TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika eriala

Karl Lääts

Sõnasageduste põhine logianalüüs

Bakalaureusetöö (9 EAP)

Juhendaja: Meelis Roos

Tartu 2016

Sõnasageduste põhine logianalüüs

Lühikokkuvõte:

Käesoleva bakalaureusetöö eesmärk on välja selgitada, kas sõnasageduste analüüsi saab kasutada logifailide uurimiseks, et leida sealt huvitavaid sündmusi, ilma eelteadmisteta logi struktuuri kohta.

Võtmesõnad:

Sagedustabel, logifail, logi, statistiline analüüs, sõnasagedus, logi analüüs

CERCS:

P175 Informaatika

Word frequency based log analysis

Abstract:

The purpose of this bachelor thesis is to explore if you can use word frequency based analysis for log files and find interesting events without knowing the log structure.

Keywords:

Frequency table, log file, log, statistic analysis, word frequency, log analysis

CERCS:

P175 Informatics

Sisukord

Sissejuhatus

Tänapäeva digiühiskonnas on arvutid ja serverid muutunud kergesti kättesaadavaks: masinaid on palju, hinnad on odavad ja kasutamine on muutunud lihtsamaks. Igast tegevusest arvutis jääb aga jälg logifailide näol. Logid on arvutite ja serverite tegevuste päevikud, mida kasutatakse nii statistilistel eesmärkidel kui ka varundamise ja taaste jaoks [1]. Kuna tekib üha rohkem logifaile, siis kasvab ka vajadus neid täpsemalt ja põhjalikumalt analüüsida ning uurida ning sellega siinne uurimus ka tegeleb.

Operatsioonisüsteem ja teised juhtprogrammid kirjutavad kõvakettale logifailid näiteks sissetulevate dialoogide, vea- ja olekuteadete ning teatud tehingudetailide jälgimise võimaldamiseks. Võidakse kirja panna ka rutiintööde käivitumis- ja peatumisajad. Logifaili võib genereerida mistahes programm. Rakendus võib genereerida logi, millele kasutaja võib vajadusel viidata või mis võib olla abiks arvuti kokkujooksmise uurimisel. Näiteks FTP-d (File Transfer Protocol) kasutav programm võib genereerida logifaili, mis näitab iga edastatud faili edastamise kuupäeva, kellaaega, allikat ja faili täisnime [1].

On olemas juba mitmeid logide töötlemise ja analüüsimise tööriistu. Lihtsamad logianalüsaatorid põhinevad tuntud või etteantud mustrite otsimisel ja nende analüüsimisel. Uuemad ja efektiivsemad kasutavad tehisintellekti meetodeid. Need aga vajavad lisateadmisi ja on kasulikud ainult spetsialistidele. Paljudel väikefirmadel ja tavakasutajatel pole selliste programmide kasutamiseks piisavalt teadmisi või aega, et need selgeks õppida. Eraldi logianalüütiku palkamine võib olla väga kulukas. Oleks vaja lihtsamalt ja tõhusamalt töötavat programmi.

Käesoleva töö eesmärk on leida struktureerimata logidest kirjeid, mille iseloomu ega mustreid pole võimalik ette teada, aga mis erialaspetsialisti poolt peale vaadates tunduvad kasulikud. Püstitame hüpoteesi, et sõnasagedustel põhineva logi analüüsi ja lihtsate statistiliste meetoditega saab leida tundmatust logist huvitavaid sündmuseid.

Töö esimeses peatükis sõnastatakse probleemi püstitus. Teises peatükis on kirjeldatud probleemi etapiviisilist lahendamist. Kolmandas peatükis on programmi kirjeldus. Neljandas peatükis on analüüsi tulemused ja järeldused.

1. Probleemi püstitus

Siin kirjeldame, et missuguste logidega tegeleme ja kuidas me otsime huvitavaid sündmuseid statistiliste meetoditega. Suurem eesmärk on leida sündmusi, mis on huvitavad erialaspetsialistile, näiteks tulemüüri administraatorile.

1.1 Logifail

Võtame uurimise alla tekstifaili formaadis logid. Logidel pole määratud piire, et kui suured nad olla saavad. Seetõttu võivad logide suurused kasvada kiiresti: mida rohkem logikirjeid tehakse, seda rohkem ruumi võtab masinas. Võib tekkida olukordi, kus logifail ei mahu tervenisti põhimällu ära ning tuleb kasutada spetsiaalseid programme. Windowsil saab kuni ~2 GB suuruseid logifaile avada programmiga Notepad++. Suuremate failide puhul saab kasutada LogExperti.

Programmid salvestavad logisid erinevates formaatides. Üks levinumatest formaatidest on syslog. Seda seepärast, et paljud programmid ja süsteemsed teenused kasutavad Unix-tüüpi operatsioonisüsteemides oma logiteateid just syslog kaudu.

Syslog on laialt levinud standardteadete logimiseks. Logimiseks nimetatakse kirjete tegemist logifaili [1]. Syslogi kasutatakse põhiliselt süsteemi haldamiseks, turvalisuse kontrollimiseks, üldiseks infoks, analüüsimiseks ja silumiseks. Syslogi formaat algab kuupäevaga, mis on kujul “Mmm dd hh:mm:ss” (ilma jutumärkideta). “Mmm” on ingliskeelne lühend kuust, millel esimene täht on suur ja teised väiksed. Sellele järgneb tühikuga eraldatud kuupäev “dd”. Kui see on väiksem, kui 10, siis lisatakse juurde üks tühik, et joondus oleks sama. Neile järgneb tühikuga eraldatult aja määramine “hh:mm:ss”, millel “hh” näitab tunde vahemikus 00-23 (kaasa arvatud) ja “mm:ss” näitavad vastavalt minuteid ja sekundeid vahemikus 00-59 (kaasa arvatud). Näiteks “Aug 24 10:23:55” on õige syslogi ajatempel. Ajatemplile järgneb sõnumi osa, mis koosneb programmi või protsessi andmetest, mis selle logi kirje tekitasid ning sündmuse kirjeldusest. Iga kirje on eraldi real [2]. Tihti peale lisatakse syslogi sõnumisse ka aeg, millal see sündmus toimus. Seda seetõttu, et kirje tegemise ja tegeliku sündmuse vahel võib olla väikene ajavahe ja täpsema aja kirjapanekuks on vajalik aeg sõnumisse kirja panna. Samuti võimaldab see lisada ka aasta, mida ametlik syslogi ajaformaat ei toeta.

```
Feb 21 06:24:38 kosmos sshd[7977]: Invalid user achikwenya from 211.49.99.14
Feb 21 06:24:38 kosmos sshd[7977]: input_userauth_request: invalid user achikwenya [preauth]
Feb 21 06:24:39 kosmos sshd[7977]: pam_unix(sshd:auth): check pass; user unknown
Feb 21 06:24:39 kosmos sshd[7977]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0
tty=ssh ruser= rhost=211.49.99.14
Feb 21 06:24:41 kosmos sshd[7977]: Failed password for invalid user achikwenya from 211.49.99.14 port
44218 ssh2
Feb 21 06:24:41 kosmos sshd[7977]: Received disconnect from 211.49.99.14: 11: Bye Bye [preauth]
```

Joonis 1. Näide auth.log.4 failist.

Joonisel 1 on näha auth.log.4 logifaili väga väikest osa, mis sisaldab korrektset syslogi formaati. Auth logi fail sisaldab infot autentimiste kohta. Jooniselt on näha, et logi kirjed on tehtud 21-se veebruari hommikul kell 6.24. Pärast ajatemplit näeme, et need logikirjed on tehtud masinast nimega kosmos. Sellele järgneb teenuse nimetus ja PID (protsessi ID), mis on antud näites sshd[7977]. Järgmiseks tuleb sõnum, mis kirjeldab sündmust. Nende 6 rea pealt on näha, et IP 211.49.99.14 pealt on üritatud SSH kaudu sisse logida masinasse kosmos. Lähemal uurimisel selgub, et see IP-aadress asub Lõuna-Koreas.

On olemas mustrilised analüsaatorid, mis analüüsivad logifailide tuntud mustreid. Sellised analüsaatorid leiavad ülesse põhilised ja levinud sündmused, kuid logifailide unikaalsed sündmused jäävad leidmata. Selle vastu aitab mingil määral ise nende mustrite sisestamine,

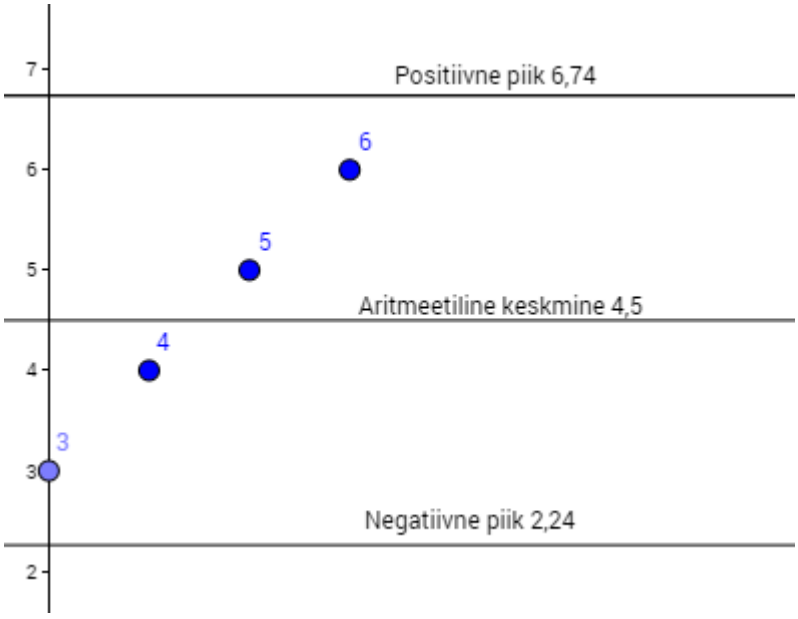
mida soovitakse leida, kuid see eeldab teadmisi sellest, millised on antud logifailid ja mida kindlalt teada tahetakse. Tundmatud sündmused jäävad ikkagi leidmata. Selle lõputöö üheks eesmärgiks ongi leida struktureerimata logifailidest huvitavaid sündmuseid, mille kohta ei ole ette antud mingeid mustreid. Esialgu toetame ainult syslog formaati, sest sellest piisab meetodi toimivusest aru saamiseks.

1.2 Piik

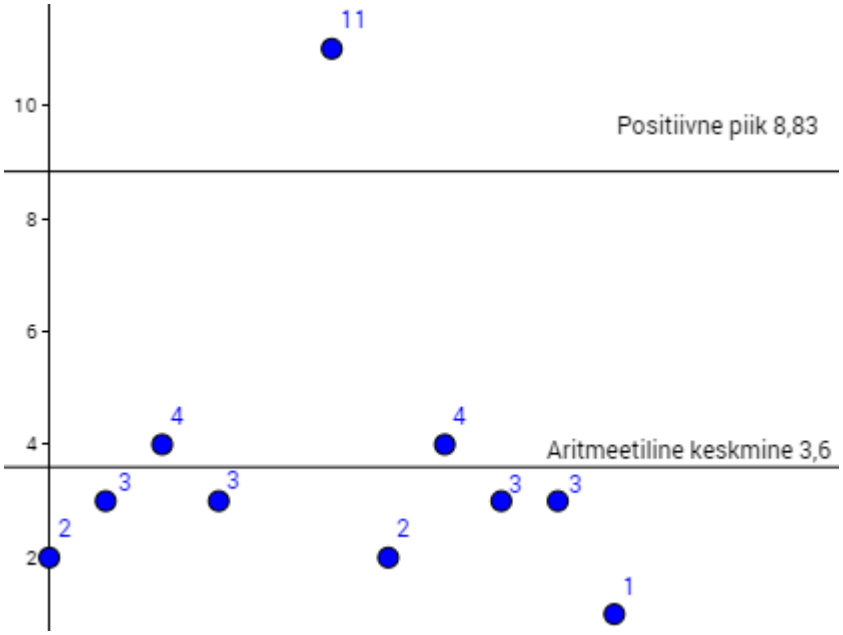
"Huvitavateks sündmusteks" on selle lõputöö raames valitud piigid. Piikide defineerimisel on kasutatud võimalikult lihtsaid matemaatilisi mõisteid. Selleks, et piike leida, loendatakse logist iga konkreetse sõna esinemiste arv vaadeldavas ajavahemikus. Piik on defineeritud kui sündmus, mille puhul sõna esinemiskordade arv erineb rohkem kui kahe standardhälbe võrra sama sõna keskmisest esinemiskordade arvust ajaühikus. Kaks standardhälvet sai läveks valitud sellepärast, et nii palju aritmeetilisest keskmisest erinemine võiks olla erandlik sündmus.

Matemaatilisteks terminiteks on siin aritmeetiline keskmine ja standardhälve. Aritmeetiline keskmine on arv, mis saadakse antud arvude summa jagamisel liidetavate arvuga. Näiteks arvud: 3,4,5 ja 6. Nende arvude summa on 18. Jagame summa liidetavate arvuga, mis on 4. Saame tulemuseks, et aritmeetiline keskmine on 18/4=4,5 [3]. Standardhälve on ruutjuur dispersioonist. Dispersioon on hälvete ruutude aritmeetiline keskmine. Hälve on üksiku väärtuse erinevus aritmeetilisest keskmisest [4]. Ehk tulles tagasi eelmise näite juurde, siis saame hälvete ruutudeks: $(3\text{-}4{,}5)^2$ = 2,25, $(4\text{-}4{,}5)^2$=0,25, $(5\text{-}4{,}5)^2$=0,25, $(6\text{-}4{,}5)^2$=2,25. Dispersiooniks saame sellisel juhul (2,25+0,25+0,25+2,25)/4 = 5/4 = 1,25. Standardhälbeks on ruutjuur 1,25-st, mis on ümardatult 1,12. Kuna piigi definitsioonis on vaja kahekordset standardhälvet, siis korrutame standardhälbe kahega läbi. Saame tulemuseks 2*1,12 = 2,24.

Piike saab olla nii positiivseid kui ka negatiivseid. Me tunneme rohkem huvi positiivsete piikide vastu, sest neist on võimalik rohkem infot kätte saada. Negatiivsed piigid ei paku huvi, sest praktikas tähendab see mingi sündmuse mittetoimumist ja sellest on raske midagi sisulist välja lugeda. Positiivseks piigiks peab olema väärtus üle 4,5+2,24 = 6,74 ja negatiivseks piigiks peab väärtus olema alla 4,5-2,24=2,26. Meie arvudest ei vasta ükski nendele nõuetele ja nagu näha ka jooniselt 2, siis ei leidu punkti, mis asuks piikide piirist üleval või allpool. Seega meie näite arvude jadas puuduvad piigid. Visuaalset kujutust on näha joonisel 2.



Joonis 2. Näite arvude visuaalne kujutus.



Joonis 3. Arvude visuaalne kujutus, mis sisaldab positiivset piiki.

Joonisel 3 on näha arvude 2, 3, 4, 3, 11, 2, 4, 3, 3, 1 visuaalset kujutust. Nende arvude aritmeetiline keskmine on 3,6 ning standardhälve on 2,615. Seega on positiivseks piigiks 8,83 ning negatiivseks -1,63. Arv 11 on positiivne piik, sest asub positiivse piigi piirist üleval pool. Neil arvudel puudub negatiivne piik. Samuti ei saa sõnade arv olla kunagi negatiivne.

2. Tegevusetapid

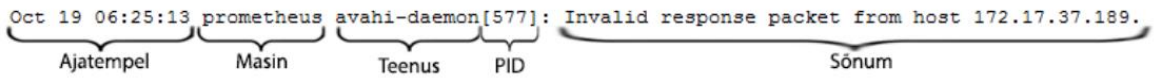
Ei olnud teada, kuidas oleks mõistlik sõnasagedusel põhinevat logi analüüs teha, siis valiti metoodika väljatöötamiseks iteratiivne lähenemine. Hüpoteeside kontrolliks kirjutati programm logAnalyser. Igas tegevusetapis analüüsiti, mis senistest meetoditest on kasulik ja kuidas seda täiendada ning mis on kasutu, et seda välja jätta. Vastavalt sellele täiendati samm-sammult programmi. Etapid kestsid tavaliselt nädala kuni paar ja iga etapi lõpus toimus arutelu.

2.1 I Etapp

Esimene etapp oli välja selgitada, milliste meetoditega oleks võimalik püstitatud eesmärki täita. Esmalt tutvuti erinevat tüüpi logifailidega, mille järel valiti syslogi logid, sest need on ühed kõige levinumad. Järgmisena oli vaja välja selgitada, milliseid sarnaseid programme logifailide analüüsimiseks juba olemas on ja rakendada sõnade lugemist logifailidest. See tähendab, et tuli teha sõnade sagedustabelid 5 minuti, 1 tunni ja 24 tunni kaupa. Need ajavahemikud valiti, sest päeva granulaarsust oli vaja pikemaajalise (vähemalt nädalase) analüüsi jaoks, tunni granulaarsust mõne päevaste vahemike täpsemaks uurimiseks ning 5-minutilist detailsema uurimise jaoks.

Otsustati, et algselt sõelutakse logifailidest välja ajad ja PID-d, et neid sõnade lugemisel mitte arvestada, sest tekitavad liiga palju müra. Ajatempli ja PID asukohta on näha joonisel 4. Edasi defineeriti statistilise poole pealt piigi olemus ja arvutati aritmeetiline keskmine ja standardhälve.

Analüüsiks võeti reaalsete serverite logisid, mis olid syslog formaadis. Katsetusi tehti kõigiga, kuid edaspidiseks uurimiseks valiti üks konkreetne autentimislogi auth.log.4, millega testiti kõiki täiendusi.



Joonis 4. Syslogi formaat.

2.2 II Etapp

Teiseks etapiks oli valminud esimene prototüüp programmist. See sisaldas logidest sõnade lugemist ja päevade kaupa sõnade sagedustabelite faili kirjutamist. Selleks ajaks töötas ainult päevade kaupa sagedustabelite tegemine. 5 minuti ja 1 tunni kaupa sõnade lugemine jäeti algselt välja, et saaks programmi põhifunktsionaalsust arendada ja testida.

```
Feb 14     6     sshd[9128]
Feb 14     7     sshd[23746]
Feb 14     6     jrwang
Feb 14     7     sshd[21216]
Feb 14     7     sshd[22547]
Feb 14     1701  BREAK-IN
```

Joonis 5. Auth.log.4 sagedustabel.

Joonisel 5 on näha paar rida tulemusfailist, mis tekkisid auth.log.4 failist. Programmi tulemusfail sisaldas sõnade esinemisarvu iga päeva kohta. Esimeselt realt saadi teada, et 14. veebruaril esines sõna *sshd[9128]* 6 korda ja 6. realt nähti, et sõna *BREAK-IN* esines 1701 korda. Sõnad loeti kokku masina nimedest, teenuse nimedest ja sõnumi osadest. Sõnumi osa

tükeldati tühiku kohast, et saada kätte sõnad. Esimesel etapil jäeti teenuse nimetused sõnade lugemisest välja, kuid jooniselt on näha, et esinesid ka teenuse nimetused. Järgmiseks etapiks oli näha, et see viga tuli ära parandada, sest lisas liiga palju müra sagedustabelitesse.

Samuti selgusid muudatused, mis aitasid saada paremat tulemust. Näiteks leiti, et igas päevas esines sõnu, mis olid tühjad stringid. See tundus kahtlane, kuna kõik sõnad eraldati tühiku koha pealt. Auth.log.4 lähemal uurimisel selgus, et seal leidus sõnumeid, kus on mitu tühikut järjest ning tühikutega poolitamisel sõnade jaoks lõi nende vahele tühjad stringid. Seega tuli järgmiseks etapiks eemaldada tühjad sõnad, mis tekkisid olukorras, kui oli mitu tühikut järjest.

Kuupäevale lisati ka kellaajad sõnasageduse tabelitesse, sest see annab täpsema ülevaate. Viie minuti ja ühe tunni jaoks on seda samuti vaja, sest siis on võimalik näha, et mis ajavahemikku antud sagedustabel kuulub.

Järgmise etapi eesmärgiks valiti positiivsete ja negatiivsete piikide arvutamine.

2.3 III Etapp

Kolmandaks etapiks oli tehtud positiivsete ja negatiivsete piikide arvutamine ja kuvamine 24 tunnistes vahemikes.

Tabel 1. Auth.log.4 negatiivsed piigid.

Kuupäev	Sõna	Esinemiste arv	Hälve	Aritmeetiline keskmine
Feb 15	55304	0	0.707	1.5
Feb 16	55651	0	0.331	0.875
Feb 17	55701	0	0.927	1.875
Feb 21	51312	0	0.331	0.875
Feb 21	pam_unix(cron:session):	15	11.758	44.0

Tabel 1 sisaldab paari auth.log.4 negatiivset piiki ja neid analüüsides on näha, et tegemist on 24-tunniste ajavahemikega, sest on esitatud ainult kuupäevad ja kellaajad puuduvad. Samuti on näha, et enamus negatiivseid piike on siis, kui sõna esines 0 korda. See tähendab, et negatiivne piik tekib siis, kui sõna antud päeval ei esine. Negatiivsete piikide loendis oli põhiline osa piikidest tekkinud siis, kui sõna mingil päeval ei esinenud. Kuna sellised piigid ei anna uurimiseks mingit infot, siis jäeti need piikide loetelust välja.

Tabel 2. Auth.log.4 positiivsed piigid.

Kuupäev	Sõna	Esinemiste arv	Hälve	Aritmeetiline keskmine
Feb 16	gqliu	6	1.984	0.750
Feb 20	202.120.62.137	1248	409.013	186.0
Feb 15	fzliu	6	1.984	0.750
Feb 21	mtrnc	18	6.666	3.750
Feb 17	4522	6	1.984	0.750

Tabel 2 sisaldab paari auth.log.4 positiivset piiki. Piike analüüsides on näha, et tegemist on 24-tunniste ajavahemikega. Samuti on ka näha, et piigid pole ajalises järjekorras. See on tingitud Java paisktabelite (ingl.k. *HashMap*) olemusest. Veel on võimalik jooniselt välja lugeda seda, et leidub sõnu, mille keskmine väärtus on väga väike ja need piigid kannavad vähe informatsiooni. Seda seepärast, et nad tekivad siis, kui esinevad ainult ühe päeva jooksul. Näiteks nädala jooksul on üks päev, kui esineb sõna *gqliu* 6 korda ja teistel päevadel 0 korda, siis tekibki sellest piik. Põhjalikumalt uurides on näha, et selliseid piike on väga palju. Seda seetõttu, et auth.log.4 fail sisaldab ka ebaõnnestunud sisselogimisi ja kui näiteks Lõuna-Koreast üritatakse SSH-sse sisse murda, siis proovitakse järjest igasuguseid aasiapäraseid nimesid, mistõttu tekib ühe päeva peale väga palju piike kasutajanimedest, mis ei anna analüüsimisele midagi juurde. Sõelumiseks lisati reegel, et piikideks loetakse ainult neid, mille keskmine on suurem kui 2,0. See vähendab müra ja neid sõnu, mis esinevad vaid ühel päeval paar korda.

Viimasena eemaldati selles etapis logikirjete parsimisel masinate nimetused, sest need tekitavad ülearust müra, mida pole piikide arvutamisel vaja. Lisaks tühikule lisati sõna eraldajaks ka koolon, võrdusmärk, ülakoma ja jutumärk. See võimaldab saada spetsiifilisemaid sõnu ning vähem müra piikide hulka.

Järgmiseks etapiks lepiti kokku, et parandatakse teada olevad vead.

2.4 IV Etapp

Neljandas etapis parandati palju puuke (ingl.k. *bug*), mis olid eelnevatest kordadest sisse jäänud ja mis oli vahepeal juurde tekkinud või leitud. Logikirjetest eemaldati kellaaeg, masina nimetus ja PID sõnade kokku lugemisel. Sõnumi osa tükeldati tühiku, kooloni, võrdusmärgi, ülakoma ja jutumärgi juurest. Selle tulemusena vähendati müra ja aidati saada täpsemaid piike. Piikidele lisati juurde reegel, et keskmine peab olema suurem kui 2. See vähendas piikide arvu, mis on väga väiksed ja ei mängi analüüsimisel rolli. Negatiivsetel piikidel eemaldati 0 countiga piigid. Üle-eelmisest etapist sai ära parandatud puuk, mille puhul tekkis mitme järjestikuse tühiku puhul sagedusloendisse tühi sõne.

Antud etapil katsetati programmi uue tulemüüri logiga, mis oli väga suure mahuga. Eesmärk oli testida, kas programm suudab arvutis joosta nii, et mälu ei saaks täis. Logi sisaldas 6,7 miljonit logikirjet, mis olid ühe nädala jooksul tehtud. Faili suuruseks oli 2,3GB ning seda tuli lugemiseks avada LogExpert-ga. Masinas, mille peal programmi katsetati, oli 8 GB mälu ning programm sai logi töötlemisega ilusti hakkama. Probleeme ei tekkinud.

Probleem tekkis aga sellega, et tulemüüri logifaili ajatempel sisaldas kuupäeva numbrit, mis oli väiksem kui 10. Antud hetkel tükeldati logikirje tühikute kohtade pealt. Kuu, kuupäev ja kellaaeg võeti järjest tükeldatud logikirjete hulgast. Kui kuupäev oli väiksem kui 10, siis formaadi hoidmiseks lisas syslog kuupäeva numbri ette ühe tühiku. Näiteks joonisel 6 on lisatud tühik *5* ette, et säilitada ajatempli formaat. Tühikutega poolitamine ei lasknud enam võtta esimest kolme esimest elementi, sest tuli tühi element. Olenemata kuupäevast peab programm saama õige kuu, kuupäev ja kellaaja logikirjest kätte.

```
Feb 14 06:29:57 kosmos sshd[2563]: Connection closed by 123.59.14.23 [preauth]
Dec  5 01:44:55 ftn2: System clock configurations have been changed by admin
```

Joonis 6. Ajatemplite erinevus.

Selles tegevusetapis leiti veel ka viga, et esimese päeva positiivseid ja negatiivseid piike ei kuvata failides. See tulenes sellest, et HashMap-de juurde lisatud kuupäevad võeti uuest päevast, mitte eelnevast. See tähendas seda, et kõik ajatemplid olid ühe võrra edasi nihkes, kuid viimase päeva oma oli õige.

Järgmiseks etapiks oli graafikute tegemine kümnest kõige suuremast positiivsest piigist. Suurimaks piigiks defineeriti piik, mille sõna esines kõige rohkem kordi. Samuti tuli ära parandada puugid, mis olid selle etapi käigus avastatud.

2.5 V Etapp

Viiendaks tegevussammuks oli parandatud ajatempli õige töötlemine sõltumata sellest, kas kuupäev oli väiksem kui 10 või ei. Samuti kuvati nüüd õigesti positiivsete ja negatiivsete piikide failides aegu, millal piigid olid esinenud.

Programmi kirjutamisel mindi Eclipse pealt üle IntelliJ peale. Programmi enda osas muudatusi ei toimunud, toimus koodi refaktoreerimine.

```
Feb 14 06:25:12 6    sawang
Feb 14 06:25:12 6    jxwu
Feb 14 06:25:12 263  root
Feb 14 06:25:12 6    uxxu
Feb 14 06:25:12 6    nywang
Feb 14 06:25:12 6    uxwu
```

Joonis 7. Auth.log.4 uus sagedustabel.

Joonisel 7 on näha auth.log.4 sõnade sagedustabeli paari rida 24 tunni jaotuse kohta. Ajatempel sisaldab ka kellaaega, muid muudatusi sagedustabeli failile ei tehtud. Näiteks 14. veebruaril esines sõna *root* 263 korda ja teised sõnad kuus korda.

Tabel 3. Auth.log.4 uued positiivsed piigid.

Kuupäev	Sõna	Esinemiste arv	Hälve	Aritmeetiline keskmine
Feb 18 00:15:00	166.111.7.72	10816	3696.339	2206.75
Feb 18 00:15:00	ekaterina	18	5.953	2.25
Feb 17 00:00:00	haviana	24	8.485	6.0
Feb 17 00:00:00	gennaro	24	8.485	6.0
Feb 17 00:00:00	caohui	24	8.485	6.0
Feb 17 00:00:00	huawei	24	8.803	6.375

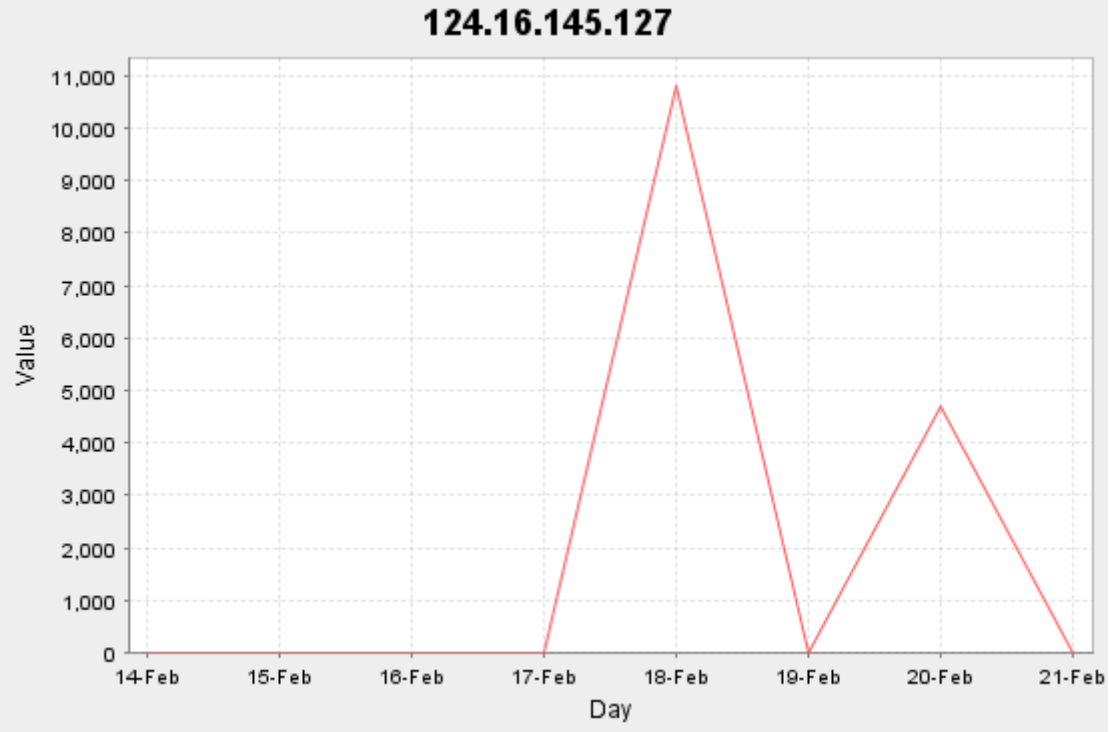
Tabel 3 sisaldab auth.log.4 positiivsete piikide väikest osa. Piikide aegadele lisati ka kellaaeg. Kellaaega saab kasutada piikide paremaks mõistmiseks ja analüüsimiseks. Samuti on see vajalik ühe tunni ja viie minutiliste ajavahemike jaoks.

Tabel 4. Auth.log.4 uued negatiivsed piigid.

Kuupäev	Sõna	Esinemiste arv	Hälve	Aritmeetiline keskmine
Feb 15 00:00:00	nagious	3	6.708	19.5
Feb 21 00:00:00	pam_unix(cron	15	11.758	44.0

Tabel 4 sisaldab auth.log.4 negatiivseid piike. Sarnaselt positiivsetele, lisati ka negatiivsetele piikidele kellaaeg.

Graafikute tegemiseks valiti teek jFreeChart, mis võimaldas teha ajadiagramme. Positiivsete piikide nimekiri sorditi sõnade esinemisarvu järgi. Esimese 10 kohta tehti graafikud, et oleks visuaalselt parem analüüsida.



Joonis 8. Sõna 124.16.156.127 esinemiste graafik.

Joonisel 8 on näha auth.log.4 logifailis esineva sõna 124.16.145.127 sagedustabeli põhjal tehtud graafik. Graafikult on näha, et 18. ja 20. veebruaril on toimunud sõna 124.16.145.127 sage kasutamine, kusjuures teistel päevadel pole seda kordagi kasutatud. Arvatavasti tuleneb see sellest, et sellelt IP-aadressilt on neil kuupäevadel üritatud katse-eksitus meetodil rünnakuga ligi pääseda arvutisse SSH kaudu. Graafiku üleval osas on näha sõna, mille sagedustabelit kuvatakse. Vasakul ääres on sõnade esinemisarv. Allosas on näha, kas ajaühikuks on päev, tund või minut.

Esimesed 10 positiivset piiki auth.log.4 sorteeritud piikide hulgast olid: 89.233.175.168, 124.116.145.127, 159.226.92.26, 159.226.92.195, 166.111.7.72, 166.11.26.5, 202.120.32.223, 202.120.58.133, 218.87.109.253 ja root. Esimesest kümnest üheksa olid IP-aadressid, kuid see oli ka loogiline, kui arvestada, et antud logi on autentimisinfo säilitamiseks. Seega neid kõiki IP-sid kasutati sellesse masinasse sisse logimise katseteks.

Selles etapis muudeti graafikud piltideks, et saaks neid ka hiljem analüüsida. Seni oli see võimalik olnud ainult programmi rakendamise ajal. Piltide formaadiks võeti PNG, et midagi ei läheks pakkimisel kaotsi.

Järgmise etapi eesmärgiks oli lisada viie minuti ja ühe tunni kaupa sõnasageduste kokku lugemine ning piikide vaheliste seoste leidmine.

2.6 VI Etapp

Kuuendas tegevussammus lisati 24-tunniste ajavahemikega sagedustabelite arvutamisele juurde ka viie minuti ja ühe tunni kaupa sagedustabelite arvutamine. Samuti lisati juurde ka seoste leidmine olemasolevate piikide vahel, mis aitaks analüüsida tulemusi ja koos esinevaid piike.

24 tunnist, ühe tunnist ja 5 minutilisi ajavahemikke kontrolliti erinevalt. 24 tunnise puhul vaadati, kas kuu ning kuupäev on samad ning kui ei ole, siis järelikult on uus päev. Ühetunnisele kehtisid samad reeglid, mis 24-tunnisele, aga sinna lisati juurde reegel, mis vaatas ka tunde: kui polnud sama tund, mis eelmisel logikirjel, siis järelikult oli järgmine tund. 5 minutiga kontrollimiseks arvutati kõikide ajatemplite minutid ümber valemiga: minut = minut - (minut % 5), kus % tähistas jääki. See võimaldas kõik minutid saada ühtsesse formaati, et neid oleks hea kontrollida. Näiteks olid kellaajad 06:25:12, 06:29:57 ja 06:30:03. Valemit rakendades saadi uuteks kellaaegadeks vastavalt 06:25:00, 06:25:00 ja 06:30:00. See võimaldas kontrollida kellaaegu, mis jäid sama 5 minuti sisse. 5 minuti kontrollimisel rakendati samu reegleid, mis ühetunnise ajavahemiku puhul ning lisati juurde ka uute minutite kontroll: kui polnud sama minut, mis eelmisel logikirjel, siis järelikult oli uus 5-minutiline ajavahemik.

Näiteks sõna *user* esines esimeses 24-tunnises, 1-tunnises ja 5-minutilises ajavahemikus vastavalt 40948, 201 ja 1 korda.

```
Feb 14 06:25:12 kosmos CRON[2431]: pam_unix(cron:session): session closed for user root
Feb 14 06:29:57 kosmos sshd[2563]: Connection closed by 123.59.14.23 [preauth]
Feb 14 06:30:03 kosmos sshd[2565]: Connection closed by 123.59.14.23 [preauth]
```

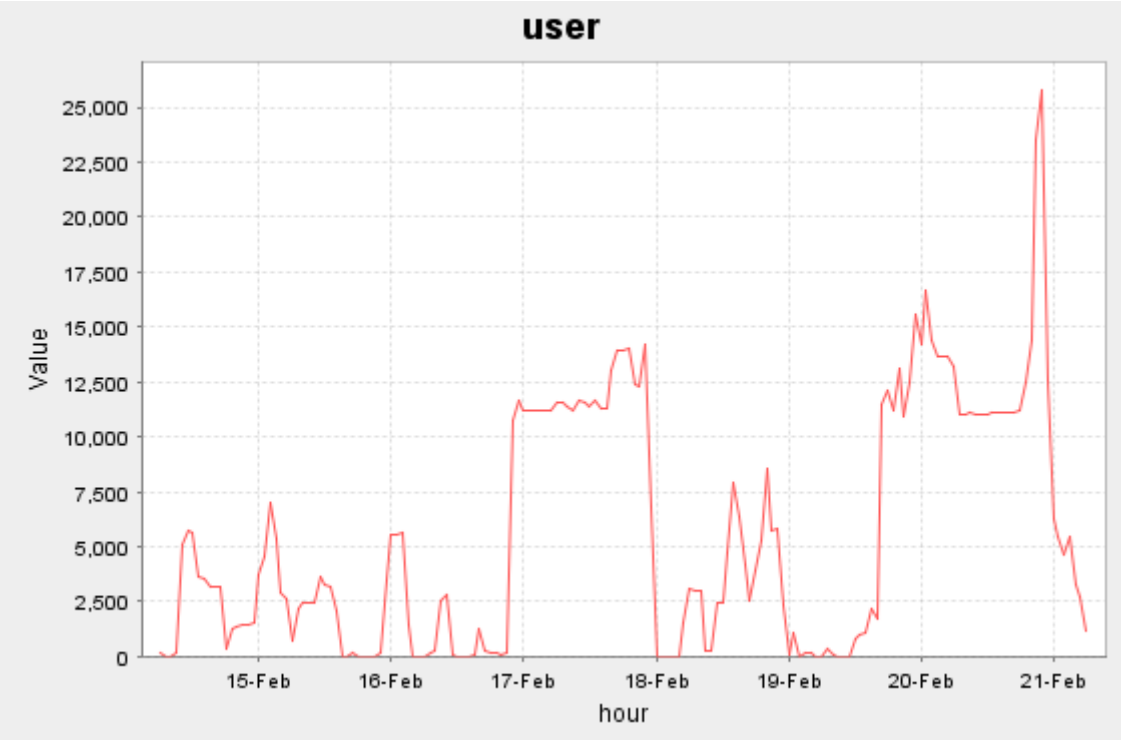
Joonis 9. Auth.log.4 logi 3 esimest rida.

```
Feb 14 06:25:00       1       session
Feb 14 06:25:00       1       root
Feb 14 06:25:00       1       by
Feb 14 06:25:00       1       for
Feb 14 06:25:00       1       Connection
Feb 14 06:25:00       2       closed
Feb 14 06:25:00       1       [preauth]
Feb 14 06:25:00       1       pam_unix(cron
Feb 14 06:25:00       1       123.59.14.23
Feb 14 06:25:00       1       session)
Feb 14 06:25:00       1       user
```

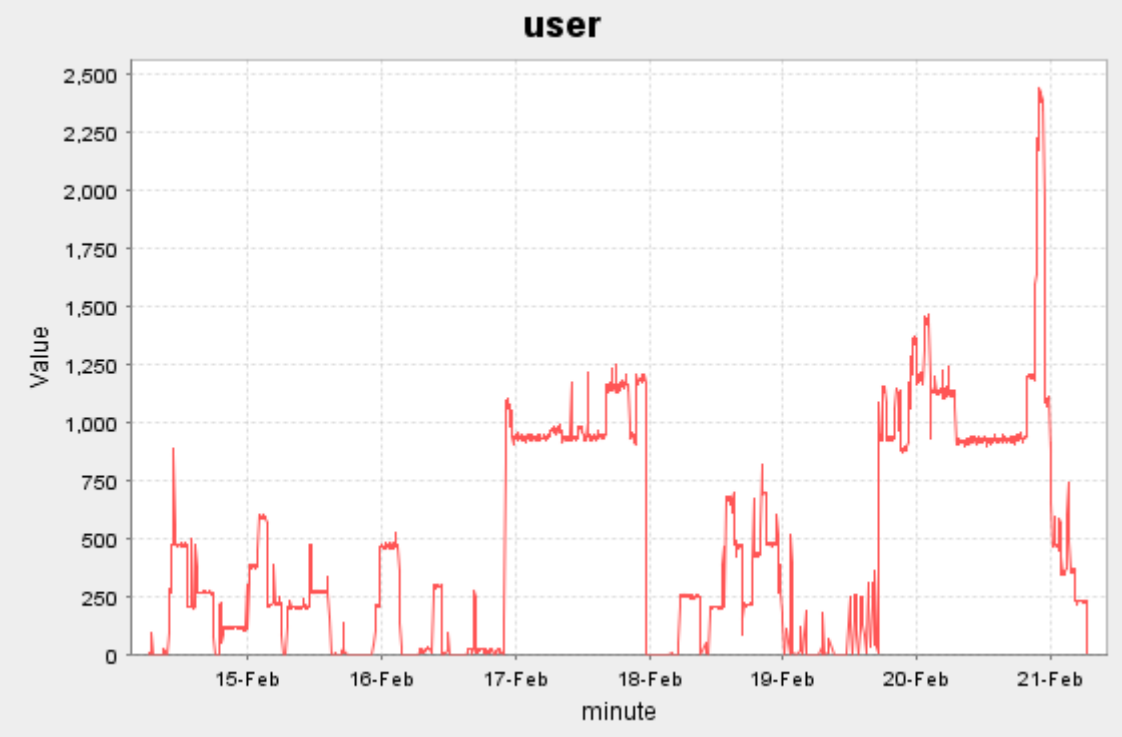
Joonis 10. Auth.log.4 esimese viie minuti sagedustabel.

Joonisel 10 on näha auth.log.4 esimest viie minuti sagedustabelit. Kõrvutades joonisega 9 on siit näha, et kokku on loetud ainult esimese kahe rea sõnad ning kolmandat rida pole arvestatud, sest sealt hakkab uus viie minuti piir. Sekundid ei mängi analüüsimisel rolli, seega on nad pandud nullideks.

Näiteks sõnaga *user* tekkisid positiivsete piikide graafikud (vt joonis 11, joonis 12) vastavalt ühe tunni ja 5-minutiliste ajavahemikega, kuid 24 tunni kaupa sõna *user* ei olnud piik, sest esines iga päev ligikaudu sama arv kordi.



Joonis 11. Sõna *user* sagedustabel tunniste ajavahemikega.



Joonis 12. Sõna *user* sagedustabel 5-minutiliste ajavahemikega.

Joonistelt 11 ja 12 on näha, et graafikud on peale vaadates väga sarnased. Viie minuti graafik on palju detailsem, sest seal on rohkem punkte. Ühe tunni punkti kohta on viie minuti graafikul 12 punkti, sest üks tund koosneb 12 korda viiest minutist. Suurem erinevus joonistel on sõnade esinemiste arv, kuid see on ka loogiline, sest 5-minutiline ajavahemik

on 12 korda väiksem, kui tunnine ajavahemik. Seega on ka graafikud keskmiselt 10 korra võrra erinevad sõnade arvu esinemise poolest.

Tabel 5. Sõna *user* positiivsed piigid tunniste ajavahemikega.

Kuupäev	Sõna	Esinemiste arv	Hälve	Aritmeetiline keskmine
Feb 20 01:00:00	user	16642	5478.587	5581.355
Feb 20 21:00:00	user	23487	5478.587	5581.355
Feb 20 22:00:00	user	25806	5478.587	5581.355

Tabel 5. on näha sõna *user* ühe tunni positiivsed piigid. Piigid lähevad kokku ka joonisel 11 kolme kõige kõrgema punktiga.

Tabel 6. Sõna *user* positiivsed piigid 5-minutiliste ajavahemikega.

Kuupäev	Sõna	Esinemiste arv	Hälve	Aritmeetiline keskmine
Feb 20 21:10:00	user	1593	449.242	589.899
Feb 20 21:15:00	user	1635	449.242	589.899
Feb 20 21:20:00	user	1791	449.242	589.899
Feb 20 21:25:00	user	2202	449.242	589.899
Feb 20 21:30:00	user	2218	449.242	589.899
Feb 20 21:35:00	user	2221	449.242	589.899
Feb 20 21:40:00	user	2238	449.242	589.899
Feb 20 21:45:00	user	2174	449.242	589.899
Feb 20 21:50:00	user	2276	449.242	589.899
Feb 20 21:55:00	user	2440	449.242	589.899
Feb 20 22:00:00	user	2419	449.242	589.899
Feb 20 22:05:00	user	2424	449.242	589.899
Feb 20 22:10:00	user	2381	449.242	589.899
Feb 20 22:15:00	user	2415	449.242	589.899
Feb 20 22:20:00	user	2387	449.242	589.899

Feb 20 22:25:00	user	2368	449.242	589.899
Feb 20 22:30:00	user	2345	449.242	589.899
Feb 20 22:35:00	user	2374	449.242	589.899
Feb 20 22:40:00	user	1958	449.242	589.899
Feb 20 22:45:00	user	1734	449.242	589.899
Feb 20 22:50:00	user	1734	449.242	589.899

Tabel 6 sisaldab viie minuti positiivseid piigke. Kaks piiki kattuvad ka tunniajase sagedustabeli piikidega: kella üheksane ja kümnene piik. Kuid puudub ühine piik 20. veebruaril kella ühesega, sest viieminutilisel on punkte nii palju rohkem ja see mõjutab piikide arvutamist.

Viie minuti ja ühe tunni piikide ja graafikute hulgast leiti rohkem huvitavaid sündmuseid, kui ühe päeva omadest. Seda seepärast, et joonised sisaldasid rohkem punkte ja sai täpsema ülevaate sõnade esinemisarvust. Päevastel graafikutel olid enamasti IP-aadressid, mis esinesid ainult ühel või kahel päeval.

Sarnaste sõnade leidmiseks arvutati korrelatsioon 100 esimese sõna kohta. Sõnadeks valiti piigid, mis olid kõige suuremad. Korrelatsiooni arvutamiseks võrreldi kahe sõna sagedustabeleid. Alustuseks lisati igale väärtusele juurde üks, et ei tuleks nulliga jagamist. Järgmiseks leiti suhe iga punkti suhtes: mitu korda on esimene punkt teisest suurem. Suhtest võeti naturaallogaritm, et võrdsustada olukorda, kus teine punkt on esimesest suurem. Lõpetuseks leiti suhete logaritmide aritmeetiline keskmine ja standardhälve. Defineerime, et sõnade vahel on seos, kui standardhälve on väiksem kui kaks. Kui standardhälve on suurem, siis seost sõnade vahel antud meetodiga ei leidu. Standardhälbe suurus valiti katseliselt, et tuleks piisavalt seoseid, kuid ei tuleks seoseid, mis joonistele peale vaadates ei ole sarnased.

```
user  from  0.43276127334324094
user  [preauth]  0.21895448350225094
from  [preauth]  0.3994341770032088
for  ssh2  0.22891629972643454
```

Joonis 13. Auth.log.4 korrelatsioonid.

Joonisel 13 on näha paari rida auth.log.4 viieminutilise ajevahemikuga sõnade korrelatsioonidest. Kõik failis esinevad sõnad kuuluvad 100 kõige rohkem esineva sõna hulka, mis on positiivsed piigid. Esimesed kaks on sõnad, mille vahel on seos ja neile järgneb standardhälve. Mida väiksem on standardhälve, seda tugevam on seos kahe sõna vahel. Kui standardhälve on null, siis on kahe sõna vahel täielik seos ja graafikud peavad olema täpselt samasugused.

Korrelatsioonide paremaks analüüsimiseks sortisime korrelatsioonid Exceli abil standardhälvete kasvamise järjekorras.

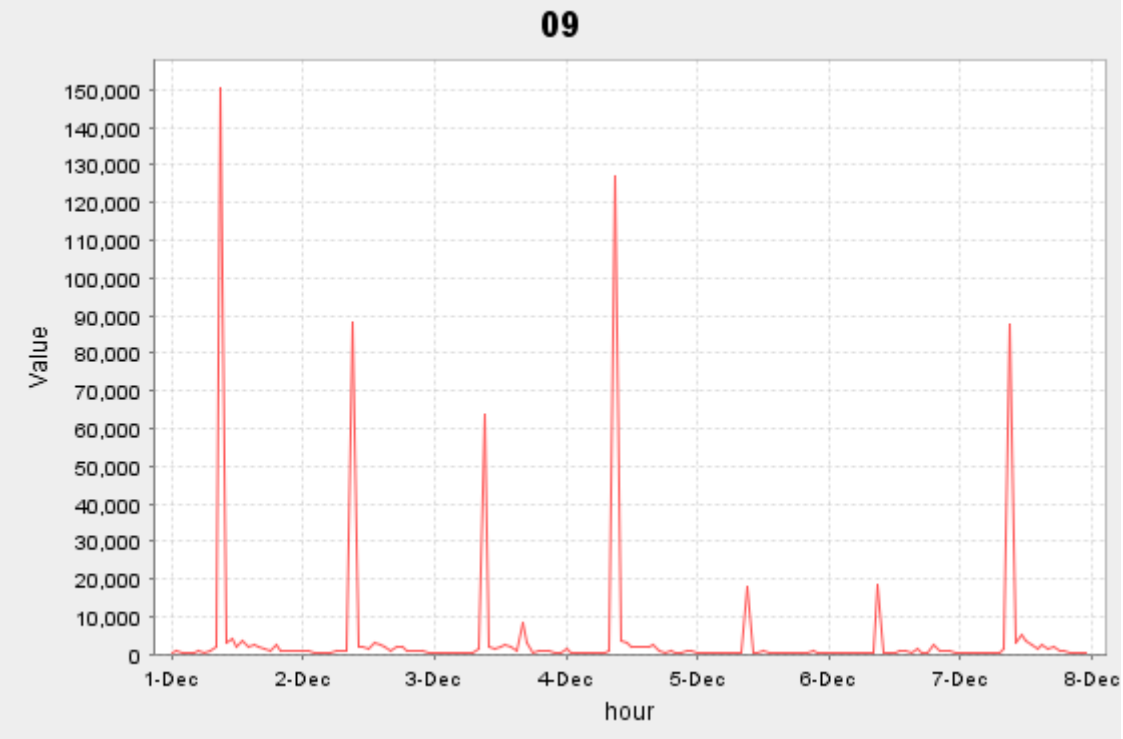
ruser	rhost	0
ruser	euid	0
ruser	tty	0
ruser	logname	0
rhost	euid	0
rhost	tty	0
rhost	logname	0
euid	tty	0
euid	logname	0
tty	logname	0
POSSIBLE	ATTEMPT!	0
POSSIBLE	BREAK-IN	0
reverse	mapping	0
reverse	getaddrinfo	0
reverse	checking	0
mapping	getaddrinfo	0
mapping	checking	0
ATTEMPT!	BREAK-IN	0
getaddrinfo	checking	0
14h245.xjtu.edu.cn	[202.117.14.245]	0

Joonis 14. Auth.log.4 sõnad, mille vahel on täielik seos.

Joonisel 14 on näha auth.log.4 viie minuti ajavahemiku korrelatsioonid, mille standardhälve on 0. Olgu meil kolm sõna: a, b ja c. Kui sõnade a ja b vahel on seos ning ka sõna a ja c vahel on seos, siis peab ka olema seos sõna b ja c vahel. See kehtib ainult juhul, kui kahe sõna vaheline standardhälve on null. Näiteks sõnad *POSSIBLE, ATTEMPT!* ja *BREAK-IN* on kõik omavahel seotud ja esinevad ainult koos. *Feb 14 10:50:04 kosmos sshd[4868]: reverse mapping checking getaddrinfo for 82-78-146-141.static.brasov.rdsnet.ro [82.78.146.141] failed - POSSIBLE BREAK-IN ATTEMPT!* on rida auth.log.4 failist, kus antud sõnad esinevad koos. Need sõnad esinevad koos, kui on toimunud ebaõnnestunud rünnak.

24-tunniste ajavahemikega auth.log.4 korrelatsioonidest leiti kaks IP-aadressi, mille vahel oli täielik seos. 166.11.26.6 ja 124.16.145.127 graafikud olid täpselt samasugused: samadel päevadel käidi sama palju kordi masinasse nimega kosmos sisse logida proovimas.

Uuriti ka nädalast tulemüüri logi, millega eelnevalt oli katsetatud, kas masinas on võimalik jooksutada suuri logisid ilma probleemideta. Tunnise ajavahemikuga korrelatsioonidest leiti kaks piiki: 09 ja 11. Põhjalikumal uurimisel leiti, et piigid esinevad ainult hommikuti ja sõnu esineb rohkem tööpäevadel (joonis 15).



Joonis 15. Tulemüüri logi sõna *09* tunnise ajavahemikuga sagedustabel.

09 ja 11 graafikud olid sarnased ning need sõnad esinesid logides järgnevalt: *Dec 1 04:14:53 ftu1 ftu1: NetScreen device_id=XXXXXXXXXXXX [Root]system-alert-00442: TCP sweep! From 10.1.42.93 to zone Untrust, proto TCP (int ethernet0/2.2). Occurred 11 times. (2015-12-01 04:14:51)*. Logikirje sisaldas sõna *11*, sest mingit logikirjet esines 11 korda järjest, aga selle asemel, et programm järjest kirjutaks sama kirjet 11 korda, tegi see seda ühe korra ning lisas lõppu, et see juhtus tegelikult 11 korda järjest. See oli hea moodus ruumi kokku hoidmiseks, sest logi oli isegi koos sellise lühendamisega väga suur. Samas oli niimoodi leitud sweep täiesti huvitav sündmus - üks proovikive tulemüüri administraatori jaoks, et kontrollida, kas programm leiab üles teadaolevad piigid hommikuti.

Järgmise etapi eesmärgiks oli lisada programmile võimalus seda jooksutada käsurealt.

2.7 VII Etapp

Viimaseks etapiks tehti programmist .jar fail, et seda saaks kergesti jagada ja käsurealt jooksutada. Lisati ka parameetrite võtmine käsurealt, et oleks võimalik erinevat sorti tulemusi saada.

Käsurealt jooksutamiseks saadava info jaoks mindi .jar faili sisaldavasse kausta ning sisestati käsk: *java -jar logAnalyser.jar -h* või *java -jar logAnalyser.jar --help*.

```
Usage: <main class> [options]
  Options:
    --corr
      Calculate positive peaks correlations.
      Default: false
    --image, -i
      Number of top peak images. 0 means no images.
      Default: 10
  * --path, -p
      Path of the log file.
    --savePath, -sp
      Path where the results will be saved
      Default: <empty string>
    --time, -t
      Frequency time.
        5 for 5 minutes
        1 for 1 hour
        24 for 24hour/1day
      Default: 24
    --topCount, -tp
      Number of peaks to use for correlation analysis.
      Default: 100


Total runtime: 101ms
```

Joonis 16. Käsurea väljund logAnalyser.jar *--help* lipule.

Joonisel 16 on näha väljundit, mis seletab, milliseid parameetreid saab kasutada, et jooksutada programmi käsurealt. *--path* parameeter on kohustuslik ning ilma selleta programm ei tööta, sest logifail on vajalik programmi töötamiseks. Teistel parameetritel on olemas vaikeväärtust, kuid seda on võimalik muuta, kui lisada lipp ja uus väärtus. Näiteks käsk *java -jar logAnalyser.jar -p C:\Users\karll\Documents\logAnalyser\logs\kosmos-auth-logs\auth.log.4 --corr -sp C:\Users\karll\Documents\saveResults\ -t 5 -i 25 --topCount 250* võtab faili auth.log.4, arvutab piigid viieminutilise ajavahemikega, leiab korrelatsioonid 250 suurimale piigile, teeb 25 suurima piigi graafikud ning salvestab need ette antud *saveResults* kausta.

See etapp oli viimane, sest suudeti kontrollida esialgset hüpoteesi, et niimoodi oli võimalik huvitavaid logikirjeid leida.

3. Programm

Siinse lõputöö raames loodi programm logAnalyser. Programmi põhiülesanneteks on piikide arvutamine, graafikute tegemine ning korrelatsioonide leidmine. Järgnevates jaotistes kirjeldatakse programmi ülesehitust ja antakse juhised projekti ülesseadmisest ning programmi kasutamisest käsureal.

3.1 Programmi ülesehitus

Programm logAnalyser on kirjutatud Javas ning selleks kasutati Java versiooni 8. Programmi kirjutamiseks kasutati kahte erinevat integreeritud programmeerimiskeskkonda: Eclipse ja IntelliJ.

Programm koosneb ühest Java klassist, kus toimub kõik vajalik tegevus. Kasutatakse kahte lisateeki: jFreeChart ja jCommander. JCommander on käsurealt parameetrite töötlemiseks ja jFreeChart on graafikute loomiseks [5][6].

Programmi rakendamine hakkab pihta *main*-meetodist, kus toimub käsurealt saadud parameetrite töötlemine jCommanderi teegiga, mis väärtustab vastavad globaalsed muutujad. Samuti alustatakse aja lugemisega, et programmi töötamise lõpus väljastada aeg, mis kulus programmil töö tegemiseks. Main meetodis jooksutatakse ka kontrolli funktsioon, mis vaatab, kas tuleb väljastada parameetrite kohta käiv info või tuleb rakendada programmi. Kui --help või -h lippu pole antud käsurealt, siis pole vaja infot kuvada.

Järgmisena kontrollitakse, kas --image või -i on positiivsed numbrid, sest käsurealt on võimalik neile anda ka negatiivseid väärtuseid. Negatiivsete väärtuste korral ei ole võimalik graafikuid teha ning tagastatakse veateade. Edasi järgneb programmi põhiosa, mis kontrollib ja kutsub vajadusel esile teised meetodid.

Programmi põhiosas võetakse alustuseks esimeselt realt kuupäev, mille järgi hakatakse võrdlema järgnevaid kuupäevi, et saaks teada, kas on järgmine päev, tund või viis minutit. Sõltuvalt ajavahemiku valikust kutsutakse välja meetod *countWords*, millele antakse kaasa argumendiks funktsioon, mis kontrollib, kas on järgnev ajavahemik. Selline lähenemine aitab vältida koodi kordumist: kasutada sama sõnade kokku lugemist ja ainult järgneva ajaühiku kontroll on sõltuv meetodist, mis kaasa anti. Sõnade kokkulugemismeetod käib logi ridu järjest läbi, saab sealt kuupäeva ja võrdleb seda kaasa antud meetodis eelneva kuupäevaga. Kui on uus ajavahemik, siis vanad sõnad salvestatakse ja kirjutatakse faili ning alustatakse uute sõnade lugemist. Kui pole uus ajavahemik, siis lisatakse või suurendatakse vastava sõna esinemiskorda HashMapis. HashMap on paisktabel, mille igale võtmele vastab teatud väärtus. Näiteks sõna ja tema esinemiste arv. Sõnu loetakse kokku, kuni jõutakse logifaili lõpuni ning siis kirjutatakse viimast korda kokku loetud sõnad faili. Lisaks loetakse kokku ka kõikide sõnade esinemisarv terve aja peale. Meetod tagastab terve aja peale sõnade esinemisarvud ja igas ajavahemikus esinevate sõnade arvud. Antakse märku, et sõnade kokku lugemine oli edukas ja on lõpuni viidud. See on programmi kõige rohkem aega võttev osa.

Kui on kätte saadud sõnade esinemiste arvud, siis alustatakse piikide leidmisega. Piikide leidmiseks tuleb esmalt arvutada kõikide sõnade aritmeetilised keskmised. Selleks kasutatakse paisktabelit, mille tagastas sõnade kokku lugemise meetod ning mis sisaldab kogu aja peale sõnade esinemiste arvu. Kui arvutamine tehtud, siis meetod tagastab uue paisktabeli, mis sisaldab kõiki esinevaid sõnu ja nende keskmist esinemiskorda ühes ajavahemikus. Meetodi lõppedes antakse teada, et aritmeetilise keskmise arvutamine on lõpetatud.

Järgmiseks arvutatakse igale sõnale standardhälbed, mida on vaja piikide leidmiseks. Meetod võtab argumentideks kõikide sõnade aritmeetilised keskmised ja iga ajavahemiku sõnade esinemisarvu. Programm arvutab kõikidele sõnadele, mis esinesid logis, standardhälbe. Sõna ja standardhälve lisatakse paisktabelisse. Meetodi lõpus tagastatakse see paisktabel ja väljastatakse teade, et standardhälbe arvutamine on edukalt lõppenud.

Kui aritmeetilised keskmised ja standardhälbed on käes, siis hakatakse arvutame piike. Piikide arvutamise meetod võtab kolm argumenti: paisktabelid, millest esimene sisaldab terve aja sõnade keskmiseid väärtuseid ja teine, mis sisaldab kõikide sõnade standard hälbeid ning kolmandaks argumendiks on kõikide ajavahemike sõnade esinemiste arv. Nende kolme argumendiga arvutatakse piigid ja kirjutatakse kahte erinevasse faili, sõltuvalt kas nad on positiivsed või negatiivsed. Meetod tagastab loendi, millel on kindel järjekord ning mis sisaldab uusi loendeid, kus on info piigi kohta: nimetus, väärtus ja esinemisaeg. Kui piigid on leitud ja faili kirjutatud, siis väljastatakse ekraanile, et piigid on arvutatud.

Järgmine samm on saadud piikide loendi sortimine. Selleks on üle kirjutatud Java *sort* meetod, kuna võrreldakse loendi seest saadud loendi kindlaid väärtusi. Sorditakse piigi esinemiskorra järgi. Näiteks, kui tagastatakse loend piikidest [[“tere”, 200, ”Nov 12 06:15:00”], [“sõna”, 521, ”Nov 13 00:05:00”], [“näide”, 333, ”Nov 14 11:30:00”]], siis sortimisel oleks tulemuseks [[“sõna”, 521, ”Nov 13 00:05:00”], [“näide”, 333, ”Nov 14 11:30:00”], [“tere”, 200, ”Nov 12 06:15:00”]], kus piigid on pandud suuruse järjekorda.

Järgmiseks eemaldatakse sorditud piikide listist kõik korduvad piigid. Graafikute tegemisel ja korrelatsioonide leidmisel pole kasu, kui sama sõna kohta käiv piik esineb mitu korda, sest seda kuvatakse nagunii ainult üks kord ning muidu tehtaks ülearu graafikute ja korrelatsioonide arvutamisi.

Kui üleliigsed piigid eemaldatud, siis luuakse piikide loendist kindla suurusega loend. Algväärtuseks on 100, aga see on lipuga muudetav käsurealt jooksutades, Listi suurus tehakse väiksemaks, sest piike võib olla tuhandeid ja neist kõigist piltide ja korrelatsioonide arvutamine oleks tulutu. Kui piikide loend on juba väiksem, kui ette antud väärtus, siis jäetakse loend muutmata.

Järgmiseks arvutatakse korrelatsioonid, kui see on käsurealt määratud. Ilma käsurea käsuta *--corr* korrelatsioone ei arvutata. Meetod võtab argumentideks minimeeritud piikide loendi ja kõikide ajavahemike sõnade väärtused. Tulemuseks kirjutatakse faili sõnad, mille vahel leidus seos ja samuti ka arvutatud standardhälbe väärtus, et oleks näha, kui seotud sõnad omavahel on. Kui korrelatsioonid on leitud, siis väljastatakse ekraanile märkus, et korrelatsioonide arvutamine on tehtud.

Algselt on määratud suurima 10 piigi graafiku tegemine, vastavalt kas tegemist on minuti, tunni või päeva graafikuga. Käsurealt saab lipuga *--image* või *-i* muuta graafikute arvu. Kui arvuks määratakse 0, siis graafikuid ei tehta ning kui arvuks määratakse suurem väärtus, kui piike kokku on, siis tehakse kõikidest piikidest graafikud. Graafikud salvestatakse kõik *graph* kausta. Graafikute tegemise lõpus väljastatakse ekraanile teade, et graafikute tegemine on lõpetatud.

Enne programmi lõppemist väljastatakse ekraanile ka programmi töötamiseks kulunud aeg.

3.2 Projekti ülesse seadmine IntelliJ-s

Järgnev jaotis selgitab, kuidas programmi kompileerida, muuta ning edasi arendada. Java arendajatele pole see vajalik, küll aga võib programm huvi pakkuda süsteemi administraatori profiiliga inimestele, kes ei ole Java arendusega kursis.

IntelliJ oli mugav ning omab lihtsat sisseehitatud *git* võimalust, et programmi saaks kasutada mitmes arvutis ning seetõttu on arendamiseks valitud IntelliJ IDEA integreeritud programmeerimiskeskkond. Projekti lähtekoodi hoitakse githubis ning on leitav aadressil https://github.com/KarlEST/logAnalyser.

Selleks, et projekti enda masinas tööle saada on kõigepealt vaja installida masinasse Java 8. Kui puudub, siis on võimalik see alla laadida lehelt http://www.oracle.com/technetwork/java/javase/downloads/index.html, kust saab valida enda masinale sobiva versiooni. Tuleb võtta JDK(Java Development Kit), mis on mõeldud Java arendamiseks. Kui sobiv versioon on alla tõmmatud, siis tuleb see installida. Esmalt programm lahti pakkida, kui tõmmatud on kokku pakitud versioon. Järgmisena topelt klõps programmil ja jälgida ekraanil olevaid juhiseid, et Java JDK installida.

Kui Java on installitud või juba varasemast olemas, siis tuleb installida *IntelliJ IDEA* lehelt https://www.jetbrains.com/idea/#chooseYourEdition. Sealt valida kas Community või Ultimate versioon. Ultimate on võimsam ja sisaldab rohkem võimalusi. Seda on võimalik kasutada 30 päeva tasuta ning üliõpilastel õppe eesmärkidel isegi kauem. Tõmmata alla endale sobiv versioon ja teha topelt klikk sellel ja installida programm.

Kui programm installitud, siis käivitada ja valida endale sobivad seaded. Laadida githubist alla projekt: vajutada nupule *Download ZIP* ning pakkida lahti või kloonida git repositoorium. Avada IntelliJ ja valida import project. Valida lahti pakitud projekti kaust ja klikkida *next*, kuni tuleb lisada JDK. Selleks valida Java 8, mis eelnevalt sai arvutisse installitud ning IntelliJ on valmis seatud. Läheb veidike aega, kuni projekt valmis seatakse ja analüüsitakse.

Kui kõik on valmis, siis on vasakul servas näha kaustasid ja faile. Avada *src* kaust, kus asub logAnalyser, mis sisaldab programmi koodi. Tehes LogAnalyser klassi peal topelt kliki avaneb kood. Koodi saab jooksutada, kui valida ülevalt *Run* ja sealt uuesti *Run*. Kiirklahvi kombinatsioon sama käsu jaoks on Alt+Shift+F10 ning selle vajutamisel avaneb väike hüpikaken, kus saab valida, mis klassi jooksutatakse. Valikus on *Edit Configuration* ja *logAnalyser*. Valida logAnalyser ning sellele järgneb programmi jooksutamine, mis väljastab all konsooli lahtris veateate, sest pole sisestatud logifaili asukohta. Vajutada uuesti kiirklahvi kombinatsiooni või valida ülevalt reast *Run* ning seekord võtta *Edit Configuration*. Avaneb aken ning kirjutada lahtrisse *Program arguments* argumendid, mida soovite programmile kaasa anda. Näiteks kui sisestada *-h* või *--help* ja jooksutada uuesti logAnalyser klassi, siis tuleb väljundiks parameetri lippude seletused. Programmi koodi saab muuta ja uuesti jooksutada enda soovide järgi.

Selleks, et teha logAnalyser .jar faili, tuleb avada ülevalt reast *File* alt *Project structure* või siis selle kiirklahvi kombinatsioon on Ctrl+Alt+Shift+S. Avaneb uus aken, kus on kohe näha, mis Java versiooni kasutatakse. Valida vasakust tulbast *Artifacts* ning vajutada väikest rohelist pluss nuppu. Avanenud menüüst valida *JAR* ja *From modules with dependencies*, mis avab uue akna. Avanenud aknas valida *Main Class*, milleks on logAnalyser ning vajutage *OK*, mis suunab tagasi eelnevasse aknasse. Vajutada *Accept* ning sulgeda aken. Nüüd on tehtud artifakt, millest hakatakse .jar faili tegema. Selleks vajutada ülevalt reast *Build* peale ning valida *Build Artifacts*. Avaneb väike aken ning sealt pealt vajutada *Build* ning IntelliJ hakkabki -jar faili tegema. Fail salvestatakse *out/artifacts/lonAnalyser_jar* kausta, kui salvestamis kohta pole muudetud. Sealt leiabki .jar faili, millega on võimalik programmi jooksutada käsurealt.

3.3 Jar faili kasutamine käsurealt

LogAnalyser .jar fail on leitav nii githubis(kaustas *jar*), kuid seda on võimalik ka ise teha kasutades selleks näiteks IntelliJ. Järgnevalt kirjeldatakse, kuidas logAnalyser.jar kasutada käsurealt ning mis parameetreid on võimalik sinna kaasa anda.

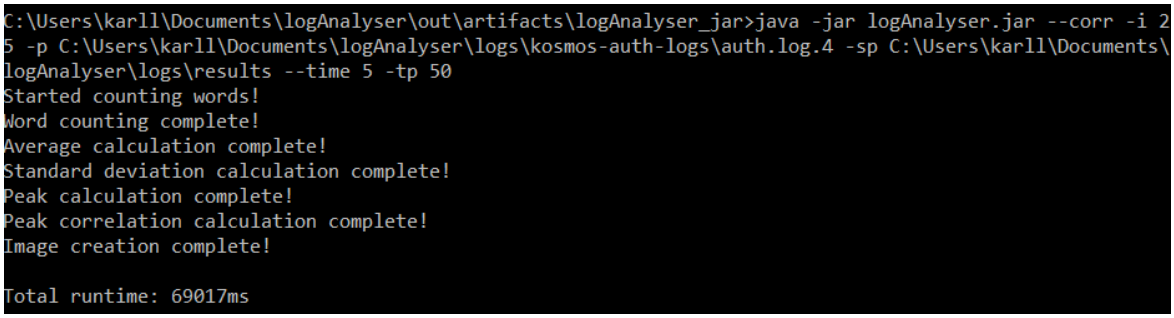
Jar-faili jooksutamiseks on vajalik, et masinas oleks installitud Java. Antud projekt on ülesse ehitatud kasutades Java 8-t ning soovitatav on kasutada seda sama. Kui Java on olemas, siis on võimalik jooksutada .jar faile järgneva käsu põhjal: *java -jar jarFailiNimi.jar*. Seega on võimalik logAnalyserit jooksutada käsuga: *java -jar logAnalyser.jar* eeldusel, et käsureaga ollakse õiges kaustas.

```
C:\Users\karll\Documents\logAnalyser\out\artifacts\logAnalyser_jar>java -jar logAnalyser.jar
Exception in thread "main" com.beust.jcommander.ParameterException: The following option is required: --path, -p
        at com.beust.jcommander.JCommander.validateOptions(JCommander.java:328)
        at com.beust.jcommander.JCommander.parse(JCommander.java:289)
        at com.beust.jcommander.JCommander.parse(JCommander.java:271)
        at com.beust.jcommander.JCommander.<init>(JCommander.java:212)
        at logAnalyser.main(logAnalyser.java:627)
```

Joonis 17. Terminali vaade logAnalyser.jar faili jooksutamisese käsust.

Joonisel 17 on näha, et ollakse kaustas, kus on logAnalyser.jar fail ning jooksutatakse käsku, kuid tulemuseks on erand. Rohkemal lugemisel leiame, et veateade märgib ära, et *--path* või *-p* on puudu ning see on kohustuslik parameeter logAnalyseril. Parameetrid sisestatakse käsurealt pärast jar-faili nimetust ning need on eraldatud omavahel tühikutega. Selleks, et saada infot parameetrite kohta, on lisatud suvand *--help* või *-h* (vaata joonis 20).

Jooniselt 20 on näha parameetreid, mida on võimalik logAnalyserile kaasa anda. *--corr* suvand on ainuke, mis ei vaja endaga argumente. See tähendab, et kui lisada *--corr* logAnalyserile, siis arvutatakse korrelatsioonid ja tehakse ka fail, mis sisaldab saadud tulemusi. *--image* või *-i* suvand nõuab enda järgi väärtust, ilma väärtuseta väljastatakse veateade. See suvand määrab ära, mitu graafikut tehakse ning selle väärtuseks võivad olla kõik naturaalarvud, kaasa arvatud null. Kui väärtuseks on 0, siis ei tehta ühtegi graafikut. Ilma suvandit määramata on vaikeväärtuseks 10, mis tähendab, et tehakse 10 graafikut. *--path* või *-p* on kohustuslik suvand ilma milleta logAnalyser ei tööta. Selle väärtuseks on rajanimi, mis viitab logifaili asukohale. *--savepath* või *-sp* määrab salvestus kausta asukoha. Programmi käigus tekkinud failid salvestatakse kõik kausta, mille nimetuseks on logifaili nimi. Kui kaust ei eksisteeri, siis see kaust luuakse. Vaikeväärtuseks on tühi sõne, mis tähendab, et kui salvestuskausta ei määrata suvandiga, siis tekitatakse tulemuskaust samasse kohta, kus asub ka logAnalyser.jar fail. Näiteks käsk *java -jar logAnalyser.jar -p C:\Users\karll\Documents\logAnalyser\logs\kosmos-auth-logs\auth.log.4 -sp C:\Users\karll\Documents\logAnalyser\logs\results* võtab logi auth.log.4 ja salvestab tulemused kausta \logs\results, mis on varem valmis tehtud. Tulemuseks on kaust auth.log.4, mis sisaldab sõnasagedusi iga päeva kohta, positiivsed ja negatiivseid piike ja kausta *graphs*, mis sisaldab 10. graafikut. Järgmiseks suvandiks on *--time* ja *-t*, mis määravad ära, kas ajavahemikuks võetakse päev, tund või viis minutit. Vaikeväärtuseks on päev ning seda saab määrata numbrilise väärtusega 24, ühe tunni jaoks on väärtus 1 ning viie minuti jaoks on väärtus 5. Sisestades mingi muu aja väärtuse, väljastatakse veateade. Viimaseks suvandiks on *--topcount* või *-tp*, mis määrab ära, mitme piigi vahel üritatakse korrelatsiooni leida. Vaikeväärtuseks on 100, mis tähendab, et 100 suurima sõnade arvuga piigi vahel otsitakse seoseid. Kui piikide arv on väiksem, kui ette antud *--topcount* väärtus, siis tehakse kõikide piikide kohta seoste otsimine.



Joonis 18. Terminali vaade logAnalyser.jar faili kasutamise käsust koos suvanditega.

Joonisel 18 on näha tüüpilist logAnalyseri kasutamise näidet, mis sisaldab kõiki suvandeid. See käsk võtab kaustast *logs/kosmos-auth-logs* logi auth.log.4 ning salvestab tulemused kausta *logs/results*. Tulemusteks on viie minuti kaupa leitud sõnasageduse tabelid, positiivsed ja negatiivsed piigid, 25 suurima piigi graafikud ning 50 piikide vahelised seosed. Ekraanil kuvatakse ka järk-järgult mingi etapi valmimine ning lõpus kuvatakse programmi tööaeg.

4. Tulemused ja järeldused

Järgnevas peatükis räägitakse, et mis olid analüüsi tulemused, mida neist järeldada ja mida tulevikus annaks paremaks muuta,

4.1 Tulemuste analüüs

Lõputöö eesmärk oli leida struktureerimata logidest huvitavaid sündmuseid, kasutades selleks sõnasagedusel põhinevat logianalüüsi. Lõputöö käigus jõuti järelduseni, et sõnasagedusel põhineva logianalüüsiga on võimalik leida huvitavaid sündmuseid.

Programm logAnalyser kasutab lihtsaid statistilisi meetodeid nagu standardhälbe ja aritmeetilise keskmise arvutamine ning rakendab sõnasagedusel põhinevat logide analüüsimist. Programmi tulemust analüüsides kontrolliti hüpoteesi, et piikide leidmine aitab leida huvitavaid sündmuseid. Põhjalikum analüüs tehti kahe logifailiga.

Esimeseks uuritavaks logifailiks oli autetmislogi auth.log.4. Päeva täpsusega analüüsimisel leiti, et põhiline osa piikidest on IP-aadressid. Kuna auth.log.4 on autentimislogi Internetist kättesaadavale SSH-ga masinale, mis sisaldab informatsiooni sisselogimiskatsete kohta, siis on täitsa mõistetav, et see sisaldab palju IP-aadresse, kust on üritatud sisse logida. Piigid olid IP-aadressidest, kust üritati sisse murda sellesse masinasse ning mõned lähemalt uuritud rünnakud tulid Lõuna-Koreast. Korrelatsioonidest leiti kaks aadressi, mis esinesid täpselt sama palju kordi ning nende vahel oli täielik seos: 166.111.26.5 ja 124.16.145.127. Leiti ka sõnagruppe, mille vahel oli täielik seos ehk need sõnad esinesid alati koos. Näiteks *Too, many* ja *failures* esinesid alati koos. 5-minutilistel ajavahemikel esinesid koos sõnad *break, in* ja *attempt*, mis tekkisid ebaõnnestunud rünnakutel.

Sõnade vahelised seosed pole huvitavad, sest annavad vähe infot, kuna enamus neist seostest kirjeldavad mitmesõnalisi tüüpilisi logisündmusi.. Rohkem huvi pakuksid IP-aadresside vahelised seosed, aga neid leidus vähe. Graafikuid vaadates oli näha seoseid, mis võiksid korrelatsioonidest välja tulla, kuid seda ei juhtunud. Päeva granulaarsusega analüüsimisel leiti vähe seoseid, sest andmepunkte polnud piisavalt ja väike erinevus nullist mõnes punktis mõjus tugevalt. Tunnisel ja 5-minutilisel analüüsimisel oli rohkem andmepunkte, erinevus üksikutes punktides segas vähem ja seetõttu leiti ka rohkem seoseid.

Teiseks analüüsitavaks logiks oli nädala pikkune hägustatud IP-aadressidega tulemüüri logi *obfuscated-ssg-1week.log*. Kuna tegu oli reaalset võrku kaitsva tulemüüri logiga, siis andmelekete vältimise eesmärgil olid IP-aadressid hägustatud.

Varem oli teada, et igal tööpäeva hommikul pommitatakse tulemüüri UDP pakettidega paljudele sihtaadressidele lühikese ajavahemiku jooksul ning need mustrid tulid välja logisid uurides. Põhjuseks oli Skype'i ühenduste loomine kontaktideni. Kuna see muster oli teada enne logide analüüsimist, siis oli sellega hea testida, et kas sõnasagedusel põhinev analüüsimine suudab üles leida need sündmused.. Joonisel 15 on näha igal tööpäeva hommikul piike, mis on tekkinud tunni kaupa analüüsimisel. Needsamad piigid ongi need, mida taheti leida. Leitud piigid on stringidele 09 ja 11, mis on esmapilgul ootamatud. Uurimisel selgus, et need on loendurid agregeerimiskirjetest, mis näitavad, mitu korda mingit täpsemat kirjet on järjest esinenud. Seose leidmiseks sellest siiski piisas.

5 minuti kaupa analüüsimisel leiti veel, et sõnad *Null*, *self* ja *320001* on omavahel seotud. Lähemal uurimisel avastasid administraatorid, et tegu oli tulemüüri omadusega, millega administraatorid ei olnud varem kursis.

Samuti leiti huvitav korrelatsioon numbrite 53 ja 17 vahel. Number 17 tähistas UDP protokolli ning 53 DNS porti. Leiti, et neid kirjeid on teinud halvasti konfigureeritud masin, mis käis DNS päringuid tegemas valest kohast.

Tulemuste analüüsimisest järeldati, et positiivsete piikide leidmine aitab avastada huvitavaid sündmuseid. Negatiivsed piigid ei sisaldanud eriti infot ja ei osutunud kasulikuks.

Graafikutest on selgelt kasu visuaalsel võrdlemisel..

Korrelatsioonidest on vähe kasu, sest põhilised seosed, mida leitakse, on sama sündmuse erinevate sõnade vahel ja neid ei nimetaks huvitavateks seosteks.

Muudest logianalüsaatoritest kasutasime võrdluseks *logwatch*'i. Logwatch on levinud logide analüüsimisvahend, mis kasutab tuntud mustreid logidest olulise välja eraldamiseks ning oskab samaliigilisi kirjeid üheks kokku agregeerida. Tundmatud logiread, proovitud kasutajanimed, logimiskatseid teinud IP-aadressid jms tuuakse ükshaaval välja ja see nimekiri ei ole kuigi kergesti loetav. Logwatchi rakendati auth.log.4 logile [7]. Ootuspäraselt leidis ta tuntud mustritele vastavad sündmused, kuid ei osanud leida midagi, mille leidmist ei olnud otseselt ette valmis häälestatud. See oli piisav, et mustripõhiste logianalüsaatoritega rohkem mitte tegelda.

4.2 Edasiarendamise võimalused

Töö tulemusena valminud meetod suudab leida piike, mis vastavad inimeste poolt huvitavaks loetavatele sündmustele. Parema tulemuse saamiseks on võimalik muuta piigi definitsiooni (näiteks varieerides, mitme standardhälbe võrra peab keskmisest erinema või kasutades hoopis muid meetodeid) ja vaadata, kas leidub rohkem või täpsemaid tulemusi.

Seoste leidmiseks oleks võimalik ka lisada meetod, mis jälgib ainult seda, kas erinevate sõnade kohta leitud piigid asuvad samas ajavahemikus. Kui terves uuritavas ajavahemikus on mõlemas sõnas ainult üks piik ja see esineb sama koha peal, siis on sõnade vahel seos. Kui on mitu piiki ja neist ⅔ kattuvad, siis saab lugeda ka sõnad seotuks. Sellist meetodit tuleks veel katsetada, et teha kindlaks, kas sagedustabelid on ka graafikute pealt vaadates sarnased.

Korrelatsioonide leidmist võiks kasutada filtrina graafikuna esitatavate sõnade valimiseks, et eelnevalt välja filtreerida paljusõnalised klastrid ja saada neist üks sündmus.

Mõistlikumate piikide leidmiseks oleks võimalik sõnu klasterdada ka muul viisil. Selleks võiks kasutada Risto Vaarandi olemasolevat programmi SLCT [8]. SLCT abil saaks leida logist terved mitmesõnalised fraasid ning üksikute sõnade asemel need sagedusanalüüsile sisendiks anda. Nii võiks SLCT-d ja logAnalyserit koos kasutades leida rohkem huvitavaid sündmuseid, sest paljusõnalistest sündmustest jääks siis alles üks sündmus.

Võimalik, et ka programmi töö kiirust on võimalik parandada, sest seni uuritud logidega polnud töökiirus probleemiks ja algoritmide keerukuse minimeerimisele pole rõhku pandud.

5. Kokkuvõte

Käesoleva bakalaureusetöö eesmärk oli leida struktureerimata logidest kirjeid, mille iseloomu ega mustreid pole võimalik ette teada, aga mis erialaspetsialisti poolt peale vaadates tunduvad kasulikud või erandlikud. Eesmärgi saavutamiseks loodi meetod statistika vahendite abil sõnasageduste analüüsi tegemiseks ning selle katsetamiseks ja analüüsimiseks loodi programm logAnalyser.

Töö esimeses osas tutvustati, milliseid logisid analüüsimiseks valiti ja milliseid statistilisi meetodeid kasutati huvitavate sündmuste leidmiseks. Töö teises osas kirjeldati probleemi lahendamiseks kasutatud metoodikat. Kolmandas osas tutvustati programmi ülesehitust, projekti seadistamist enda masinasse ja käsurea kasutamisvõimalusi. Neljandas peatükis anti ülevaade tulemustest ja järeldustest. Samuti toodi välja, kuidas oleks võimalik seda programmi ja lähenemist logifailidele edasi arendada.

Töö käigus tehtud programmi logAnalyser abil kinnitati hüpotees, et sõnasagedusel põhineva logi analüüsi ja lihtsate statistiliste meetoditega on võimalik leida tundmatutest logidest huvitavaid sündmuseid.

6. Viited

[1] E-teatmik: IT- ja sidetehnika seletav sõnaraamat http://vallaste.ee

[2] C. Lonvick, “The BSD syslog Protocol” August 2001 https://www.ietf.org/rfc/rfc3164.txt [Kasutatud 24.03.2016]

[3] A. Tiits. Õppematerjal “Matemaatika põhivara 5. ja 6. klassile”. http://www.matemaatika.edu.ee/sisu/0012/index.html [Kasutatud 24.04.2016]

[4] K. Nõmmiste. Õppematerjal “Kirjeldavad statistikud”. http://opetaja.edu.ee/kyllin/statistics/kirjeldavad_statistikud.html [Kasutatud 24.04.2016]

[5] JFreeChart koduleht http://www.jfree.org/jfreechart/ [Kasutatud 09.05.2016]

[6] C.Beust, “JComannder” Juuli 2015 http://jcommander.org/ [Kasutatud 09.05.2016]

[7] Logwatch logianalüsaator https://sourceforge.net/projects/logwatch/ [Kasutatud 10.05.2016]

[8] R. Vaarandi. “Simple logfile clustering tool”. https://ristov.github.io/slct/ [Kasutatud 09.05.2016]

Lisad

1. Programmi logAnalyser lähtekood.

Programmi logAnalyser lähtekood, mis on saadaval ka aadressil https://github.com/KarlEST/logAnalyser. Sisaldab lähtekoodi, mis asub *src* kaustas, lisatud Java teeke, mis asuvad *lib* kaustas ja valmis tehtud logAnalyser.jar faili, mis asub *jar* kaustas.

2. Auth.log.4 logi ja 5-minuti tulemused.

Kokku pakitud auth.log.4 logi, millega tehti põhiline testimine ja 5-minuti tulemused, mis on saadud käsuga *java -jar logAnalyser.jar -p logiFailiAsukoht\auth.log.4 -t 5 --corr -i 25* asuvad kaustas *logid*.

3. Litsents

Mina, **Karl Lääts**,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
   Sõnasageduste põhine logianalüüs,
   mille juhendaja on **Meelis Roos**,
   1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **16.05.2016**