ANALIZA MOBILNIH APLIKACIJ NA PLATFORMI ANDROID

Milan Gabor, Danijel Grah

Viris d.o.o.
e-pošta: milan@viris.si, danijel.grah@viris.si
URL: http://www.viris.si

Povzetek: *Količina in vrednost podatkov s katerimi razpolagajo mobilne aplikacije narašča skladno s širjenjem trga »pametnih« mobilnih telefonov in integracijo mobilnih aplikacij z vsakdanjim življenjem posameznikov. Zaradi aktualnih dogodkov in čedalje bolj kompleksnih tehnologij se širi tudi nezaupanje v legitimno početje mobilnih aplikacij. Trendom naraščanja posledično sledi povpraševanje po analizi mobilnih aplikacij. V ta namen predstavljamo dva praktična pristopa k analizi mobilnih aplikacij in orodja, ki se uporabljajo v okviru posameznih pristopov. Dodatno opisujemo dve orodji, ki smo jih razvili na podlagi zahtev s katerimi se srečujemo pri analizi mobilnih aplikacij. Prvo uporabljamo pri statični analizi in je namenjeno iskanju vzorcev in ranljivosti v izvorni kodi mobilnih aplikacij. Drugo uvrščamo med orodja za dinamično analizo in se lahko uporablja za dostop in za spreminjanje stanja objektov v času izvajanja aplikacije. Omogoča poljubno izvajanje izvorne kode Java v kontekstu analizirane mobilne aplikacije.*

1. UVOD

Zaradi vse večje razširjenosti mobilnih telefonov [1], preprostega razvojnega procesa in velikega tržnega deleža platforme Android[2], se povpraševanje in uporaba mobilnih aplikacij za platformo Android povečuje. Mobilne aplikacije v vse večjem obsegu razpolagajo z osebnimi podatki posameznikov, zato se upravičeno sprašujemo, če jim lahko zaupamo. Kako in kateri podatki se pošiljajo na strežnik, kje so podatki shranjeni in ali so mobilne aplikacije sploh varne pred zlonamernimi hekerskimi napadi, so vprašanja, ki pridejo na misel. Skladno s vprašanji, se povečuje tudi potreba po analiziranju mobilnih aplikacij. V okviru prispevka bomo predstavili dva praktična pristopa k analizi mobilnih aplikacij na platformi android in dve orodji, ki smo jih razvili z namenom bolj učinkovitega in lažjega testiranja mobilnih aplikacij.

2. MOBILNE APLIKACIJE NA PLATFORMI ANDROID

2.1. Platforma Android

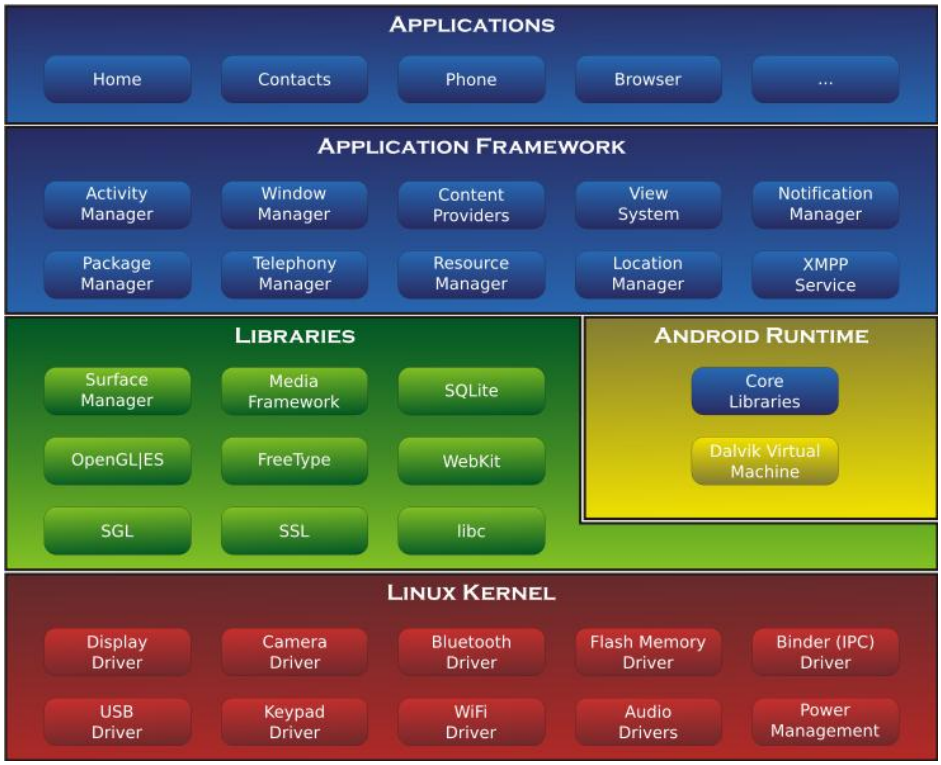
V prispevku obravnavamo Android kot platformo, ki je sestavljena iz operacijskega sistema, vmesnega programja in ključnih mobilnih aplikacij [3]. Android bazira na jedru operacijskega sistema Linux [4]. Aplikacije tečejo v virtualni napravi Dalvik, ki v realnem času izvaja strojno kodo Dalvik. Med strojno kodo Dalvik in strojno kodo Java obstaja preslikava, ki poteka tudi v razvojnem procesu mobilnih aplikacij. Aplikacije pogosto pišemo v izvorni kodi Java, ki se najprej prevede v strojno kodo Java (datoteke s končnico class) in nato transformira v strojno kodo Dalvik (datoteke s končnico dex in odex). Sam razvojni proces in izgradnjo pogojuje transformacija strojne kode kot tudi arhitektura Android, ki je prikazana na sliki 1.

2.2. Struktura mobilnih aplikacij

Čeprav razvoj mobilnih aplikacij poteka v izvorni kodi Java, proces izgradnje poteka v več fazah. Končna datoteka, ki je pripravljena za namestitev, ima končnico apk in običajno vsebuje naslednje mape in datoteke:

- MANIFEST.MF
- CERT.RS
- CERT.SF
- lib
- res
- assests
- AndroidManifest.xml
- classes.dex
- resources.arsc

Z vidika analize mobilnih aplikacij se bomo osredotočili na datoteko AndroidManifest.xml, ki vsebuje zahtevane pravice in definicije uporabljenih gradnikov ter datoteko classes.dex, ki je skupek strojne kode Dalvik (prevedena in transformirana izvorna koda aplikacije).



Slika 1. Arhitektura Android

3. PRAKTIČNI PRISTOPI K ANALIZI MOBILNIH APLIKACIJ NA PLATFORMI ANDROID

V grobem lahko praktične pristope k analizi mobilnih aplikacij razdelimo na dva sklopa: statična analiza in dinamična analiza. Statično analizo obravnavamo v prispevku kot vsi znani pristopi pridobivanja informacij o delovanju v stanju mirovanja aplikacije. Za izvajanje statične analize torej potrebujemo le datoteko s končnico apk. Po drugi strani pa dinamično analizo obravnavamo kot skupek korakov, ki se izvršijo v času

izvajanja aplikacije in vključujejo dostop kot tudi spreminjanje stanja z namenom pridobivanja informacij o delovanju aplikacije.

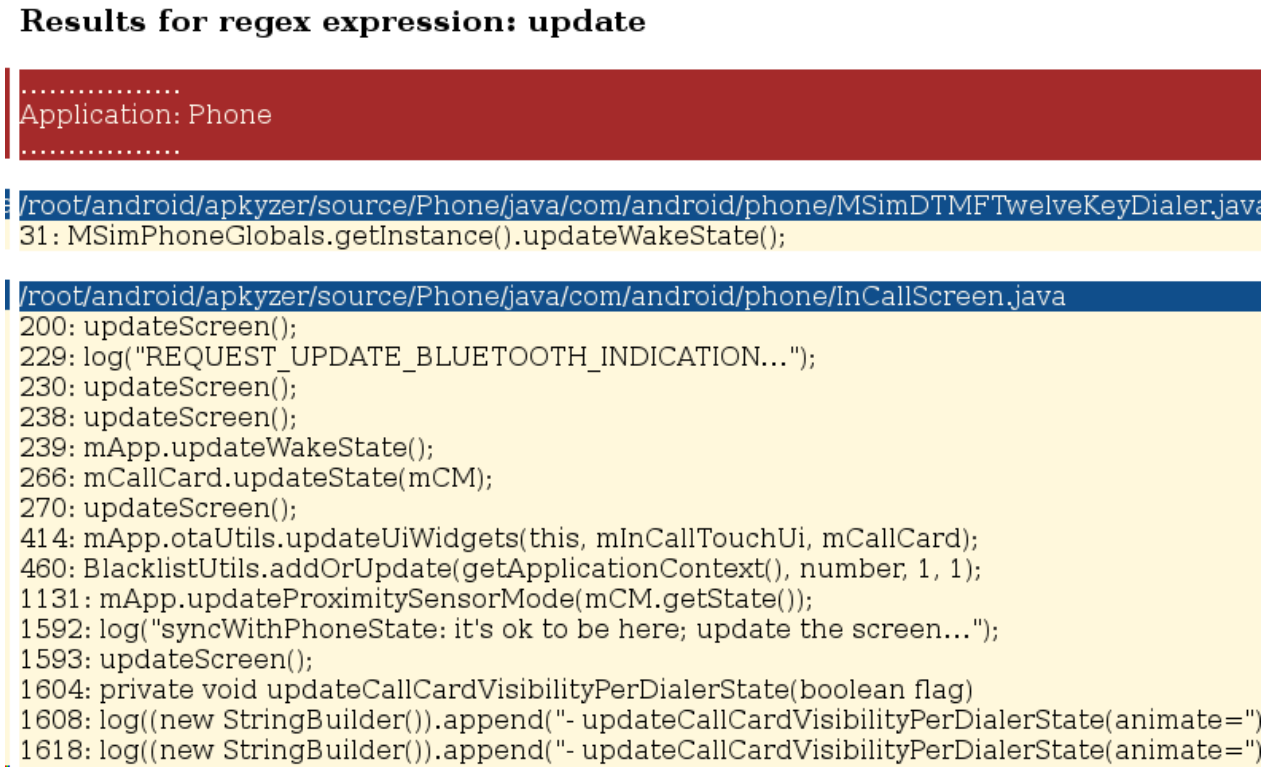
3.1. Statična analiza

Pri statični analizi si prizadevamo doseči oziroma se približati stanju v katerem je bil razvijalec mobilne aplikacije. Poizkušamo pa sicer obrniti proces izgradnje. Običajno izhajamo iz datoteke s končnico apk, ki ni nič drugega kot skupek kompresiranih datotek. Pri tem si lahko pomagamo z različnimi orodji. Pogosto se uporablja orodje apktool [5], s pomočjo katerega pridemo do stanja, kjer se datoteka AndroidManifest.xml in drugi viri v binarni obliki pretvorijo v berljivo obliko in kjer iz datoteke classes.dex dobimo več datotek v zbirniku Smali (zbirnik za strojno kodo Dalvik). Poleg avtomatizacije invertiranja z orodjem apktool lahko pridemo do podobnega končnega stanja. Še preden spreminjamo datoteko apk lahko binarne datoteke preberemo s pomočjo orodja aapt, ki je del ogrodja adt [6]. Nato spremenimo končnico datoteke apk v zip. V naslednjem koraku odpremo kompresirano datoteko in jo razširimo s poljubnim programom za dekompresiranje. Datoteko classes.dex lahko pretvorimo v zbirnik Smali z istoimenskim orodjem smali ozirom baksmali [7]. Če se želimo bolj približati stanju v katerem je bil razvijalec mobilne aplikacije uporabimo orodje dex2jar, ki datoteko classes.dex pretvori v strojno kodo Java s končnico jar. Ko imamo datoteko jar, lahko uporabimo orodje, ki generira izvorno kodo iz strojne kode Java. Primer takšnega orodja je JD-Gui [8]. Pri invertiranju strojne kode Java se lahko podatki izgubijo in dobimo manjkajoče dele izvorne kode, vendar pomanjkljive dele lahko pomensko zapolnimo s poznavanjem zbirnika Smali. Ko pridobimo dovolj informacij o delovanju mobilne aplikacije na podlagi vpogleda v izvorno kodo, lahko preverimo način shranjevanja podatkov. Potrebno je ugotoviti, kje in na kakšen način se shranjuje podatki ter katere aplikacije imajo dostop do njih. Obstaja več različnih načinov shranjevanje podatkov:

- nastavitve v skupni rabi,
- notranji pomnilnik mobilnega telefona,
- zunanja spominska kartica,
- podatkovna baza SQLite,
- spletni strežnik.

Pri shranjevanju podatkov na spletnem strežniku ne moremo zagotovo vedeti načina shranjevanje podatkov, lahko pa preverimo ali se podatki na strežnik pošiljajo po zaščiteni ali po nezaščiteni povezavi. Ostali načini shranjevanja omogočajo, da preverimo kako in kateri podatki se hranijo.

Po pregledu načina upravljanja s podatki sledi analize izvorne kode. V tem koraku poizkušamo identificirati dele izvorne kode z znanimi ranljivosti ali poiskati slabe varnostne prakse kodiranja. Da bi olajšali iskanje teh blokov izvorne kode smo razvili orodje Apkyzer. Apkyzer uporablja orodja dex2jar za pretvorbo strojne kode Dalvik v strojno kodo Java. Nato s pomočjo orodja JAD [9] pretvorimo stojno kodo Java v izvorno kodo. Na koncu preverimo v katerih izvornih datotekah obstaja iskani niz, ki je lahko tudi regularni izraz in rezultate izpiše v datoteko s končnico html. Orodje je primerno za masovno analizo mobilnih aplikacij. Kot primer lahko navedemo ranljivost oddaljenega izvajanja ukazov, ki izvira iz ogrodja za dodajanje oglasov v mobilne aplikacije [10]. Aplikacije, ki jih želimo preveriti prenesemo v eno mapo, zaženemo orodje Apkyzer in pri tem iščemo po nizu »addJavascriptInterface«. Orodje poišče vse aplikacije z nizom v izvorni kodi in prikaže rezultate v html datoteki. Tako na preprost način ugotovimo katere aplikacije vsebujejo ranljivost oddaljenega izvajanja ukazov, ostanke testiranja, uporabniška imena in gesla, ki so jih razvijalci pozabili odstraniti … Primer datoteke z rezultati prikazuje slika 2, kjer smo v izvorni kodi aplikacije Phone iskali niz »update«.



Slika 2. Rezultati analize

Zraven nabora naštetih orodij obstaja še več odprtokodnih rešitev, ki približajo stanje v katerem je bil razvijalec mobilne aplikacije pred izgradnjo. V tem stanju imamo vpogled v izvorno kodo, kar omogoča da preverimo kaj počne aplikacija in kako to počne. Delo nam lahko oteži obfuskacija izvorne kode. V tem primeru si pomagamo z orodji za deobfuskacijo in z dinamično analizo.

3.2. Dinamična analiza

Za celovito analizo mobilne aplikacije statična analiza ne zadošča oziroma lahko traja predolgo. Zato si pomagamo z dinamično analizo, kjer preverjamo in poizkušamo vplivati na stanje aplikacije v času izvajanja. Pri dinamični analizi si lahko pomagamo z več orodji in pristopi.

Prvi pristop je prestrezanje komunikacije. V ta namen najprej uporabimo posredniški strežnik, ki posluša na poljubnih vratih. Pred tem na mobilni telefon namestimo digitalno potrdilo posredniškega strežnika. V nasprotnem primeru mobilni telefon ne bo obravnaval posredniškega strežnika kot zaupanje vrednega. Na koncu še nastavimo povezavo v brezžično omrežje tako, da dodamo naslov in vrata posredniškega strežnika, ki smo ga postavili. Opisane nastavitve zadostujejo za prestrezanje komunikacije pri večini mobilnih aplikacij.

Naslednji pristop, ki omogoča celovitejšo analizo in je hkrati tudi bolj radikalen, saj posega v izvorno kodo aplikacije, je namestitev stranskih vrat. Stranska vrata omogočajo kasnejšo povezavo in dostop do stanja mobilne aplikacije. K temu pristopu lahko uvrščamo orodje Fino [11], ki ga sestavlja več komponent. V grobem so sestavni deli orodja skripta, ki skrbi za namestitev stranskih vrat, odjemalec, ki je napisan v programskem jeziku Python in omogoča povezavo do stranskih vrat ter pošiljanje in sprejemanje ukazov in samostojna mobilna aplikacija Gadget, ki predstavlja most med odjemalcem in ciljno mobilno aplikacijo. Skripta za namestitev stranskih vrat v mobilno aplikacijo namesti storitev, ki je izpostavljena in tako dosegljiva ostalim mobilnim aplikacijam. To izkorišča Gadget, ki najprej prejme povezavo od odjemalca in se nato poveže na storitev. Vsa nadaljnja komunikacija poteka skozi Gadget, ki ga lahko v tem primeru

enačimo s posredniškim strežnikom. Fino oziroma storitev, ki je vrinjena v ciljni mobilni aplikaciji, uporablja Java Reflection [12], in sicer za dostop do spremenljivk in metod. Tako je možno dostopati do vseh vrst spremenljivk ne glede na tip (private, public, protected). Prav tako je možno spreminjati stanje spremenljivk in klicati metode posameznih objektov. Na takšen način lahko testiramo kako se obnaša mobilna aplikacija pri vnosu robnih pogojev, ali se vhodni podatki preverjajo na strežniku in kaj se dogaja pri izvajanju posamezne metode.

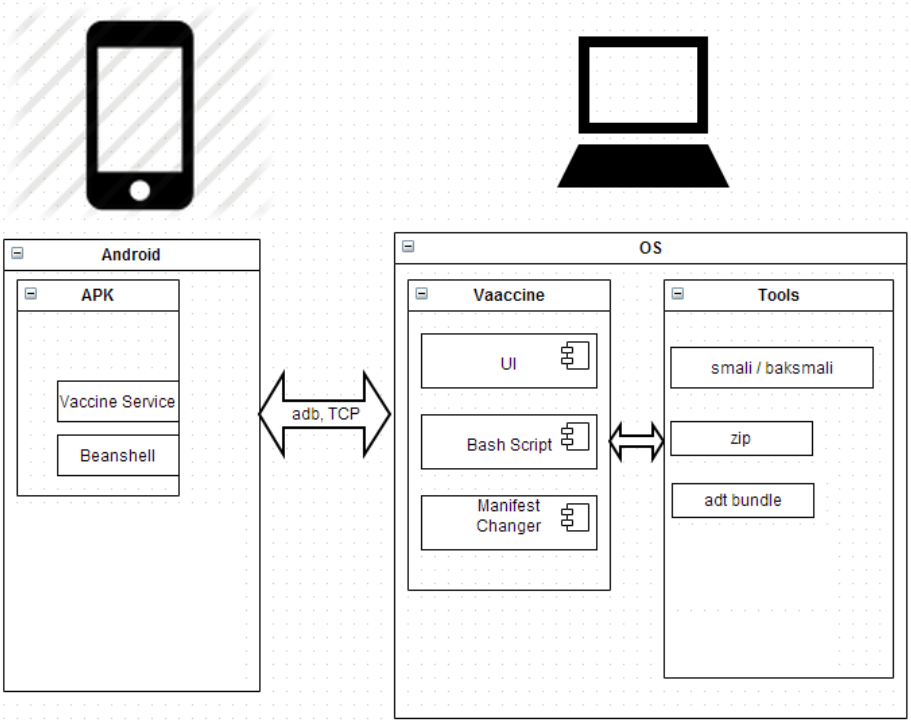
Fino je koristno orodje za dinamično analizo mobilnih aplikacij, vendar smo pri uporabi pogrešali grafični uporabniški vmesnik in manj omejitev pri izvajanju izvorne kode. Prav tako smo mnenja, da lahko različne tehnologije, ki jih uporablja Fino, v nekaterih primerih otežijo iskanje napak. Zato smo se odločili za razvoj orodja, ki vsebuje opisane prednosti in vključuje elemente, ki smo jih pogrešali pri orodju Fino.

3.2.1. Dinamična analiza z Vaccine

Vaccine je orodje za dinamično analizo mobilnih aplikacij, ki v mobilno aplikacijo vrine storitev, aplikacijo namesti na mobilni telefon, zažene storitev in se nanjo poveže. Ko je povezava vzpostavljena se prikaže grafični vmesnik, ki ga obravnavano kot vstopno točko za dinamično analizo z Vaccine. Pred prikazom grafičnega vmesnika Vaccine mobilno aplikacijo pripravi. Pri tem se izvedejo naslednji koraki:

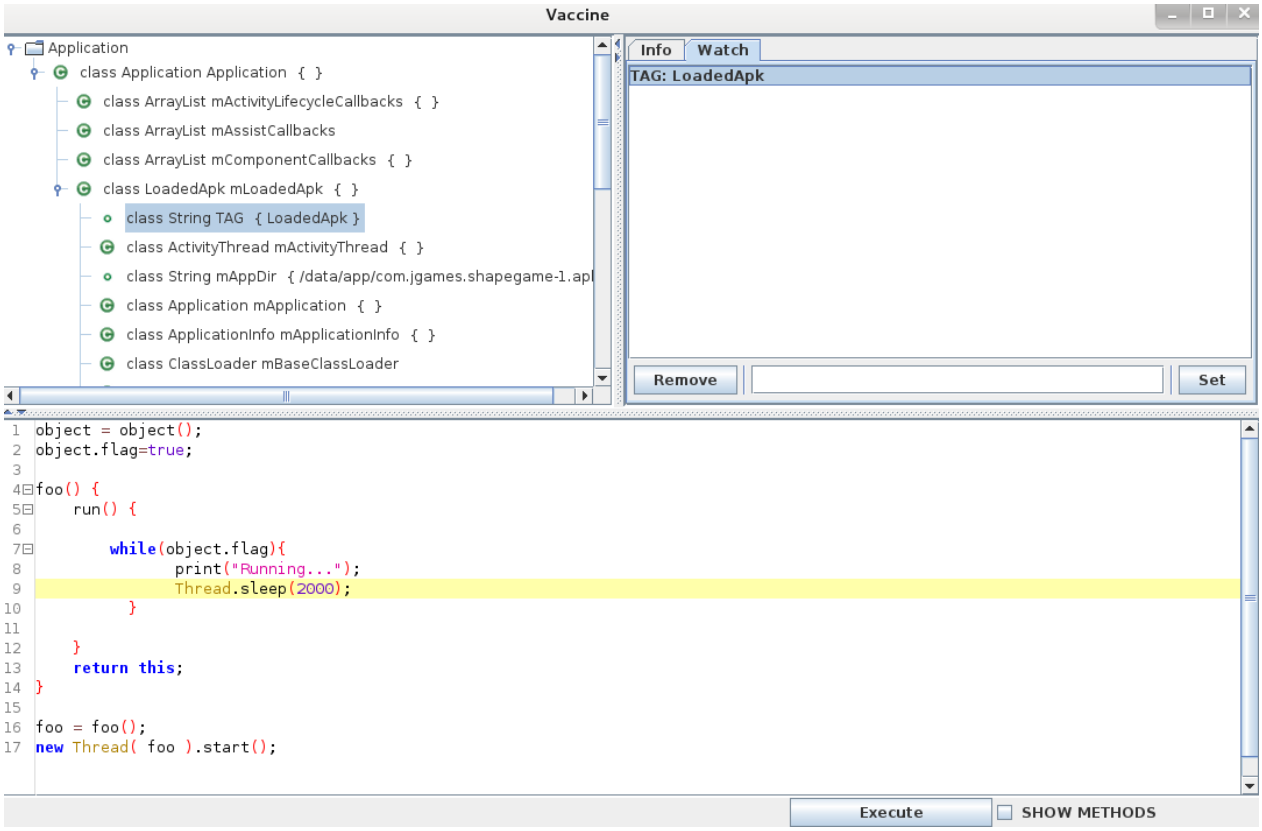
- razširitev mobilne aplikacije,
- pridobitev smali kode iz datoteke classes.dex,
- vrivanje storitve in dodatnih knjižnic,
- prevajanje smali kode v datoteko classes.dex,
- zamenjava datotek classes.dex in AndroidManifest.xml,
- odstranjevanje digitalnega podpisa,
- podpisovanje mobilne aplikacije,
- namestitev mobilne aplikacije,
- zagon vrinjene storitve,
- povezava do mobilne aplikacije in zagon grafičnega uporabniškega vmesnika.

Naštete korake izvede skripta, ki je ena izmed komponent orodja Vaccine. Arhitekturo orodja z okoljem, ki je potrebno za dinamično analizo z Vaccine, prikazuje slika 3.



Slika 3: Okolje Vaccine

Komunikacija med mobilno aplikacijo in operacijskim sistemom na računalniku poteka preko protokola TCP s pomočjo orodja adb, ki je del ogrodja adt [6]. Poleg vrinjene storitve, Vaccine skripta vrine še knjižnico Beanshell, ki omogoča izvajanje izvorne kode Java v realnem času brez predhodnega prevajanja. Beanshell je interpreter programskega jezika Java in je tudi v njem napisan [13]. Grafični vmesnik Vaccine kot tudi komponenta za spreminjanje datoteke AndroidManifest.xml sta prav tako zapisana v programskem jeziku Java. Tako smo poizkusili čim bolj poenotiti tehnologije, ki so v interakciji z mobilno aplikacijo. Grafični vmesnik smo razdelili na tri glavne dele. Levi zgornji del je hierarhično urejena drevesna struktura z objekti, ki jih aplikacija uporablja v času izvajanja. Korensko vozlišče je umetno dodano in predstavlja mobilno aplikacijo, ki jo trenutno analiziramo. En nivo pod korenskim vozliščem so aktivnosti, ki se dodajo in odstranjujejo avtomatično in skladno z življenjskim ciklom Android aktivnosti [14]. Desni zgornji del je sestavljen iz dveh pogledov. Prvi se imenuje »Info« in prikazuje osnovne informacije o izbranem objektu v drevesni strukturi objektov. V drugem pogledu, ki se imenuje »Watch«, se prikazujejo vrednosti dodanih objektov. Prikaz se periodično osvežuje v regularnih časovnih intervalih. S pomočjo pogleda »Watch« je tudi možno nastavljati vrednosti dodanih objektov. Tretji del grafičnega vmesnika je namenjen pisanju skript. Omogoča izvajanje kode Beanshell kot tudi izvorne kode Java. Slika 4 prikazuje primer izvajanje kode Beanshell, ki najprej ustvari nov objekt, mu dodeli zastavico in jo nastavi na vrednost »true« ter nato v ločeni niti vsake dve sekundi izpiše niz »Running …«. Nit se izvaja dokler je vrednost zastavice na »true«. Drevesna struktura objektov (levi zgornji del grafičnega vmesnika) se odziva na štiri vrste akcij: levi klik, dvojni klik, desni klik in kombinacija CRL+desni klik. Z levim klikom izberemo objekt, njegovo stanje pa se izpiše v pogledu »Info«. Desni klik na objekt v drevesni strukturi Vaccine zazna kot ukaz za poizvedbo, ki vrne vsa polja izbranega objekta ne glede na tip. Polja, ki vključujejo vse definirane spremenljivke (ne vključuje lokalnih) in metode razreda na podlagi katerega je bil objekt inicializiran, se dodajo pod izbrani objekt v drevesni strukturi. Dvojni klik objekta v drevesni strukturi povzroči, da se njegova referenca prenese v skriptni del Vaccine (spodnji del grafičnega vmesnika). Na podlagi reference, ki se je prenesla v skriptni del se lahko sklicujemo in uporabljamo objekte v kodi Beanshell.



Slika 4: Grafični vmesnik Vaccine

Vaccine omogoča izvajanje poljubne izvorne kode Java oziroma Beanshell, dostop do stanja spremenljivk in njihovega spreminjanja ter izvajanje definiranih metod. Poljubno izvajanje kode ni omejeno na metode, spremenljivke in razrede mobilne aplikacije. Uporabljamo lahko poljubne knjižnice ogrodja Android, vendar se moramo zavedati, da bo izvedena koda tekla v kontekstu mobilne aplikacije v ločeni niti, kjer teče vrinjena storitev. Razširljivost Vaccine smo zagotovili s pomočjo poljubnega dodajanje kode Beanshell v obliki skriptnih datotek. Pripravljene skritpe Beanshell se pred zagonom Vaccine skripte dodajo v ustrezno mapo in se avtomatično vključijo v mobilno aplikacijo. Kasneje jih lahko kličemo in uporabljamo v skriptnem delu grafičnega vmesnika Vaccine.

3. ZAKLJUČEK

V prispevku smo predstavili dva različna pristopa analize mobilnih aplikacij ter pri vsakem pristopu nekaj prosto dostopnih orodij, ki do določene mere avtomatizirajo in olajšajo postopke analize. Pri vsakem pristopu smo prav tako predstavili orodji, ki smo jih razvili in prilagodili zahtevam in izzivom s katerimi se srečujemo pri našem delu. Menimo, da je za celovito in učinkovito analizo mobilni aplikacij potrebno orodja in pristope kombinirati, saj le tako lahko dosežemo maksimalen učinek analize mobilnih aplikacij.

4. LITERATURA

[1] SCHWEIGHOFER Tina, KOCBEK Mateja , KOŠIČ Kristjan "Celovito testiranje mobilnih aplikacij", Sodobne tehnologije in storitve OTS 2011: Zbornik osemnajste konference, Maribor, Fakulteta za elektrotehniko, računalništvo in informatiko, Inštitut za informatiko, 50-59.

[2] http://en.wikipedia.org/wiki/Smartphone#Market_share, (Smartphone), obiskano 27.3.2014

[3] http://www.openhandsetalliance.com/android_overview.html, (Open handset alliance), obiskano 26.3.2014

[4] http://en.wikipedia.org/wiki/Android_(operating_system), (Android operating system), obiskano 26.3.2014
[5] https://code.google.com/p/android-apktool/ , (android-apktool), obiskano 26.3.2014
[6] http://developer.android.com/tools/help/adt.html, (Android Developer Tools), obiskano 26.3.2014
[7] https://code.google.com/p/smali/, (smali), obiskano 26.3.2014
[8] http://jd.benow.ca/, (JD-Gui), obiskano 26.3.2014
[9] http://en.wikipedia.org/wiki/JAD_(JAva_Decompiler), (Java Decompiler), obiskano 26.3.2014
[10] https://labs.mwrinfosecurity.com/blog/2013/09/24/webview-addjavascriptinterface-remote-code-execution/, (WebView addJavascriptInterface Remote Code Execution), obiskano 26.3.2014
[11] https://github.com/sysdream/fino, (Sysdream/Fino), obiskano 27.3.2014
[12] http://docs.oracle.com/javase/tutorial/reflect/index.html, (The Java Tutorials), obiskano 27.3.2014
[13] http://www.beanshell.org/home.html, (Beanshell), obiskano 27.3.2014
[14] http://developer.android.com/reference/android/app/Activity.html, (Activity), obiskano 27.3.2014