemdir
umount systemdir
```

Kuva 9 – su-binäärin lisäys (Drake.J, Ym…2014)

Liitteissä 2 ja 3 käsitellään toinen metodi joka on paljon suoraviivaisempi ja kenties käyttäjäystävällisempi. Niissä asennetaan kustomoitu image sekä järjestelmätietoihin ajetaan su-binääri joka ilmenee SuperSU-sovelluksena.

Android-versiosta 4.1 eteenpäin järjestelmässä on ollut sideload-toiminto. Tämän avulla laitteeseen voidaan asentaa zip-päivityspaketteja ilman että niitä tarvitsee ladata laitteeseen etukäteen. Tämä onnistuu antamalla komento adb sideload su-packege.zip, jossa su-packege.zip kuvastaa päivityspaketin nimeä.

Kun Boot Loader on avattu, joihinkin laitteisiin ei pysty ajaa allekirjoittamatonta koodia. Tälläisissa tapauksissa custom-imagen asennus on mahdollista vasta kun laite rootattu. Kuva 10 esittelee Boot Loaderin avausprosessin yleisellä tasolla.
(Drake.J, Ym…2014)




Kuva 10 – Boot Loaderin avausprosessi (Drake.J, Ym…2014)

# 6 Yhteenveto ja pohdinta

Projekti oli mielenkiintoinen ja hieman yllättäväkin, aiheena toimenpiteen toteutukseen käytettävien teknologioiden laajuuden huomioon ottaen. Työtä voisi jalostaa vielä lisää jolloin siitä saataisiin varmasti kattava paketti aiheesta. Tutkimusten perusteella tästä kyseisestä aiheesta ei löydy saman laajuista dokumentaatiota suomen kielellä. Työ ei kuitenkaan ollut kovin ajankohtainen, sillä onhan näitä toimenpiteitä tehty jo käyttöjärjestelmän alusta alkaen.

## 6.1 Tulokset

Tämän projektin tuloksena syntyi kattava dokumentaatio Android-mobiilikäyttöjärjestelmän historiasta, sen eri versioista, arkkitehtuurista, sen komponenteista sekä niiden toiminnallisuudesta. Lisäksi työn aikana jalostui tieto siitä, kuinka aiheena ollut toimenpide suoritetaan.

Työ olisi voinut toisaalta olla laajempikin, huomioon ottaen aiheena olleen Android-mobiilikäyttöjärjestelmän laajuuden kompleksisuuden. Tutkijana tietämykseni järjestelmän toiminnallisuudesta sekä sen arkkitehtuurista oli projektia käynnistettäessä mitäänsanomaton, joten projektin alussa asetettu oppimistavoite täyttyi.

### 6.2 Haasteet

Kuten aikaisemmassa kappaleessa todettiin että tutkijan tietämys aiheesta oli mitätön, lähteistä poimitut asiat menivät aluksi yli ymmärryksen. Tästäkin tosin selvittiin perinteisellä tutkimustyöllä.

# 7 Lähteet


Android Developer 2015. Luettavissa: http://developer.android.com/index.html Luettu:14.5.2015

Android Ohjelmointi. Luettavissa: http://some.lappia.fi/wiki/images/f/fe/Android_ohjelmointi_01.pdf Luettu: 21.5.2015

AndroidXDA 2015. How to root Samsung Galaxy S4 LTE. Luettavissa: http://androidxda.com/root-samsung-galaxy-s4-lte-gt-i9506 Luettu: 14.5.2015

Brachmann.S 2014. A Brief History of Google's Android Operating System. Luettavissa: http://www.ipwatchdog.com/2014/11/26/a-brief-history-of-googles-android-operating-system/id=52285/ Luettu: 28.4.2015

CyanogenMod 2014. How to Install CyanogenMod on the Samsung Galaxy S4 LTE-A (GT-I9506) Luettavissa: http://wiki.cyanogenmod.org/w/Install_CM_for_ks01lte_ Luettu: 14.5.2015

Drake.J, Fora.P, Lanier.Z, Mulliner.C, Ridley.S & Wicherski.G 2014. Android Hacker´s Handbook. John Wiley & Sons, Inc., Indianapolis, Indiana. s. 1- 83 Luettavissa: http://it-ebooks.info/book/3767/. Luettu: 21.4.2015

Hoffmann C. 2014. Is It Illegal To Root Your Android or Jailbreak Your iPhone? Luettavissa: http://www.makeuseof.com/tag/illegal-root-android-jailbreak-iphone/. Luettu: 21.4.2015

Hruska.J 2015.Google throws nearly a billion Android users under the bus, refuses to patch OS vulnerability. Luettavissa: http://www.extremetech.com/mobile/197346-google-throws-nearly-a-billion-android-users-under-the-bus-refuses-to-patch-os-vulnerability Luettu: 14.5.2015

Rosenblatt.S 2014. Android's phone wiping fails to delete personal data. Luettavissa: http://www.cnet.com/news/android-phone-wiping-fails-to-delete-personal-data/ Luettu: 14.5.2015

Statista 2015. Android OS: distribution of Android platforms used in February 2015. Luettavissa: http://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/ Luettu: 28.4.2015

# 8 Liite 1 – Terminologia

Alla esiteltynä muutamia termejä joita ei työssä ole avattu:

ADB – Android Debugger Bridge. Muodostaa sillan Android-laitteen ja tietokoneen välille.

API – Ohjelmointirajapinta.

APK – Android Application Package. Android sovelluksen koodi paketoidaan .apk muotoon.

Bloatware – Valmistajan esiasentama ohjelma jonka poistaminen on mahdotonta ilman root-tausta.

Brikkaantuminen – Laitteesta tulee kuvainnollisesti ”tiiliskivi”. Android-kehittäjien käyttämä slangisana.

DEX- Dalvik Executable file

GID – Group ID. Ryhmän yksilöivä tunniste.

Kernel – Käyttöjärjestelmän ydin.

IPC – Inter Process Communication. Prosesien välinen kommunikaatio.

ODEX – Optimoitu Dalvik Executable-tiedosto

PID – Process ID. Prosessin yksilöivä tunniste.

Roottaus – Toimenpide jonka aikana suorittaja saa laitteen pääkäyttäjä-oikeudet(SuperUser, root)

UID – User ID. Käyttäjän yksilöivä tunniste.

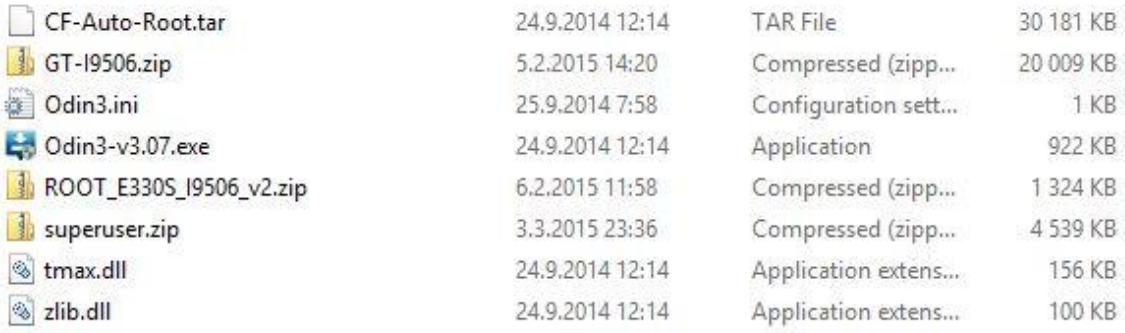
# 9 Liite 2 – Ohjeet roottaukseen

Tässä liitteessä esitellään miten Samsung Galaxy S4 LTE-malli rootataan hyödyntämällä Odin-sovellusta. Pari havaintoa kuitenkin ennen kuin suoritat prosessin. Tämän jälkeen mikäli laitteessasi on takuuta jäljellä se raukeaa.

Lisäksi jos roottaat laitevalmistajan oman imagen et pysty hyödyntämään enää heidän järjestelmäpäivityksiä. Mikäli näistä huomioista huolimatta haluat suorittaa prosessin, alta löytyy ohjeet siihen:

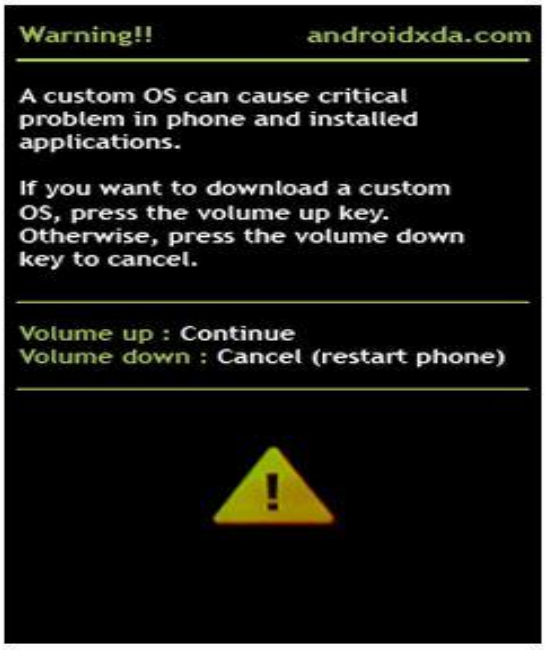
1. Varmista ensin että tietokoneessasi on Samsungin USB-ajurit asennettuna.

2. Lataa ja pura root-tiedostot koneellesi. Tiedostot näyttävät tältä:




Kuva 11 – Root-tiedostot

3. Sammuta laite seuraavaksi ja käynnistä se Download-tilaan. Tämä onnistuu painamalla Äänenvoimakkuus alas, Koti sekä Virtanäppäintä ja pitämällä niitä pohjassa noin 5-8 sekunnin ajan.



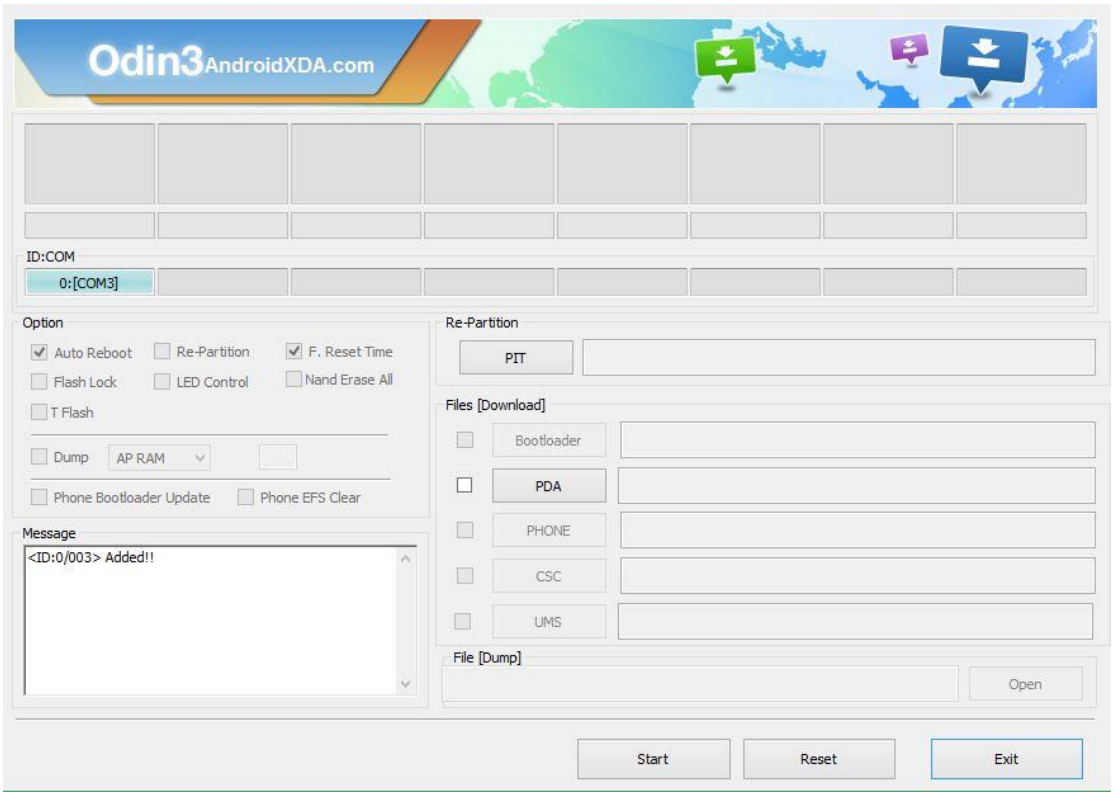

Kuva 12 – Download-tila

4. Download-tilaan päästyäsi laitteen ruudulle ilmestyy keltainen varoituskolia. Painamalla Äänenvoimakkuus ylös-nappia pääset eteenpäin.

5. Seuraavaksi käynnistä Odin3-ohjelmisto ja yhdistä laitteesi tietokoneeseesi.

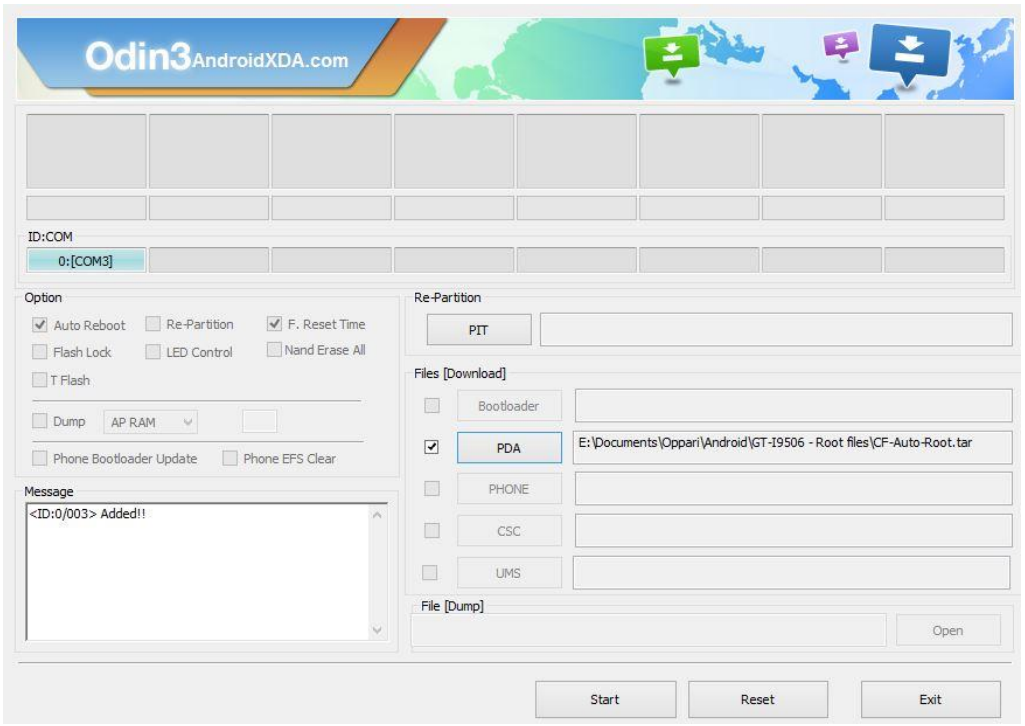
–

6. Kun olet yhdistänyt laitteen, Odin tunnistaa sen automaattisesti vasempaan alalaitaan ilmestyy ”Added”-teksti.



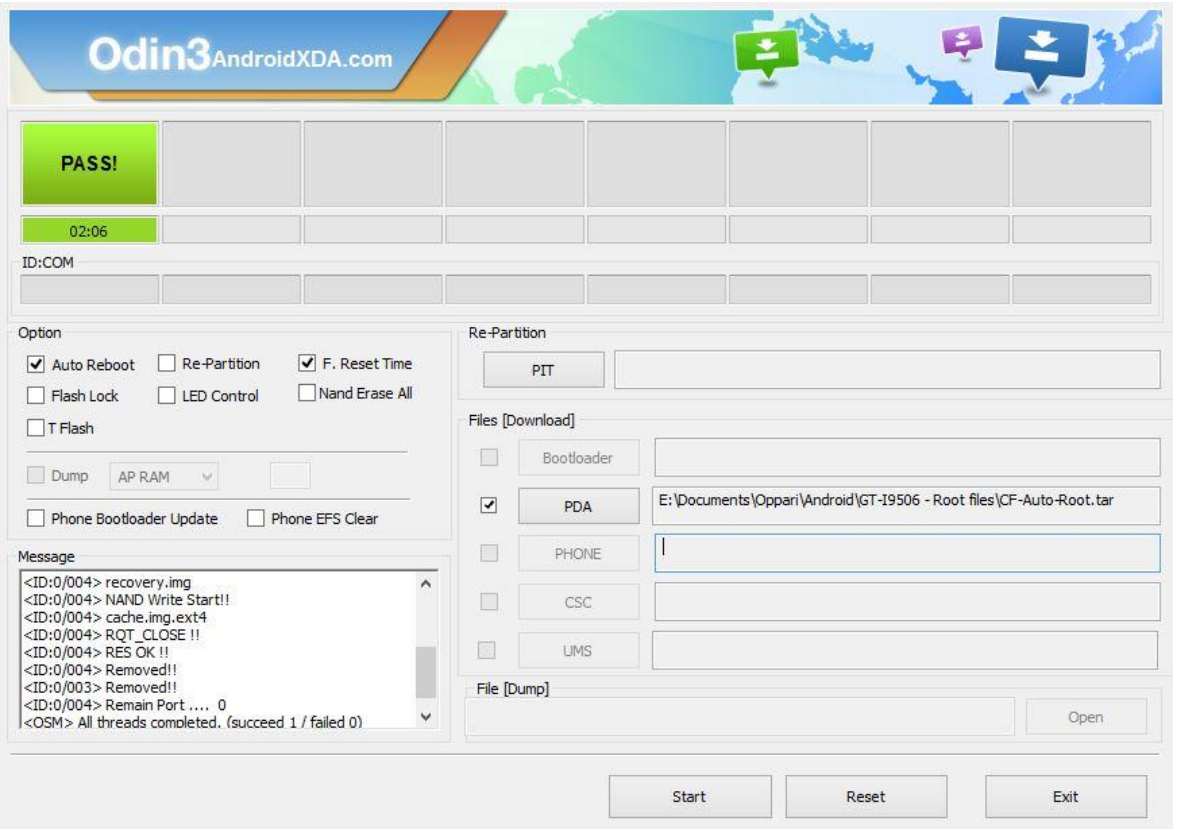

Kuva 13 – Odin tunnistaa laitteen

7. Tämän jälkeen paina PDA-näppäintä ja valitse siihen aikaisemmin ladattu CF-Auto-Root.tar-tiedosto



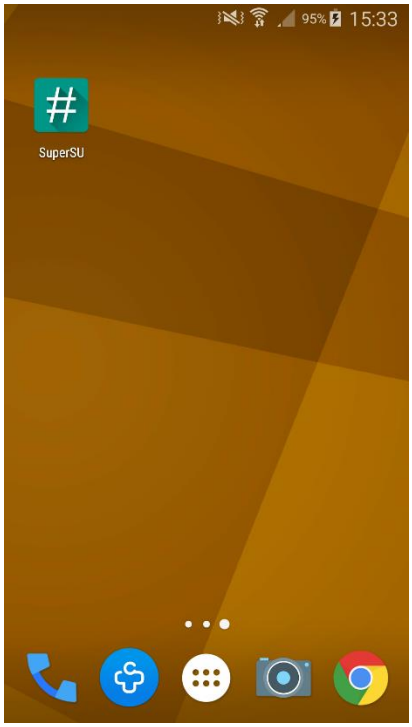

Kuva 14 – Tar-tiedoston lisääminen Odiniin

8. Painamalla Start-näppäintä Odin aloittaa laitteen flash-prosessin. Prosessi kestää noin puoli minuuttia, jonka jälkeen laite käynnistää itsensä uudelleen. Odinin vasempaan yläkulmaan ilmestyy myöskin vihreään laatikkoon PASS-teksti




Kuva 15 – Odin valmiina

9. Mikäli laitteesi työpöydälle on ilmestynyt SuperUser-sovelluksen pikakuvake, on roottaus-prosessi onnistunut ja voit aloittaa laitteesi hallinnoinnin.




Kuva 16 – SuperUser asennettuna

## 10 Liite 3 – CyanogenMod-Imagen asennus

Tässä liitteessä asennetaan CyanogenMod-ryhmän oma custom-image Samsung Galaxy S4 LTE-älypuhelimeen. Prosessin aikana hyödynnetään ClockWorkMod-ryhmän recovery-ohjelmistoa Samsungin oman recoveryn sijaan. Sen asennus on suoritettu samalla tavalla kuin liitteessä 2 asennetut root-tiedostot.

1. Varmista että tietokoneessa on toimiva adb eli varmista että koneella on sopivat USB-ajurit.

2. Lataa CyanogenMod-image heidän sivustoltaan. Lisäksi suositellaan asentamaan Google Apps – paketti, joka vaaditaan jos halutaan myöhemmin hyödyntää Googlen sovelluskauppaa.Siirrä .zip paketit esim. laitteen Downloads-kansioon.

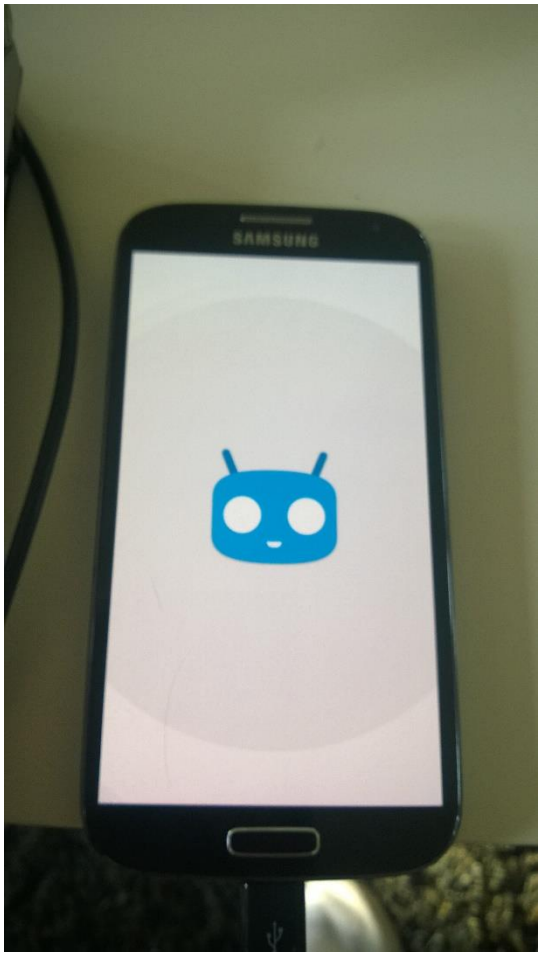
3. Käynnistä laite CWMRecovery-tilaan painamalla Äänenvoimakkuus ylös, Koti sekä Virta-näppäimiä niin kauan että laite käynnistyy tilaan. Virta-näppäin toimii ns. Enterinä.


4. Voit tallentaa laitteessa olevan tämänhetkisen varmuuskopion imagesta CWM:n avulla SD-kortille, josta sen voi myös palauttaa samalla tavalla.

5. Seuraavaksi tyhjennetään laite valitsemalla wipe data / factory reset.

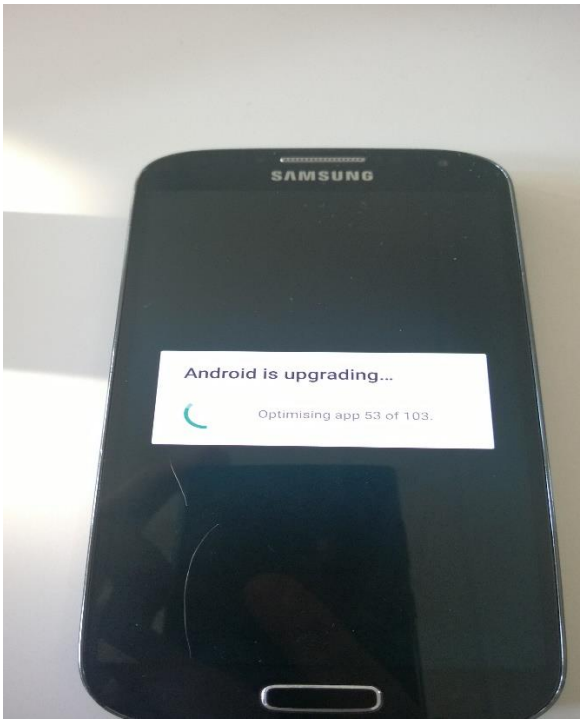
6. Laitteen tyhjennyksen jälkeen. Valitse install zip -> choose zip from /sdcard/0/Download sekä lataamasi .zip-paketti. CyanogenMod ensiksi, seuraavaksi Google Apps

7. Lopuksi reboot system now. Mikäli laitteen käynnistyessä CyanogenMod-logo ilmenee ruudulle, asennus on onnistunut.




Kuva 17 – CyanogenMod-käynnistyslogo

8. Laite käynnistelee itseään muutaman minuutin ja päivittelee uutta käyttöjärjestelmää samanaikaisesti optimoiden sovelluksia.

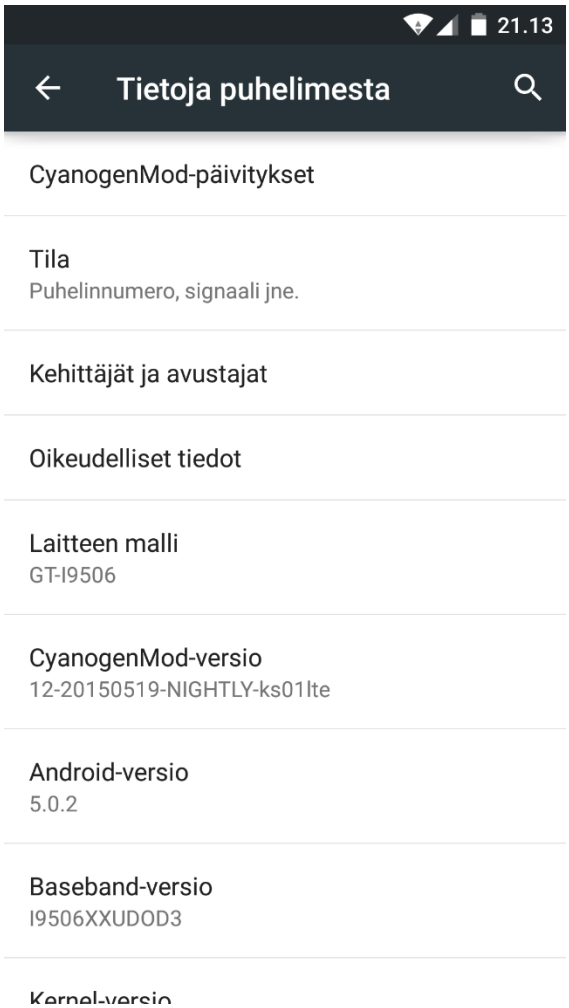



Kuva 18 – Sovellusten optimointi

9. Laitteen suoritettua sovellusten optimoinnin järjestelmän päänäkymä avautuu.


Kuva 19 – CyanogenMod Päänäkymä

10. Asetuksista voidaan varmistaa että CyanogenMod on asentunut onnistuneesti. Lisäksi on huomion arvoinen asia että tämä modi hyödyntää toistaiseksi julkaisematonta Android Lollipop 5.0.2 – versiota.




Kuva 20 – Käyttöjärjestelmän tiedot