SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3296

POBOLJŠANJE UPRAVLJANJA DOZVOLAMA NA OPERACIJSKOM SUSTAVU ANDROID

Denis Osvald

Zagreb, lipanj 2013.

Sadržaj

Uvod

Googleov operacijski sustav Android danas je prisutan i rasprostranjen na mnogo vrsta uređaja kao što su pametni telefoni, tablet računala, multimedijska pa nekad i stolna računala. Operacijski je sustav najvećim dijelom otvorenog koda, te zbog otvorenosti aplikacijske platforme i velike rasprostranjenosti na uređajima, za Android postoji mnogo aplikacija. Kao što je i inače slučaj s popularnim, široko implementiranim tehnološkim sustavima, za Android su se vremenom pojavile zlonamjerne aplikacije.

Android u svojem dizajnu ima sigurnost kao relativno bitnu karakteristiku, ali unatoč tome pokazalo se da su zlonamjerne aplikacije za Android brojne. Zbog toga je zanimljivo analizirati njegovu arhitekturu i sigurnosne mogućnosti, s razmišljanjem o nadogradnjama ili sustavu koji bi potpomogao sigurnost.

S druge strane, u pravilu na računalnim sustavima zlonamjerni program mijenja ponašanje sustava tako da ono više nije normalno i uobičajeno, već ispunjava neku novu (neželjenu) zadaću, pa se to može nazvati anomalijom na nekoj visokoj razini. Otkrivanjem takvih anomalija uslijed korištenja nekih programa bilo bi moguće pripisati im ponašanje koje je potencijalno zlonamjerno.

Pristup sigurnosti Android uređaja s stajališta – anomalije prilikom korištenja aplikacija – tema je ovog rada. U drugom poglavlju opisuje se Android i sigurnosne značajke njegove platforme za izradu aplikacija, te su navedene neke mogućnosti za njenu nadogradnju. Kako postoje prijašnji radovi na proširenju sigurnosti Androida, treće se poglavlje bavi nekim radovima vezanim za tu temu, gdje je uključen i pregled anomalija i otkrivanja anomalija kao temelj jednog proširenja Androida koje funkcionira na temelju prikupljenih podataka o radu sustava. Konačno, u četvrtom poglavlju opisano je dodavanje praćenja rada aplikacija i sustava odnosno ugradnja u sustav za sakupljanje informacija, na temelju kojih se može odlučivati o zloćudnosti aplikacija.

Sigurnost Android sustava

Motivacija tvoraca zlonamjernih programa može biti svakakva, ali jedna od sigurno važnijih činjenica jest specifična vrsta uređaja na kojima se ovaj operacijski sustav koristi. Kao što je ranije spomenuto, to su većinom pametni telefoni i tableti – ukratko, pametni mobilni uređaji, a na njima je nekoliko specifičnosti u području sigurnosti.

Prvo, takvi uređaji koriste se najviše za komunikaciju pa se samim time na njima nalazi i razmjenjuje više privatnih i osjetljivih informacija nego kod, primjerice, stolnih računala.

Prije pametnih telefona na koje korisnici mogu stavljati vlastite programe praktički jedini vid sigurnosnog rizika mobilnih telefona bila je fizička krađa. Pojavom prvih pametnih telefona na koje korisnici mogu stavljati vlastite programe (temeljenih primjerice na Symbian ili Windows Mobile) to se donekle promijenilo. Ipak, tek u posljednje vrijeme, dolaskom nove generacije mobilnih OS-ova kao što je Android, zastupljenost je uvelike porasla. Tako je danas sa stajališta sigurnosti postao realniji softverski faktor rizika, drugim riječima zloćudne aplikacije postale su vjerojatniji i češće sredstvo zlonamjernog napadača. Ciljevi koje takvi napadači mogu ostvariti uključuju slanje SMS poruka na brojeve s posebnom tarifom (a čiji je vlasnik napadač), krađa osobnih podataka iz imenika ili poruka, pozadinsko i neprimjetno prisluškivanje ili praćenje lokacije putem GPS i srodnih tehnologija koje su podržane.

Druga posebnost odnosi se na mjere zaštite od zlonamjernih napadača. Mobilni uređaji po svojoj prirodi koriste se u različitim mrežnim okruženjima, pa je vrlo teško osigurati uređaj korištenjem alata na razini mrežne sigurnosti. Osobna računala lakše je djelomično zaštititi uporabom vatrozida ili kontrole podataka na mreži u koju je ono spojeno, te će se tako smanjiti mogućnost da ono primi neželjeni softver s Interneta. Osim donekle slabije mogućnosti zaštite na mrežnoj razini, postoji i problem s uporabom lokalnih sigurnosnih alata kao što su antivirusni programi. Naime, osnovno svojstvo mobilnih uređaja jest rad na baterije. Korištenje klasičnih postupaka kao što je neprestano pozadinsko skeniranje

sustava otežano je, jer bi uzrokovalo vrlo brzu potrošnju baterije. Također, iako performanse memorije i procesora vrlo brzo rastu, u nekim uređajima oni ograničavaju uporabu alata koji intenzivno rade u pozadini.

Za Android je u ovom kontekstu specifična još i odvojenost različitih aplikacija u sustavu, pa tako jedna zaštitna (primjerice antivirus) aplikacija može teže otkrivati zloćudnu aktivnost ili blokirati takvo djelovanje.

Instaliranje aplikacija

Na Android se aplikacije mogu instalirati korištenjem više distribucijskih kanala. Načini na koji se aplikacije instaliraju na uređaj su vezani za sigurnost. Naime, neki servisi s kojih se aplikacije instaliraju kontroliraju i uklanjaju aplikaciju u slučaju da je ona zlonamjerna, dok korištenjem drugih načina nema te mogućnosti.

Službeni je Googleov kanal Google Play Store. Na uređajima postoji istoimena aplikacija koja je vezana za korisnikov Google račun, i koja ima mogućnost direktno instalirati druge aplikacije koje korisnik odabere ili kupi.

S druge strane, postoji još nekoliko drugih "tržišta" odnosno repozitorija aplikacija koje je moguće koristiti i funkcioniraju na sličan način kao Google Play. Među njima najpopularniji su Amazon Appstore, Aptoide, F-Droid, Getjar, SlideME te razni vezani za mrežne operatere ili proizvođače. Konačno, aplikacije se mogu pribaviti i iz ostalih izvora i prebaciti na razne načine na uređaj, te ručno instalirati neovisno o bilo kojem servisu. U tom slučaju koristi se sistemska aplikacija *Package Installer*. Treba napomenuti da ako na uređaju postoji Google Play Store, moguće je instalirati aplikaciju na ovaj način, ali i uz dodatnu prethodnu provjeru lokalno pribavljene aplikacije, korištenjem tog Googleovog servisa koji će prijaviti eventualne nađene zlonamjerne programe.

Spomenuta sistemska aplikacija (*Package Installer*), odnosno aplikacija jednog od korištenih repozitorija (ako su ugrađeni u sustav) ne vrši sama instalaciju aplikacija, već pristupa sistemskoj komponenti koja to čini. Pristup toj komponenti ograničen je na aplikacije sustava, što je važno jer se njima vjeruje da ne instaliraju aplikacije bez potvrde korisnika.

Aplikacije koje se instaliraju na Android su apk (zip) datoteke, paketi koji sadrže izvršni kod, potrebne resurse, eventualne biblioteke te manifest. U manifestu se nalaze informacije o aplikaciji, kao što su naziv paketa, zahtjevi na verziju Android platforme i karakteristike uređaja, popis javnih komponenti aplikacije te zahtijevane dozvole.

Dozvole i digitalni potpisi

Pretpostavljeno aplikacija radi u izoliranoj okolini te nema pristup svim resursima i mogućnostima koje su dostupne u sustavu. Aplikacija pristup njima dobiva ako deklarira u manifestu da koristi takve mogućnosti odnosno pripadajuće dozvole. Pri instalaciji paketa pretpostavlja se da je korisnik dao aplikaciji sve dozvole koje su tražene u manifestu, kako je i prikazano prilikom instalacije. Dozvole se provjeravaju prilikom korištenja funkcionalnosti koje traže primjerice privatne podatke spremljene na uređaju, detaljne informacije o uređaju te pristup resursima koji mogu biti osjetljivi (mikrofon, GPS prijemnik, kamera i slično), ili kada aplikacija kontrolira neki važan resurs kao spajanje na mrežu.

U manifestu aplikacije mogu najaviti i vlastite dozvole koje su potrebne za korištenje nekih njenih dijelova ako su oni javni (namijenjeni drugim aplikacijama), pa će se korištenjem tih komponenti provjeriti postojanje dozvole u aplikaciji iz koje se koristi. Finija kontrola dozvola izvodi se programski.

Paket je digitalno potpisan korištenjem kriptografije javnog ključa s samopotpisanim certifikatom programera. Pri nadogradnji aplikacije, provjerava se je li certifikat potpisa nove verzije odgovara staroj. Ova mjera pruža zaštitu od zlonamjerne izmjene ili nedozvoljene nadogradnje ako se radi o nadogradnji trenutno instalirane aplikacije. Provjera se vrši i pri instalaciji aplikacije koja eksplicitno u svom manifestu deklarira da želi dijeliti podatke i komunicirati s nekom postojećom. Aplikacije to čine tako da u manifestu zahtijevaju zajednički korisnički identifikator i moraju biti potpisane istim certifikatom. Prilikom instalacije aplikacije koja nije već instalirana na uređaju, certifikat nije poznat pa ga se ne može usporediti s već postojećim. Provjera certifikata s certifikacijskim tijelom (CA), koja bi u ovom slučaju omogućila autentifikaciju potpisa, ne koristi se.

Kada je riječ o ugrađenim aplikacijama sustava poput onih za postavke ili telefoniranje, zapravo se radi o aplikacijama koje preko dozvola traže i dobivaju pristup mogućnostima koje su dostupne samo sustavu. Preciznije, svaka dozvola u Androidu ima pridružen nivo zaštite. Dozvole za korisničke aplikacije mogu osim običnih biti i "opasne", pa će se u prikazu korisniku prilikom instalacije kao takve posebno naglasiti. Sistemske dozvole su one koje imaju nivo zaštite koji zahtjeva potpis sistemskim certifikatom ili ugrađenost aplikacije u sistemsku particiju ugrađene memorije (dozvola za izravnu instalaciju aplikacija koja je ranije navedena jedna je od takvih). Sistemski certifikat je ovdje jednostavno onaj certifikat kojim su potpisane komponente sustava (koje nisu nužno posebne aplikacije).

Izoliranje aplikacija od pristupa sustavu provedeno je, dakle, kroz dozvole, dok je međusobna odvojenost aplikacija izvedena kroz pokretanje različitih aplikacija pod vlastitim korisničkim identifikatorom. Odvajanje korisnika u aplikacijama koristi i za izolaciju kad se koriste nativne biblioteke u aplikacijama kako se ne bi zaobišla provjera dozvola i slično. Aplikacije su pokrenute pod istim korisničkim identifikatorom samo kad obje to zatraže.

Podaci i postavke i baze podataka koje aplikacija koristi većinom su spremljeni na datotečni sustav u memoriji uređaja. Tako se spremaju i dijelovi raspakiranih (instaliranih) aplikacija. Na razini datotečnog sustava interne memorije koriste se Unix dozvole i vlasništvo. Vlasništvo i dozvole postavljeni su tako da odvojene aplikacije nemaju pristup datotekama drugih aplikacija.

Nedostaci i moguća proširenja

Opisana sigurnosna arhitektura ipak ima svoje nedostatke.

Jedan od problema je što se vrlo slično programsko sučelje koristi za izradu korisničkih aplikacija kao i za one u sustavu, iako to s druge strane omogućava i veću fleksibilnost odnosno prilagodljivost platforme. Kako aplikacije sustava koriste opisane dozvole koje nisu dostupne korisničkim aplikacijama, neki dijelovi takvog sučelja sakriveni su za

korisničke aplikacije (uporabom anotacije `@hide`), te su namijenjeni samo za privatnu uporabu u sustavu. Međutim, takva se sučelja ipak mogu koristiti uporabom *Java Reflection API*. Prividna nedostupnost tih sučelja može uzrokovati da se takva funkcionalnost čini neiskoristiva no ipak je treba tretirati kao javnu te provjeravati svaki pristup kroz dozvole.

Kako aplikacije mogu imati svoje javno sučelje, postoji mogućnost da aplikacija koja pristupi nekim osjetljivim informacijama te informacije podijeli drugima bez kontrole: namjerno, ako je zlonamjerna, ili slučajno, kada je u pitanju sigurnosni propust. U nekim slučajevima propusti u sistemskim aplikacijama mogu ozbiljno ugroziti sigurnost [1], a proizvođači koji koriste Android na svojim uređajima često ugrađuju vlastite aplikacije. Greške poput takvih, u kojima aplikacija dijeli podatke ili mogućnosti koje ne bi smjela, zapravo nisu propusti u platformi, no pokazalo se [2] da mnogo aplikacija kroz dozvole traži više nego je potrebno za rad te aplikacije. I u službenoj dokumentaciji u prošlosti se mogao naći krivo dokumentiran rad s nekim dozvolama zbog kojih se otvara mogućnost da se ona iskoristi za curenje podataka: dokumentacija je navodila da je potrebna jača dozvola nego što je bilo potrebno.

Sam sustav dozvola osmišljen je dobro, jer nije potrebno da svaka aplikacija može pristupati svemu, ali ima neke mane. Najznačajnija mana jest nemogućnost upravljanja njima nakon instalacije aplikacije. Ako neka aplikacija traži neki skup dozvola, nije joj moguće selektivno dodijeliti samo neke dozvole. Jedina mogućnost jest odbiti instalirati takvu aplikaciju. Na taj način, predvidivo je da se među korisnicima stvori navika da te dozvole doživljavaju samo kao još jedan korak koji treba proći pri instalaciji, bez stvarnog obraćanja pažnje na njih.

Kod provjera koje dozvole ima aplikacija koja poziva neke dijelove programskog sučelja, sustav provjerava sve dozvole vezane uz taj korisnički identifikator. Kao što je opisano, više aplikacija može, zatraži li svaka to u manifestu, dijeliti korisnički identifikator. Sve dozvole tako povezanih aplikacija tada zajedno čine skup dozvola koje *svaka* od tih aplikacija može koristiti. Na taj način moguće je da jedna aplikacija pristupa nekim resursima za koje nema sama dozvolu, ali je tu dozvolu dobila putem druge. Ovaj mehanizam omogućava modularni dizajn aplikacija, a sigurnost je donekle zadržana jer samo aplikacije istog programera (potpisane istim certifikatom) mogu tako surađivati. Ipak,

ovakav model zahtijeva da korisnik vjeruje svim aplikacijama pojedinog programera i dozvole razmatra kao skup, različito od uobičajenog načina da svaka aplikacija radi za sebe. Na primjer, ako dvije takve aplikacije surađuju i jedna traži pristup mikrofonu, a druga dozvolu za rad u pozadini čim se uređaj upali, moguće je da druga mikrofon koristi cijelo vrijeme u pozadini.

"Nadogradnja" kroz ukidanje ovakvog dijeljenja dozvola teoretski bi pomogla u većoj sigurnosnoj izolaciji aplikacija, ali bi značajno smanjila mogućnosti aplikacija. Ipak, sustav se u radu oslanja na ovakvu funkcionalnost pa je to neizvedivo bez velikih promjena. Umjesto toga, dovoljno bi bilo da se, prilikom instalacije, uz prikaz dozvola koje će nova aplikacija dobiti, navede i informacija o postojećim dozvolama povezane aplikacije s kojim se one dijele. Bolja dokumentiranost ovakvih mogućnosti i njihova vidljivost korisnicima mogla bi se još proširiti da uključuje osnovne informacije o javnom sučelju aplikacije koje ona definira u manifestu, jer bi se kroz njega mogle podijeliti osjetljive informacije kojima ta aplikacija pristupa.

Također, osim modela sve-ili-ništa za dodjelu dozvola, moguće je zamisliti neke drugačije izvedbe njihovim upravljanjem. Jedan bi bio da se omogući da za neke dozvole aplikacije svaki put trebaju dobiti potvrdu korisnika. Googleovo je stajalište da bi takvo ponašanje korisnike još više izmorilo da ne obraćaju pažnju na to što se traži. Druga varijanta bila bi da se aplikaciji uskrate neke odabrane dozvole nakon instalacije, a još jedna mogućnost bila bi da se za određene zahtjeve ovisno o dozvolama za privatne informacije pokazuju lažni ili nepostojeći podaci.

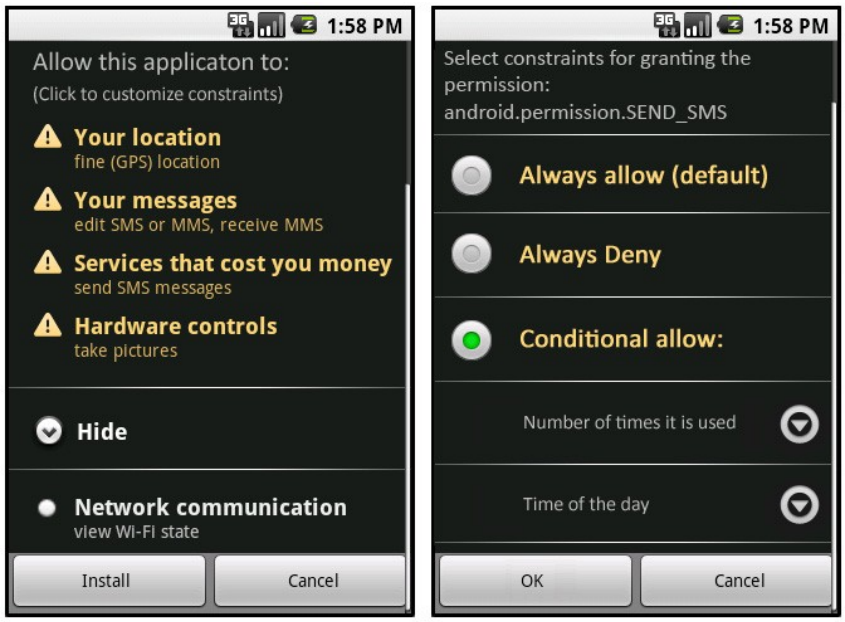
Postojeći sustavi i prijašnji radovi

U prošlosti je bilo dosta radova o nadogradnji operacijskog sustava Android s ciljem poboljšanja sigurnosti ili otkrivanja zlućudnih aplikacija, kao i analiza. Vezano za ovu tematiku je i otkrivanje anomalija u općem smislu, koje se onda može konkretno primijeniti u slučaju Android aplikacija. Opis nekih od takvih nadogradnji odnosno pratećih sustava dan je u nastavku.

APEX

Problemom dodjele i zabrane pojedinih dozvola bavi se članak APEX [3], i istoimena implementacija sustava (nadogradnje Androida). Glavni cilj je omogućiti da se, pri instalaciji aplikacija, može ograničiti odobravanje dozvola. Osim binarne odluke dodjele ili zabrane pojedine dozvole, omogućeno je i dinamičko odlučivanje o pristupu ovisno o nekim vanjskim vrijednostima (vrijeme, učestalost zahtjeva i drugo). Na slici 1 vidljivo je grafičko sučelje kojim se ograničavaju dozvole.

Dinamičko odlučivanje i kontrola pristupa definirani su formalno. Dane su formalne definicije dozvole, uvjeta pristupa i komunikacije među aplikacijama, odnosno pokretanja drugih aplikacija. Formalizacija sigurnosnih mogućnosti Androida je provedena na ovaj način kako bi se izradio jezik za opis pravila pristupa. Sama pravila za odlučivanje se mogu posebno odrediti za svaku aplikaciju. Pravila su izvedena u tako napravljenom jeziku i s ovim sustavom čine osnovni okvir za izradu sigurnosne politike koja kontrolira dozvole aplikacija.



Slika 1: Mogućnosti kontrole dozvola u sustavu APEX [3]

Slično prethodno navedenom radu, neka proširenja za Android sustav nastala su u njegovim neslužbenim verzijama (*forkovima*). Zajedničko ovim nadogradnjama za dozvole je da se uglavnom radi o isključenju pojedinih dozvola, no budući da aplikacija koristi svoju funkcionalnost i ne zna za tu promjenu u dozvolama, pri radu dolazi do sigurnosnih iznimaka i aplikacije su nestabilne. Takvo neželjeno ponašanje u ovakvim proširenjima je dovelo do napuštanja rada na nekima od njih, jer dovode do temeljne nekompatibilnosti s uobičajenom Android platformom i kao takva nisu poželjna.

Postoje, međutim, drugi sustavi koji, kako bi se navedeni problem nestabilnosti aplikacija riješio, dodatno proširuju platformu. Sistemske komponente koje poslužuju zahtjeve za funkcionalnosti (na primjer pristup kontaktima) koje inače trebaju dozvole mogu se proširiti ili se oko njih naprave dekoratori (omotači) koji koriste stvarnu funkcionalnost, ili vraćaju lažne ili nepostojeće podatke, ovisno o postavljenim dozvolama i odabiru korisnika.

Nasuprot navedenim mogućnostima da se kontroliraju dozvole i rad neke aplikacije ili pojedinih njenih komponenata, neka proširenja vezana su samo za analizu rada aplikacija,

ali ne i upravljanjem radom. U kontekstu sigurnosne terminologije, to je sustav za otkrivanje napada (i*ntrusion detection system*), nasuprot sustavu za sprečavanje napada (i*ntrusion prevention system)*. U obje vrste sustava potrebno je u osnovi moći odrediti kada se dogodilo neko ponašanje koje predstavlja napad.

Detekcija anomalija

Anomalija općenito znači odstupanje od predviđenog ili uobičajenog uzorka ponašanja. Obično je takvo odstupanje u nekom promatranom ponašanju posljedica posebnog događaja kojeg je zanimljivo pratiti.

Detekcija anomalija u ponašanju problem je kojemu se može pristupiti na mnogo načina, čija primjenjivost ovisi o domeni problema, a svakom od tih načina je u osnovi cilj podijeliti ulazni skup uzoraka na normalne i neuobičajene, koji označavaju anomalije. Varijacije u primjenjivosti postupaka ne ovise samo o domeni problema nego i o obliku podataka u kojima se traže uzorci ponašanja – na primjer, vrijednosti iz ulaza mogu biti diskretne ili kontinuirane vrijednosti, može ih biti više, istog ili različitog tipa, a podaci mogu nositi i vremensku povezanost, ili pak povezanost u graf. Sve to utječe na odabir parametara i postupka detekcije, kao što su prikladna metrika za udaljenost uzoraka ili pojedinosti statističkog modela za klasifikaciju.

Nekoliko je važnih karakteristika postupaka detekcije anomalija, bez obzira na konkretnu primjenu, a to su: vrsta anomalija koje će biti prepoznate, označenost ulaznog skupa te izlaz detektora.

Anomalije koje se prepoznaju mogu biti točkaste (ako se podaci ističu i odstupaju od normalnih pojedinačno), kontekstualne (ako je odluka za odstupanje podatka ovisna o kontekstu, to jest okolnim podacima) ili kolektivne (skup podataka predstavlja anomaliju u odnosu na ukupne podatke). Anomalije jednog tipa često se ne detektiraju ako se promatraju kao druge, primjerice bez njihovog konteksta. Zato je nekad korisno za detekciju kolektivnih anomalija (ili ako su podaci slijedno vezani) problem i ulazne podatke transformirati u oblik gdje će jedna točka možda predstavljati neki skup iz

originalnih podataka, ali će se problem svesti na nalaženje točkastih anomalija, što može biti jednostavnije ili pouzdanije.

Označenost ulaznog skupa odnosi se na dostupnost ulaznih podataka koji su već označeni kao normalni ili anomalije. Postojanje tih informacija (podaci za učenje) omogućuje potpuno ili djelomično vođen rad postupka detekcije, te se on može prilagoditi odnosno strojno naučiti da daje točnije izlaze.

Izlazna vrijednost detektora anomalije može biti direktna odluka o označivanju anomalije, ili može biti bodovana ljestvica pojedinih uzoraka iz ulaznih podataka, kako bi se kasnije filtrirale anomalije ovisno o nekom pragu, koji se može i dinamički mijenjati i pruža fleksibilnost.

Konkretni postupci detekcija anomalija mogu se svrstati u nekoliko kategorija, a svaki od njih temelji se na svojoj pretpostavci za rad. Različiti načini imaju svoje prednosti i mane kao što su potreba za podacima za učenje, vrsta izlaza (oznaka ili bodovanje) ili računska složenost.

Pristup temeljen na klasifikaciji pretpostavlja da je moguće izraditi i naučiti klasifikator koji će podatke klasificirati u jednu ili više normalnih klasa, a izdvojiti anomalije (podaci koji nisu u nijednoj klasi). Prednost klasifikacijskog pristupa je što se jednom dobro naučeni model ispituje na ulaznim podacima uz malu složenost. Ipak, klasifikacijom ne može se dobiti bodovni rezultat za podatke, već će se oni označiti kao normalni ili ne, Očito, za rad je potrebno imati podatke za učenje, pa ovakav pristup ne možemo koristiti ako nemamo takve podatke. Konkretno učenje modela klasifikacije može biti izvedeno koristeći Bayesov vjerojatnosni model (računajući aposteriorne vjerojatnosti), sustav temeljen na pravilima ili neuronske mreže.

Detekcija se može izvesti na temelju promatranja najbližih susjeda. Podaci se predstavljaju kao točke u prostoru, i definira se neka metrika kojom se računaju udaljenosti. Bodovna vrijednost nekog podatka se dodjeljuje u ovisnosti o udaljenosti do susjednih točaka, primjerice kao funkcija udaljenosti k-tog najbližeg susjeda, ili kao funkcija lokalne gustoće točaka. Prag za odluku o anomaliji, funkcija, metrika mogu se odabrati na mnogo načina, a razvijene su i razne izmjene ovakvih postupaka koje daju bolje rezultate u posebnim slučajevima. Za razliku od prošlog pristupa, ovaj ne zahtjeva podatke za učenje,

omogućava osim označavanja anomalija i bodovno rangiranje, ali je značajno veće računske složenosti.

Pristup sličan prethodnom je i analiza klastera. Oba pristupa u postupku koriste međusobnu udaljenost točaka podataka, ali pri analizi klastera je naglasak na skupovima bliskih točaka (klasterima). U tom smislu se u analizi klastera vrednuju udaljenosti do točaka u klasteru umjesto točaka lokalne okoline. Klasteri predstavljaju normalne podatke, dok su anomalije daleko od najbližeg ili izvan svih klastera, ili anomalije čine svoj klaster manje gustoće.

Ostali pristupi uključuju sa stajališta teorije informacija izdvajanje prema količini informacija koje podaci donose u skup, spektralne metode te statistički pogled u kojem se traže podaci koji ne odgovaraju pretpostavljenom stohastičkom modelu za konkretni problem.

Kao što je spomenuto u uvodu, anomalije u domeni problema obično označavaju posebno važne događaje ili segmente. Neke od primjena mogu biti otkrivanje prijevara na bankovnim transakcijama ili osiguranju, medicinske primjene u pronalaženju tumora, otkrivanje kvara uređaja temeljem podataka sa senzora, obrada slike te mnoge druge.

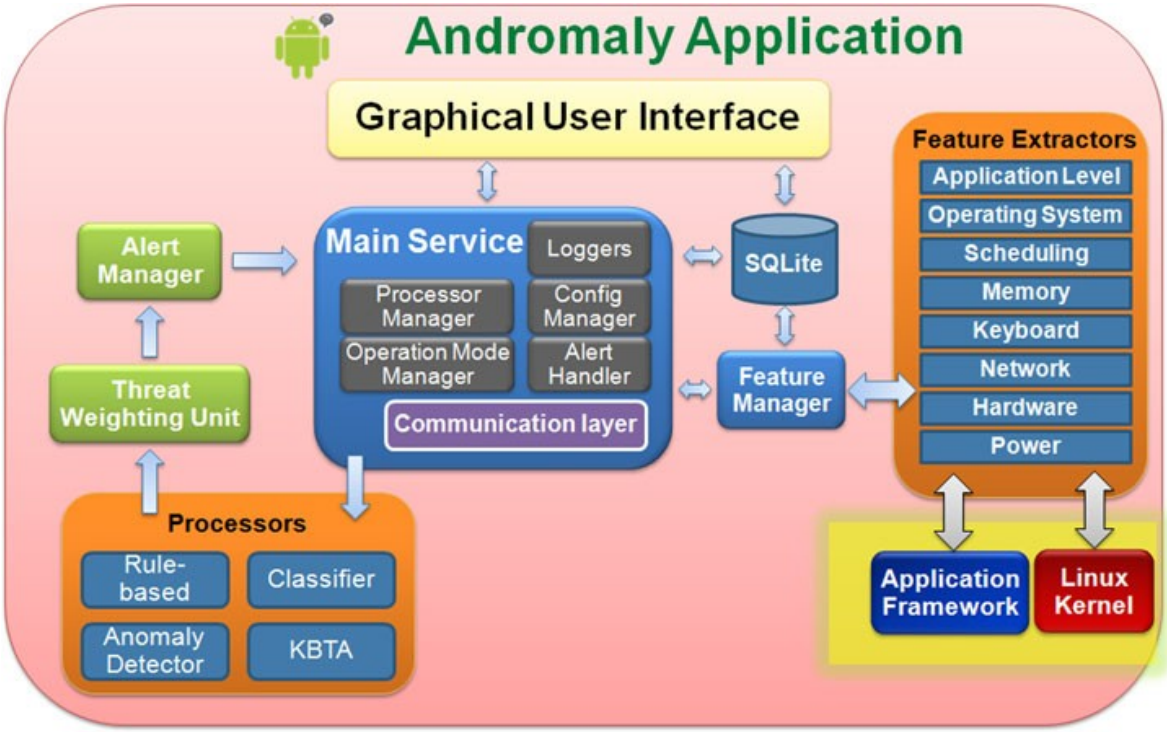
Ovdje najzanimljivija primjena jest detekcija napada na računalnu mrežu ili sustav. Posebna podvrsta su sustavi koji rade na lokalnom računalu i otkrivaju ponašanje koje ukazuje da je ono napadnuto (*Host based intrusion detection system*). Obično se za takve sustave prikupljaju podaci o korištenju sistemskih poziva ili drugih komponenata sustava (primjerice čitanjem dnevnika). Anomalija u prikupljenim podacima može biti indikator napada, nedozvoljenog pristupa ili iskorištavanja ranjivosti, i takvo ponašanje korelira s prisutnošću zloćudnih programa. [4]

Andromaly

Sustav Andromaly [5] primjena je metoda detekcije anomalija za otkrivanje zloćudnih aplikacija na Androidu. Temelji se na servisu koji u pozadini sakuplja razne informacije o sustavu tijekom rada aplikacija. Sakupljene značajke se koriste u strojnom učenju kako bi sustav, na temelju pokazatelja za određene aplikacije, mogao naučiti predstavlja li neka od

aplikacija prijetnju odnosno je li zloćudna ili ne. Odluka se temelji na razlici između vrijednosti različitih značajki za normalan rad i rad zlonamjerne aplikacije.

Sustav je organiziran kroz više komponenti, kao što je prikazano na slici 2. Glavna komponenta je servis (*Main Service*) koji upravlja radom ostalih dijelova. Kroz grafičko sučelje upravlja se postavkama sustava. Odabire se profil rada odnosno značajke koji će se sakupljati (*Feature Manager*), postupci obrade tih vrijednosti *(Processors)* te akcije za obavještavanje o rezultatima (*Alert Manager*).



Slika 2: Organizacija sustava Andromaly [5]

Sakupljanje značajki odvija se periodički, kada se pojedini sakupljač značajki aktivira i bilježi vrijednost. Konkretne značajke razrađene u sustavu Andromaly su mnogobrojne, a mogu se podijeliti na one vezane uz Linux jezgru i pojedinosti Android platforme. Kako se u konačnici koristi strojno učenje za rad sustava, uz veći broj značajki omogućeno je da se postupak tako prilagodi za bolje rezultate i da rabi samo one značajke koje daju bolje rezultate analize. Kroz više faza se eliminiraju značajke koje su objektivno imale nizak doprinos u analizi, pa se ona i ubrza.

Komponente za obradu značajki (procesori) oblikovane su da na ulazu primaju vrijednosti značajki i daju procjenu prijetnje na izlazu kao rezultat obrade. Predviđeno je da se obrada može raditi neovisno od ostatka sustava, bilo koristeći sustav pravila, bilo da se koriste tehnike strojnog učenja pa se vrijednosti klasificiraju kao uobičajene ili kao anomalije, ili kombinacija nečeg od navedenog. Izmjenom ovih komponenti može se birati korišteni postupak detekcije anomalija.

Nakon obrade, izlazi iz svih procesora uzimaju se u obzir pri donošenju odluke o prijetnji neke aplikacije. Konačna odluka može rezultirati da se poduzme neka automatska radnja kao što je stopiranje aplikacije, ili samo obavještavanje korisnika, što je zadaća komponente *Alert Manager*.

Osim arhitekture sustava, u radu su opisani i rezultati odnosno zaključci korištenja takvog sustava za detekciju zlonamjernih aplikacija, što uključuje i usporedbu različitih algoritama klasifikacije te izdvajanje najkorisnijih značajki.

Kao zaključak rada navodi se da je predloženi sustav imao dobre rezultate pri strojnom učenju na dvadesetak normalnih Android aplikacija i zatim ispravno odredio druge četiri izrađene kao zloćudne, što su one i bile. Sustav opisan u radu nije objavljen, odnosno nije dostupan za korištenje niti proširivanje.

Sustav za praćenje dozvola i aplikacija

Kao što je navedeno, sustav koji povećava sigurnost Android OS-a može biti izrađen slično sustavu Andromaly koji je opisan u prethodnom poglavlju. Cilj takvog sustava je pružanje korisniku Android uređaja obavijesti o sigurnosnim rizicima u aplikacijama koje su instalirane na uređaju, a tako i njihovu kontrolu.

Implementacija proširenja se temelji na izvornom kodu aplikacije *Anomaly Detector* [6]. Ta je aplikacija je napravljena u nekoliko komponenta, i obavlja funkciju prikupljanja vrijednosti značajki, i tako bi odgovarala komponentama *Feature Manager* i *Feature Extractors* sustava Andromaly. Komponente same aplikacije odnose se na pozadinski servis koji upravlja i periodički pokreće prikupljanje vrijednosti značajke, te konkretnih komponenti koje sakupljaju te vrijednosti. Da bi se pozadinski servis mogao pokrenuti odnosno zaustaviti iz aplikacije, aplikacija ima sučelje za korištenje.

Razmatraju se dvije nadogradnje aplikacije, a obje se odnose na konkretne informacije koje se prikupljaju: praćenje dozvola koja pojedina aplikacija koristi, te praćenje kada se koji dio aplikacije pokreće i zaustavlja. Ove značajke mogle bi nositi dosta informacija koje bi koristile pri određivanju zloćudnosti aplikacije. Praćenje dozvola moglo bi dijelom pružiti uvid u korištenje poziva programskog sučelja sustava. Ako se u nekom trenutku dozvola odobri, znači da se tada koristi neka funkcionalnost za koju je ta dozvola potrebna, pa trenutak odnosno redoslijed njihovog odobravanja i korištenja može pokazati uzorke ponašanja za praćenje. Odbijena dozvola nije uobičajena pojava, ali može pokazati ponašanje aplikacija. Praćenje pokretanja komponenata aplikacije daje informaciju o tijeku izvršavanja dijelova aplikacije.

Za praćenje ovih značajki ranije opisani mehanizam odvajanja aplikacija u Androidu predstavlja prepreku. Procesi aplikacija rade pod različitim korisničkim identifikatorom, pa ostale aplikacije ne mogu pristupiti svim informacijama, a tako niti pratiti ove značajke. Također, da bi se odredilo kako se koja aplikacija koristi, ili kada ju je korisnik pokrenuo, potrebno je pristupiti sistemskim informacijama koristeći pozive programskog sučelja. Za takve mogućnosti potrebne su posebne dozvole, od kojih neke nisu dostupne običnim aplikacijama.

Prvi dio odnosi se na prikupljanje informacija o korištenju dozvola od strane aplikacija. U Android platformi ni za aplikacije ni za sustav nema nikakve dostupne funkcionalnosti kojom bi se moglo to pratiti. Postoji jedino mogućnost da se dohvati popis informacija o instaliranoj aplikaciji, uključujući i njene dozvole. Da bi se ostvarilo vremensko praćenje korištenja dozvola, potrebna je promjena u sustavu, i to u komponenti *Package Manager*, koja upravlja instaliranim aplikacijama, metapodacima o njima, te pregledava njihove dozvole.

Korištenju dozvole od strane neke aplikacije prethodi provjera ima li ta aplikacija tu dozvolu. U Androidu se provjera dozvola provjerava na spomenutom jedinstvenom mjestu (*Package Manager*), i to olakšava promjene kojima se omogućava praćenje. Dozvole se mogu provjeravati pri pokretanju dijelova aplikacija koje su tu dozvolu naznačile, tijekom slanja podataka drugim aplikacijama, ili pri korištenju poziva programskog sučelja. Svaki od načina provjere, neovisno je li pozvan iz aplikacije u sustavu ili instalirana od korisnika, te nevezano pokreće li se iz pozadinskog servisa ili aktivne aplikacije, završava u sistemskoj komponenti odnosno servisu *Package Manager*. Provjera dozvola vrši se tako da se tražena dozvola pronađe u bazi dodijeljenih dozvola za korisnički identifikator aplikacije.

Promjena Android sustava koja omogućava da se prati korištenje pojedine dozvole nalazi se u toj komponenti. U izvornom kodu, radi se o promjeni u kodu razreda `com.android.server.pm.PackageManagerService`. Dvije metode u tom razredu, `checkPermission` i `checkUidPermission` mjesta su na kojima se donosi odluka o dozvoli, a jedna ili druga koristi se ovisno ima li pozivajuća aplikacija dijeljeni korisnički identifikator. Prije povratka iz tih metoda, potrebno je proslijediti aplikaciji za praćenje informaciju o ishodu provjere (je li dozvola odobrena) zajedno s vremenom te aplikacijom odnosno korisničkim identifikatorom. Aplikacija ove informacije bilježi kao vrijednost značajke vezane za korištenje dozvola. Budući da su aplikacija i komponenta sustava koja daje informacije o dozvoli odvojeni procesi različitih korisnika, potrebno je informaciju prenijeti nekim načinom komunikacije među procesima.

Android pruža komunikaciju među procesima koristeći nekoliko načina. *Intent* je osnovni mehanizam dohvaćanja drugih aplikacija odnosno njihovih dijelova (to *Activity* kao vidljivi dio aplikacije sa svojim grafičkim sučeljem ili *Service* kao pozadinski aktivan servis), a

omogućuje pokretanje i zaustavljanje uz predaju podataka kao argumente ili povratni rezultat. Ovaj način komunikacije, iako se može događati između različitih procesa, je osnovni i ograničeni slučaj jednokratne komunikacije. Jednokratna komunikacija između dvije pokrenute aplikacije može se ostvariti i korištenjem komponenti *Broadcast* i *BroadcastReceiver*, gdje se poruka "oglašava" jednom ili više primatelja. Za komunikaciju među aplikacijama koja je redovita ili se često ponavlja, koriste se servis koji implementira posebno zajednički definirano AIDL sučelje, koje Android dopuni metodama za udaljeno pozivanje iz druge aplikacije. Tada se druge aplikacije prijave da komuniciraju sa servisom, a *Binder* sustav će povezati pozive. *Binder* se koristi i pri komunikaciji temeljenoj na porukama, kod kojeg se umjesto definiranja sučelja i udaljenog poziva koristi kombinacija *Messenger*, *Handler* odnosno red poruka.

Zadnji način, komunikacija porukama, odabran je za prenošenje informacija iz nadograđene komponente sustava u aplikaciju. Uobičajen je slučaj u Androidu da se aplikacije vežu za servise u sustavu, koji im pružaju neke podatke ili funkcionalnost. Za ovu nadogradnju to nije provedeno na taj način jer postoje ograničenja na mogućnost povezivanja servisa. Umjesto toga, u aplikaciju je postavljen servis koji kad je pokrenut prihvaća i obrađuje poruke o korištenim dozvolama, a u sustavu se ostvaruje veza na taj servis. U korisničku aplikaciju promjena se uključuje kao nova konkretna komponenta za sakupljanje vrijednosti. Također je potrebna i odgovarajuća izmjena u komponenti tvornici, da bi ona mogla stvarati primjerke za konkretne značajke koje se promatraju. Kako je taj dio izvornog koda organiziran po obrascu tvornice, moguće ga je ostvariti neovisno, kao generičku tvornicu, no u Androidu je nešto složenije izraditi dinamičko uključivanje komponenti u aplikacije bez promjene, zbog specifičnog pakiranja aplikacija.

Drugi dio, praćenje pokretanja i gašenja aplikacija i njihovih dijelova moguć je u Android SDK razvojnim alatima za aplikacije, a tamo služi za otklanjanje pogrešaka pri razvoju. Ova mogućnost koristi se iz razvojnog okruženja, s Android uređajem za razvoj ili emulatorom. To pruža naredba `am monitor`. U Android platformi aktivnosti vezane za životni vijek aplikacija nalaze se u *Activity Manager* komponenti, a ova naredba pristupa privatnim dijelovima tog sučelja koji su dostupni unutar sustava, ali ne i aplikacijama platforme. Kako je i navedeno, skrivanje važnih dijelova operacijskog sustava iz sučelja

moglo bi se zaobići, pa je korištenje ograničeno i dozvolom, `SET_ACTIVITY_WATCHER`, a ta dozvola dodjeljuje se samo aplikacijama koje su potpisane certifikatom sistema. Pri razvoju aplikacija, naredba se može koristiti jer je sama dio platforme, a za pokretanje svih naredbi pri razvoju koristi se posebni korisnički identifikator koji ima traženu dozvolu. Iz ovih ograničenja proizlazi poteškoća integriranja prikupljanja opisanih podataka u aplikaciju. Dobra alternativa ovom načinu praćenja aktivnosti aplikacija je periodičko čitanje popisa aplikacija koje su pokrenute, iako to daje nešto manje informacija. Kako u aplikaciji za prikupljanje već postoji praćenje procesa, nadogradnja u ovom dijelu nije provedena.

S obzirom na provedenu implementaciju praćenja dozvola i sve navedeno, moguć je niz budućih nadogradnji.

Prvo, da bi cijeli sustav imao smisla prikupljene značajke o korištenju trebaju se iskoristiti u svrhu detekcije zloćudnih aplikacija. Temelj za rad sustava može biti detekcija anomalija, a u konkretnom slučaju praćenja dozvola koje je ostvareno, može biti korišten postupak kao i za otkrivanje zloćudnih programa uz praćenje sistemskih poziva.

Drugo, budući da u trenutnoj implementaciji praćenje dozvola ovisi o promjeni sustava, i tako zahtijeva da se praćenje neke aplikacije odvija u pripremljenoj okolini, nije moguće pratiti aplikacije na običnim uređajima bez njihove pripreme za ovu svrhu (kroz instalaciju promijenjenog Androida koja na neke uređaje nije moguća). Praćenje dozvola aplikacija bez posebnog izgrađenog ili zakrpanog sustava sigurno nije moguće (jer bi takva mogućnost bila propust i koristila samim zloćudnim aplikacijama), ali je ipak korisno za analizu zloćudnih aplikacija. Ipak, poželjno bi bilo da se postavljanje praćenja dozvola na neki uređaj omogući bez izgradnje cijelog OS-a, već da se zahtijevaju samo *root* (administratorske) ovlasti koje su prisutne na uobičajenom emulatoru i češće dostupne na uređajima od postavljanja cijelog OS-a. Za navedeno bi trebalo istražiti mogućnosti nadogradnje već izgrađenog sustava.

Konačno, spomenuto je praćenje komponenata aplikacija, ali ta mogućnost nije implementirana. Buduća implementacija po mogućnosti bila bi u skladu s prethodnim zahtjevom.

Zaključak

Sigurnost uređaja koji rade s Android operacijskim sustavom važna je upravo zbog prirode uređaja na kojima se on koristi. U skladu s tom namjenom, Android je dizajniran sa sigurnosnim mogućnostima koje su opisane u drugom poglavlju. Android je sustav koji se aktivno razvija, pa se opisano odnosi na verziju 4.2 (*Jelly Bean*), a u sljedećim se inačicama očekuju i dodatno sigurnosno ojačavanje, kao što se i uobičajeno sigurnost sustava razvija i poboljšava. Ipak, neki od opisanih problema sigurnosti u Androidu ne mogu se u novim inačicama riješiti bez gubitka kompatibilnosti, ili su pak rezultat čvrstih odluka pri dizajnu sustava (što ima i dobre i loše strane), a zloćudne ih aplikacije mogu iskorištavati.

Prijašnji radovi na temu sigurnosti Androida su brojni, i u trećem poglavlju prikazano ih je par. Uglavnom se ti radovi bave dozvolama u Androidu, a osim njih postoje i razna proširenja u neslužbenim verzijama Androida. Nešto drugačiji je rad Andromaly, koji se temelji na otkrivanju zloćudnog ponašanja aplikacija. Rad tog sustava temeljen je na otkrivanju anomalija u podacima koji se prikupljaju, pa je u istom poglavlju dan i pregled načina kako se anomalije otkrivaju.

Primjena detekcija anomalija na sigurnosne zaštite operacijskog sustava obično se ostvaruje analizom tijeka poziva iz aplikacije kroz vrijeme. U četvrtom poglavlju opisano je kako se mogu ugraditi takve mogućnosti praćenja, uz objašnjenja sistemskih komponenata koje su u tom dijelu važne. Opisane i izrađene nadogradnje služit će za izradu sustava nalik na Andromaly, a koji u konačnici može služiti za analizu ponašanja aplikacija.

Takav bi se sustav mogao prilagoditi i koristiti da u pozadini radi na ciljnim (običnim) uređajima i informira korisnika o aplikacijama koje koristi, ili bi mogao raditi kao namjenski prilagođen sustav koji služi isključivo za iscrpniju analizu aplikacija, i čiji rezultati se prosljeđuju.

Literatura

[1] Moulu, A: *Android OEM's applications (in)security and backdoors without permission*, 2013, http://www.quarkslab.com/dl/Android-OEM-applications-insecurity-and-backdoors-without-permission.pdf

[2] Felt, A.P.; Chin, E.; Hanna, S.; Song, D.; Wagner, D.. Android Permissions Demystified, 2011

[3] Nauman, M; Khan, S; Zhang, X.. Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints, 2010

[4] Chandola, V.; Banerjee, A.; Kumar, V.. Anomaly Detection: A Survey, 2007

[5] Shabtai, A.; Kanonov, U.; Elovici, Y.; Glezer, C.; Weiss, Y.. "Andromaly": a behavioral malware detection framework for android devices, 2011

[6] Humić, B.: *Anomaly Detector*, 2013, https://github.com/bhumic/anomaly_detector

[7] Felt, A.P.; Chin, E.; Hanna, S.; Song, D.; Wagner, D.: *Permission Map*, 2011, http://android-permissions.org/permissionmap.html

[8] Android Open Source Project: *Downloading and Building*, 2013, http://source.android.com/source/building.html

[9] Android Open Source Project: *Technical Information*, 2013, http://source.android.com/tech/index.html

[10] CyanogenMod Project: *Wiki*, 2013., http://wiki.cyanogenmod.org/w/Main_Page

[11] Android Developers: *Documentation*, 2013, http://developer.android.com/develop/index.html

[12] Brahler, S.. Analysis of the Android Architecture. Karlsruher Institut fur Technologie, 2010

Dodatak – blog postovi

U nastavku su preneseni u cijelosti blog postovi pisani tijekom izrade rada. Poveznice iz svakog od tekstova su izdvojene kako bi bile vidljive.

Arhitektura i sigurnost operacijskog sustava Android

[http://sgros-students.blogspot.com/2012/11/arhitektura-i-sigurnost-operacijskog.html]

Tekst ovog posta opisuje temu kojom ću se baviti kroz projekt na preddiplomskom studiju, a u budućnosti i diplomskom. Naglasak je za sada samo na preddiplomskom studiju, iz razloga što ga je lakše planirati.

Rad će se sastojati od proučavanja operacijskog sustava Android, s fokusom na njegovoj arhitekturi, po mogućnosti istražujući sigurnosne aspekte. To će uključivati najprije istraživanje načina na koji je sustav dizajniran, i na temelju općenite slike odabir nekih od konkretnih dijelova za daljnji rad. Konkretno, Android u sebi ima više slojeva - Dalvik virtualni stroj, Linux jezgra, Linux korisnički prostor (engl. userspace), itd., te će vrlo vjerojatno rad biti ograničen na samo jedan od tih dijelova (ili barem jedan po jedan).

Motivacija odnosno način na koji je ovo proizašlo kao predmet ovog projekta vezan je za moj prijedlog moguće teme mentoru. To se odnosilo na istraživanje povezanosti Linux jezgre s korisničkim prostorom: postoji mnogo operacijskih sustava baziranim na Linux jezgri (Android, webOS, Chrome OS, Firefox OS, ...) koji imaju različite primjene, ali bi bilo zamislivo njihovo izmjenjivanje ovisno o željenoj primjeni (na istom uređaju i jezgri), te implementacija funkcionalnosti za tu svrhu

Kako je navedeno bilo dosta opsežno ali ne previše jasno, odabrana je tema koja je također vezana za razinu operacijskog sustava, ali konkretnije (Android) i s jasnijom orjentacijom (sigurnost sustava). Kako se Android u velikoj mjeri danas koristi, tema je neupitno korisna. U konačnici, cilj rada bila bi neka poboljšanja u sigurnosti sustava, ili identifikacija mogućih problema.

Sa svim ovim u vidu, prvi od koraka za rad na projektu biti će pregledavanje materijala, službenih i onih sakupljenih u sklopu drugih radova, koji se bave Android operacijskim sustavom.

Android overview

[http://sgros-students.blogspot.com/2013/05/android-overview.html]

In order to work on Android and it's security, it was necessary for me to become familiar with the Android system, architecture, fundamental topics and frameworks in general. I shall describe that briefly in this post.

A starting point for this was a thesis paper[1] written by Stefan Brahler, titled "*Analysis of the Android Architecture*." The paper provides an outline of how applications are installed and run; how application bytecode is executed; what power management techniques are used, Android Linux kernel specifics, etc. There are also sections about Android framework APIs and application development. Additionally, the official documentation[2] as another resource is invaluable and was perused extensively. These were good resources, and I will attempt to summarize some points from them which may be useful:

Structurally, Android is based on the Linux kernel, but lags a few versions behind the mainline version. The Android Linux kernel historically had more differences from mainline than in newer versions, but in newer versions they are diminished. Android-specific components interface between the kernel drivers and applications (e.g. SurfaceFlinger for graphics). At the userspace level, some well known Linux and tools are used mainly for system services e.g. bluetooth (bluez), wifi (wpa_supplicant), VPN.

System and user applications are written in Java, compiled into Dalvik bytecode and run in a virtual machine (Dalvik VM), which additionally acts as a sandbox layer. The Java APIs for Android applications are very similar but not identical to the Java SE APIs. Applications may also contain native library code, which is not executed in a sandbox, unlike Dalvik code.

Multi-user capabilities of the OS are used to separate applications from each other. Each application runs as a separate user and cannot access additional resources unless explicitly granted.

Access to such resources of an Android device (sensors, camera), contents (files, contacts, media), connectivity (calls, texts, internet) etc. is controlled through permissions. These are defined mostly by the system services and applications, and applications using them must declare them in their manifest, and are then granted at install time by the user.

Poveznice

[1] http://os.ibds.kit.edu/downloads/sa_2010_braehler-stefan_android-architecture.pdf

[2] http://source.android.com/tech/index.html

Scenarios of having an Android device stolen

[http://sgros-students.blogspot.com/2013/05/scenarios-of-having-android-device.html]

One simple aspect of Android device security includes the protections one has when their device is stolen. A brief consideration of these related to Android follows. The features that Android offers which are relevant are screen / device lock, device encryption and external storage.

Screen lock locks the device into a lockscreen after inactivity (or on user request). To use any features of the device, the user must use the owner-selected unlock mechanism. PIN, passphrase, pattern or face unlock are available. Face unlock, present on some devices, uses the front camera and face recognition to match faces. Pattern unlock uses a rectangular (3x3) grid of dots through which the user swipes their finger in a pattern on the screen. PIN and password/passphrase correspond to entering a preset number or phrase. From a security standpoint, some of the methods are weaker than others. Face unlock can be defeated by obtaining a photo of the owner's face and presenting it to the camera, while pattern unlocks can be broken by viewing smudges present on the touchscreen. The PIN method is similar to a password, but provides a smaller password space. Apart from preventing access to the device by using its interface, the screen locking mechanisms also prevent computer connectivity (acces to media storage, or developer/debug access) when locked.

Device encryption on Android[1] uses the dm-crypt mechanism in Linux for block-level encryption of the device's /data partition on internal storage. The internal data partition holds most of the device user's application data and applications themselves. External storage such as SD cards remain unencrypted in plain Android systems, but some vendors provide an option for encryption. If external storage is unencrypted and present on a device, data may be obtained from a compromised device. Although Google's developer guidelines[2] state that security-critical data should not be stored there, external storage contains most of user's data, such as camera photos, media and documents, in addition to other nonsensitive application data. External storage also provides no facility for fine

access control - unlike application data on internal storage (which uses privilege separation). Applications are only given (or not given) the permission to access the external filesystem with read-write permission. On older Android versions, even though the permission exists, it is not enforced.

The lack of encryption for the device's /system partition is another point to address but is not a big issue. The /system partition is read-only mounted, so root access is required to remount it, as well as access most files because of ownership and permissions. Moreover, there is no private data to extract from that partition, because the device never modifies any files in /system except when installing core software updates. One protection that an encrypted /system adds is that it prevents tampering with the device's system files (e.g. adding malware) using hardware methods (more direct chip access).

Here are some possible loss/theft scenarios concerning data security and how the above mentioned features affect it:

Example scenarios

- device with no locking - the attacker can access all data and accounts, and depending on the device, may be able to plant undetected malware.
- device with screen lock and no encryption - the attacker cannot easily access any data on the device without trying to crack the lock / PIN / password. If the lock is bypassed, or either way with hardware access, complete access is possible, or malware may be planted.
- device with data encryption - the attacker cannot retrieve data even with hardware access. It may be possible (but probably detectable - dismantling may be required) for the attacker to use low-level hardware access to plant malware into the unencrypted system portion and return the device to the owner in order to gain access.

Poveznice

[1] http://source.android.com/tech/encryption/android_crypto_implementation.html

[2] http://developer.android.com/training/articles/security-tips.html

Emulators, Cyanogenmod and Android

[http://sgros-students.blogspot.com/2013/05/emulators-cyanogenmod-and-android.html]

When developing Android applications, the Android emulator may be useful for testing, because it provides better debugging support or devices are not available. The Android emulator is based on the QEMU[1] machine emulator and supports MIPS, ARM and x86 architectures. Currently, most Android devices are ARM-based, but because of performance issues with emulating ARM on the host PC, the x86 variant is commonly used for testing applications. The x86 emulator can use the host processor's virtualization extensions, which greatly increases performance

Android platform changes may also be suitable for testing in the emulator. The Android Open Source Project provides simple instructions for building[2] and testing on an emulator. Since Android supports various mobile devices and due to their variety, the build is device-dependent, and the emulator is technically just another device.

In comparison, the cyanogenmod.org wiki[3] currently doesn't provide specific instructions on building for the emulator, such as it does for the supported devices. Nevertheless, the old wiki has a page[4] describing precisely this, but it is slightly outdated. Although the build process is very similar to the official AOSP one, some users were reporting various problems[5] with the build. The problems seem related to compatibility of the kernel with the emulator, and may have been resolved in the meantime, although I can't find recent success reports online. The build system does provide a device target `full` which refers to the emulator debug image as in Google's AOSP build instructions, so it appears that the only relevant information about whether it currently works on the emulator can be seen by testing it:

I succeeded in building the cyanogenmod image from source and ran it on the emulator. It should be noted that at that time, in two most recent code branches (corresponding to android releases 4.1 and 4.2) the build failed when targeting an x86 system (for example, sometimes there was inline ARM assembly in some sources), but finished normally for

ARM targets. The build system is set up so that once the image is build, the emulator can be launched for it directly.

Poveznice

[1] http://wiki.qemu.org/Main_Page

[2] http://source.android.com/source/building.html

[3] http://wiki.cyanogenmod.org/w/Main_Page

[4] http://oldwiki.cyanogenmod.org/wiki/Android_SDK_Emulator:_Compile_CyanogenMod_%28Linux%29

[5] http://forum.cyanogenmod.org/topic/39203-compiling-cyanogenmod-gb-branch-from-git-for-android-sdk-emulator-kernel-panic/

Thoughts about enhancing application permission control

[http://sgros-students.blogspot.com/2013/05/thoughts-about-enhancing-application.html]

It occurred to me and (a lot of other people, as will be shown) that with Android applications, it may be useful to extend permission management for installed applications to enable logging/auditing or finer grained control. I remembered seeing some application which provided some of that functionality, but was unsure of exact details. Since this area of Android just might be suitable for me to try some enhancements, it seemed appropriate to explore the following points:

- What existing solutions or ideas are available for extending Android's permission system? What are their principles of operation?
- How does the Android system enforce Application permissions, and where is the relevant component in it's source code?
- What can malware do with regard to application permissions?

The existing application / modification and how these work

It turns out that there have been a couple of similar attempts to enhance permissions on android, below listed in no particular order:

- PDroid, made by xda-developers.com community members. Consists of Android source patches and controlling user application. The patches modify system services which provide data for telephony, location and similar API calls (`TelephonyManager`, `LocationManager`, ...), creating wrappers for these components. The wrapper components check for application-set rules for handling permissions or logging. Patches (source) available, but application is closed-source.
- Cyanogenmod had modifications in the system components and application (UI) for

revoking specific permissions to applications. Applications behaved as though the permission was not requested at install time by the application, and may crash. Theese extensions are no longer present in newer versions (since they caused applications to misbehave, become incompatible, etc.)

- papers Android Permissions Demystified[1], APEX[2], and a few other referenced therein. These enhancements have different approaches, goals or implementations which are described well in the respective papers, but do not have source code available.

- a few proprietary or commercial tools such as LBE Privacy Guard

Identifying components in Android system which grant permissions when application does an action which requires permission

Android's developer documentation[3] on permissions outlines general situations when permissions are necessary: API calls for activity or services, accepting data from other applications (receiving broadcasts) or providing it (content providers).

The documentation also shows actual permissions required for calls to certain parts of the API. A somewhat useful representation of this is the permission map[4] which was created from the API as part of the Android Permissions Demystified project mentioned above.

The aforementioned papers Android Permissions Demystified[1] and APEX[2] provide a description of how the permissions are checked, around page 3, or pages 5, 6, section 4, for the two papers respectively. A summary of this follows. When an application calls an API method, the call is propagated through Android's inter-process communication to the system service which can service the API request (example: `LocationManagerService` provides location). The system component classes must check for permissions regarding to the action, using methods such as `checkPermission`, `enforcePermission` from Context (be it Activity, Service or other context). The actual decision is done in the `PackageManagerService` which grants or denies permissions depending on those requested at install time.

How simple malware might be done, relating to permission management

Some permissions in the Android system are broad, while others are fine-grained. An example of broad permission is the `INTERNET` permission, which allows any socket access. Malicious applications might claim to be using the permissions for some legitimate and benign task, while in reality that permission allows it to do additional malicious functionality.

Poveznice

[1] http://www.cs.berkeley.edu/%7Eemc/papers/android_permissions.pdf

[2] http://csrdu.org/pub/nauman/pubs/apex-asiaccs10-long.pdf

[3] http://developer.android.com/guide/topics/security/permissions.html

[4] http://android-permissions.org/permissionmap.html

Sažetak

Poboljšanje upravljanja dozvolama na operacijskom sustavu Android

Operacijski sustav Android namijenjen je mobilnim uređajima i u širokoj je uporabi. Raširenost i popularnost potaknula je na izradu zlećudnih aplikacija koje mogu omesti rad uređaja, krasti podatke, špijunirati korisnike i slično. Sigurnosne zaštite ugrađene u operacijski sustav uvelike pomažu, ali ne uklanjaju sve rizike. Jedan od sigurnosnih mehanizama Androida je sustav dozvola kojim aplikacije traže pristup resursima potrebnim za rad. Sakupljanje i praćenje korištenja tih dozvola može se koristiti zajedno sa sustavom koji bi detektirao anomalije u podacima i otkrio potencijalno zlećudne aplikacije.

Ključne riječi: sigurnost, Android, privatnost, dozvole, nadzor, zlećudna aplikacija, detekcija anomalija

Abstract

Improvement of permission management on Android operating system

The Android operating system is designed for mobile devices and is in widespread use. Its spread and popularity have encouraged creation of malicious applications which can disrupt functionality, steal data, spy on users, etc. Android's security mechanisms greatly help diminish, but do not eliminate all risk. One of the security features in Android is the permissions model, under which each application seeks access to resources it requires. Collecting data and tracking of permission usage can be used in conjunction with a system which would detect anomalies and reveal potentially malicious applications.

Keywords: security, Android, privacy, permissions, audit, malware, malicious application, anomaly detection