TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Tanel Peep 176277IDAR

Võtmete tuvastuse automatiseerimine koodivaramutest suurettevõtte näitel

Diplomitöö

Juhendaja: Edmund Laugasson
MSc

Tallinn 2020

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Tanel Peep

02.12.2020

Annotatsioon

Käesoleva diplomitöö eesmärgiks on luua suurettevõttele automatiseeritud lahendus, mis võimaldaks tuvastada koodivaramutesse talletatavaid võtmeid ning vähendaks ettevõtte jaoks võtmete lekkimisest tulenevat riski.

Töös kirjeldatakse koodivaramutes talletatavate ja lekkivate võtmete probleemi ning kogutakse lähtetingimused töös kirjeldatud suurettevõtte näitel. Lisaks võrreldakse töös olemasolevaid lahendusi, valitakse parim võimalik ning arendatakse käesoleva tööga juurde puuduvad lähtetingimused töö ulatuseks.

Arendusprotsessi käigus kirjeldatakse loodava lahenduse tehniline kirjeldus tehnoloogiate ja arhitektuuriga, teostuse protsess koos autori selgitustega ning kirjeldus töötava lahenduse testimisest ja tagasisidest.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 36 leheküljel, 6 peatükki, 8 joonist, 5 tabelit.

Abstract

Secrets detection automation from the code repositories in an example of a large company

The purpose of this thesis is to create an automated solution for a large company, which would enable secrets detection from the code repositories and thereby reducing the risk of secrets leakage for the company. Solution will support to add multiple repositories, automatically detecting new secrets from the code repository content changes and providing interface for detected secrets analysis and management.

To solve a problem with this thesis, it is necessary to formulate the problem in advance and then analyse problem together with solution methods. Therefore, separate chapters are described for problems, and methods before the substantive part.

Substantive part of the work is divided into four chapters. First chapter in substantive part of the work is analysis section, which is necessary to collect possibilities to detect secrets and gather requirements for the solution and thereby determine whether the existing solutions will meet the requirements. Second chapter in substantive part of the work is technical solution to interpret requirements into technical requirements, structure and design. Third chapter in substantive part of the work is solution implementation which will describe how the solution was implemented while taking technical requirements into account and consists of development process with author's explanations. Fourth chapter concludes the created solution, describes the compliance with the initial requirements, provides an overview of selective testing in an additional organisation, and provides overview of the further development of the solution.

The thesis is in Estonian and contains 36 pages of text, 6 chapters, 8 figures, 5 tables.

Lühendite ja mõistete sõnastik

IaC	Ingl. *Infrastructure as Code* - taristu koodina
Shannon Entropy	Ingl. *Shannon Entropy* – Shannoni entroopia
API	Ingl. *Application Programming Interface* – rakendusliides
RSA	Ingl. *Rivest-Shamir-Adleman* – avaliku võtmesüsteemiga krüpteerimise algoritm
ECC	Ingl. Elliptic-Curve Cryptography – avaliku võtmesüsteemiga krüpteerimise algoritm
Git	Hajutatud versioonihalduse süsteem
SVN	Tsentraliseeritud versioonihalduse süsteem
SAST	Ingl. *Static Analys Security Testing* – Staatiline koodi analüüsimise turvatestimise meetod, kasutatakse eelõige tarkvara jaoks, mis suudab nimetatud meetodit kasutades testida
Cron	Tarkvara, mis võimaldab luua ajastatud toiminguid
JSON	Ingl. *JavaScript Object Notation* – lihtsustatud andmevahetusvorming
YAML	Ingl. *YAML Ain't Markup Language* - Inimloetav andmevahetusvorming
SUS	Ingl. *System Usability Scale* – Meetod süsteemi kasutatavuse hindamiseks

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tänapäeval kasutatakse järjest enam lähtekoodi versioneerimiseks ja hoiustamiseks koodivaramuid. Koodivaramuid kasutavad tarkvaraarendajad ja süsteemihaldurid, näiteks tarkvara lähtekoodi hoiustamiseks või taristu paigalduse sätete ja skriptide hoiustamiseks. Selline koodivaramu kasutamine on loonud uue ohu kui koodivaramus talletatakse lisaks lähtekoodile ka salajasi võtmeid, mille hulgas on näiteks andmebaasi salasõnad, rakendusliidese võtmed, rakenduse kontod koos salasõnadega jne. Salajaste võtmete talletamine koodivaramus on risk, sest koodivaramust võivad võtmed olla kättesaadaval kolmandatele isikutele.

Käesoleva diplomitöö eesmärgiks on luua suurettevõttele automatiseeritud lahendus, mis võimaldaks koodivaramutest tuvastada talletatavaid võtmeid. Loodav lahendus vähendaks ettevõttele võtmete lekkimisest tulenevat riski, tuvastades võimalikult varakult võtmete salvestamist koodivaramusse.

Töö struktuur on üles ehitatud etapiliselt, mis on detailsemalt kirjeldatud peatükis „Metoodika“. Sisuline osa on jaotatud viite peatükki:

- „Probleem“ kirjeldab käesoleva tööga lahendatava probleemi.
- „Analüüs“ kogub võimalused ja lähtetingimused töö teostamiseks.
- „Tehniline lahendus“ kirjeldab tehnilise kirjelduse lahenduse loomiseks.
- „Teostamine“ kirjeldab autor töö teostamise lahenduskäiku koos selgitustega.
- „Tulemused“ kirjeldab lahenduse vastavust lähtetingimustele ja annab ülevaate lahenduse edasiarendusest.

Lisaväärtuse loomiseks ja suurema tagasiside saamiseks testitakse lahendust lisaks töös kajastatud suurettevõttele ka teises valiktestimise ulatuse jaoks sobilikus organisatsioonis. Ülevaade valiktestimisest on peatükis „Tulemused“

2 Metoodika

Probleemi lahenduseks on eelnevalt vaja kirjeldada probleem ning seejärel analüüsida probleemi koos lahendusmeetoditega. Seetõttu on töös välja toodud eraldi peatükk „Probleem“, mis kirjeldab koodivaramutes olevate võtmete lekkimise probleemi olemust.

Analüüsi peatükis on kirjeldatud probleemi lahenduseks vajalik taustanalüüs, et koguda võimalused ja lähtetingimused automatiseeritud võtmete tuvastuseks koodivaramutes ning võrrelda olemasolevaid võtmete tuvastuse lahendusi. Olemasolevaid lahendusi võrreldakse lähtetingimustele ning valitakse parim võimalik lahendus seatud probleemi lahenduseks. Lisaks kirjeldatakse, millistele lähtetingimustele valitud olemasolev lahendus vastab ning millistele lähtetingimustele lahendus ei vasta ja tuleb antud töö raames lahendada.

Tehnilise lahenduse peatükk kirjeldab tehnilise lahenduse, võttes arvesse töö analüüsi osas kirjeldatud lähtetingimused ja valitud olemasoleva lahenduse. Selles töö osas kirjeldatakse ka loodava arhitektuur ja disain.

Teostamise peatükk kirjeldab loodava lahenduse arendamist vastavalt eelnevates peatükkides kirjeldatud määratletud tingimustele ja tehnilistele nõuetele. Lisaks kirjeldab peatükk lahenduse ülesehitust, struktuuri ja realiseerimist koos autori põhjenduste ja selgitustega.

Tulemuste peatükk loob ülevaate loodud lahendusest ja kontrollib lahenduse vastavust töö analüüsi osas seatud tingimustele. Töö lisaväärtuse loomiseks ja laiendatud testimise saavutuseks on lisaks töös kajastatud suurettevõttele kaasatud testimisse ka teine skoobiks sobilik organisatsioon, mille tulemusi kirjeldatakse valiktestimise peatükis. Arvestades nõuete kontrolli ja testimise tulemusi, antakse ülevaade lahenduse edasiarendamise võimalustest.

3 Probleem

Koodivaramut kasutatakse enamjaolt lähtekoodi hoiustamiseks ning peamiselt tuleb koodivaramuga koos versioonihalduse tarkvara, mis pakub lisaks tavapärasele hoiustamisele võimalust koodi versioneerida, säilitada ajalugu ja töötada mitmel inimesel samaaegselt ühe projektiga. Koodivaramuid kasutatakse valdavalt tarkvaraarenduses, et hoiustada tarkvara lähtekoodi, kuid järjest enam on saanud koodivaramute kasutamine populaarseks ka taristu automatiseeritud paigalduse ja sätete koodi hoiustamiseks, mis on tuntud IaC (*Infrastructure as Code*) protsessina. Seoses suureneva koodivaramute kasutusega on tekkinud oht, kus tarkvaraarendajad ja süsteemihaldurid talletavad salajasi võtmeid koodivaramus asetsevas lähtekoodis.

Võtmed, mida tarkvaraarendajad ja süsteemihaldurid võivad koodivaramutesse talletada, on erinevad, näiteks nende hulgas võivad olla krüptograafilised võtmed, süsteemsed kontod koos salasõnade, API (*Application Programming Interface*) võtmed jms. Iga võtme lekkimise puhul võib mõju olla erinev, sõltuvalt sellest, milliseid ligipääsuõiguseid on võimalik võtmega saavutada. Ei ole oluline kui kriitiline on võti, ei tohiks siiski ühtegi võtit talletada koodivaramus olevasse lähtekoodi. Põhjuseid, miks talletatakse võtmeid lähtekoodi, on mitmeid, näiteks teadmatus, eksimus või isegi harjumus ja mugavus. Inimesi on küll võimalik teadvustada ja õpetada, kuid seejuures on oht alati olemas ning seetõttu tuleks rakendada samuti ka kontrollivaid meetmeid.

Üldjuhul on koodivaramud privaatsed, kuid avatud lähtekoodiga programmide koodi hoiustamiseks on koodivaramud avalikud ja kättesaadavad kõigile huvilistele. Avalikus koodivaramus võtmete hoidmine on suurema riskiga, sest lähtekood on kättesaadav suurele hulgale inimestele ning tõenäosus, et keegi selle leiab, on suurem. Lekkinud võtme kahjud võivad olla erinevad, mainekahjust kuni varalise kahjuni, sõltuvalt lekkinud võtme kriitilisusest. Näiteks lekkinud andmebaasivõtmega on võimalik ründajal kätte saada ettevõtte klientide isikuandmeid või näiteks isegi muuta tehingute andmeid, saades sellest varalist kasu. Lekkinud võtme potentsiaalne mõju sõltub süsteemist ja lekkinud võtme ligipääsuõigustest.

Aastal 2014 avastas Amazon avalikust GitHub koodivaramust tuhandeid AWS võtmeid ning teavitas sellest koodiprojektide omanikke. Kahjudest ei ole palju räägitud, kuid välja on toodud, et paljude klientide andmemahu arved suurenesid kordades, millest võib

järeldada massilisi andmelekkeid [1]. Aastal 2020 lekitas Amazoni enda töötaja kogemata avalikku koodivaramusse RSA privaatvõtmeid ja AWS-i rakendusliidese võtmeid, millede kasutusotstarve on täpselt teadmata [2]. Mõlemad juhtumid näitavad, et salajaste võtmete talletamine on probleem ning lekke kahjud võivad olla varjatud ja suured, mida ettevõtted ei taha maine kahjustumise tõttu avalikustada. Samuti ei ole ka alati võimalik kogu kahju hinnata rahalises väärtuses.

Võtmete lekkimine on probleem ning iga ettevõte, organisatsioon peaks koodivaramute kasutamisel ohtu teadvustama ja ennetama. Probleemi ennetamiseks on kaks põhilist meedet, mida peaks rakendama käsikäes, et minimaliseerida riski tekkivat riski. Esimeseks meetmeks on töötajate teadvustamine ja koolitamine, et töötajad ei talletaks võtmeid koodivaramus. Teine meede on koodivaramus olevate võtmete perioodiline seire ja piiramine. Käesolev töö ei uuri töötajate teadvustamise ja koolitamise meetmeid, vaid uurib ja keskendub koodivaramute võtmete automatiseeritud seire ja piiramise meetmetele.

4 Analüüs

Selles peatükis luuakse ülevaade võtmete tuvastamise võimalustest ja arhitektuurist, kogutakse lähtetingimused töö skoobiks ning analüüsitakse olemasolevaid lahendusi tuginedes lähtetingimustele. Analüüsi peatüki lõppväljundiks on lähtetingimused koodivaramus leiduvate võtmete automatiseeritud tuvastuse realiseerimiseks.

4.1 Võtmete tuvastamise võimalused

Võtmete tuvastamiseks on mitmeid võimalusi, igal ühel neist on omad puudused ja eelised. Valdavalt võib võtmete tuvastamist jagada kolme kategooriasse:

- entroopiapõhine võtmete tuvastamine
- võtme mustripõhine tuvastamine
- muud kaudsed tuvastuse võimalused

Järgnevad alapeatükid sisaldavad kirjeldust eeltoodud kategooriatele.

4.1.1 Entroopiapõhine tuvastamine

Matemaatik Claude Shannon on loonud juhuslikkuse valemi, mida kutsutakse ka Shannoni entroopiaks (*Shannon Entropy*), mis lihtsasti öeldes võimaldab arvutada sõne juhuslikkuse väärtuse, kasutades sõnes esinevate tähemärkide hulga esinemissagedust vastavas sõnes [3]. Eeldades, et võtmete määratlemiseks on kasutatud tuntud soovitust, kasutada piisavalt pikka ja täiesti juhuslikku tähemärkide kombinatsiooni, et teha võtmete arvamine arvajale piisavalt keeruliseks, siis on selliste võtmete tuvastamine võimalik Shannoni valemi põhjal. Shannoni valemi kasutamisel on üldjuhul juhuslikkuse väärtus tavaliste sõnade puhul madal ehk läheneb väärtusele 0, kuid võtmete puhul märkimisväärselt kõrgem ehk läheneb väärtusele 8 [4].

Kui võtmed on piisavalt pikad ja kasutatud erinevaid juhuslikke tähemärke siis Shannoni valem tuvastab tõhusalt kõrge juhuslikkuse väärtuse kuid sealhulgas võib koodiprojektist võtmete tuvastamisel tuvastada ka palju vale-positiivseid. Seda seetõttu, et koodis võidakse kasutada erinevaid väärtusi ja nimetusi, näiteks koosnevad nimetused mitmest

sõnast ja nende vahel puuduvad tühikud või kasutatakse koodiprojektis kahendkoodi objekte.

4.1.2 Mustripõhine tuvastamine

Mustripõhine võtmete tuvastamine on tõhus ainult siis kui võtmel on oma kindel ja eristatav struktuur. Näiteks mõned populaarsed rakendusliidese teenused lisavad juhuslikult tekitatud rakendusliidese võtmetele juurde eristatava signatuuri. Kõik Amazoni AWS (*Amazon Web Services*) API juurdepääsuvõtmete ID väärtused algavad tähtede kombinatsiooniga AKIA ja Google'i API võtmed algavad tähtede kombinatsiooniga AIza. Sellise signatuuri kasutamine ei vähenda võtme juhuslikkuse turvalisust, vaid võimaldab võtmete otsimist lihtsustada [5]. Sellise mustri põhjal otsimist on võimalik teostada regulaaravaldistega.

Mustripõhise võtmete tuvastuse puhul peaks valepositiivsete hulk olema väiksem kui entroopiapõhiste võtmete tuvastuse meetmetega, sest mustripõhise otsingu puhul otsitakse kindlat võtit, kus otsingu reegel on unikaalne ja loodud just kindla struktuuriga võtme tuvastamiseks.

4.1.3 Muud tuvastuse võimalused

Muid tuvastuse meetmeid kasutatakse tavaliselt täiendava sammuna enne või pärast mustri- ja entroopiapõhiste võtmete tuvastuse teostamist, et vähendada valepositiivsete hulka. Sarnast meetoditõhusust on tõestanud ka Põhja-Carolina Ülikooli teadusartikkel, kus esialgu esimene samm kasutati regulaaravaldise põhjal võtmete tuvastust ning seejärel leitud võtmete hulgale rakendati omakorda muid tuvastuse võimalusi, et vähendada valepositiivsete hulka [5].

Sellisteks tuvastusmeetmeteks võivad olla järgnevad lahendused:

- Failipõhine tuvastamine. Eesmärk tuvastada faile, mis võivad sisaldada salajasi võtmeid, näiteks sätetefailid, andmebaasifailid jne. Tuvastamine toimub failinime ja faililaiendi põhjal.

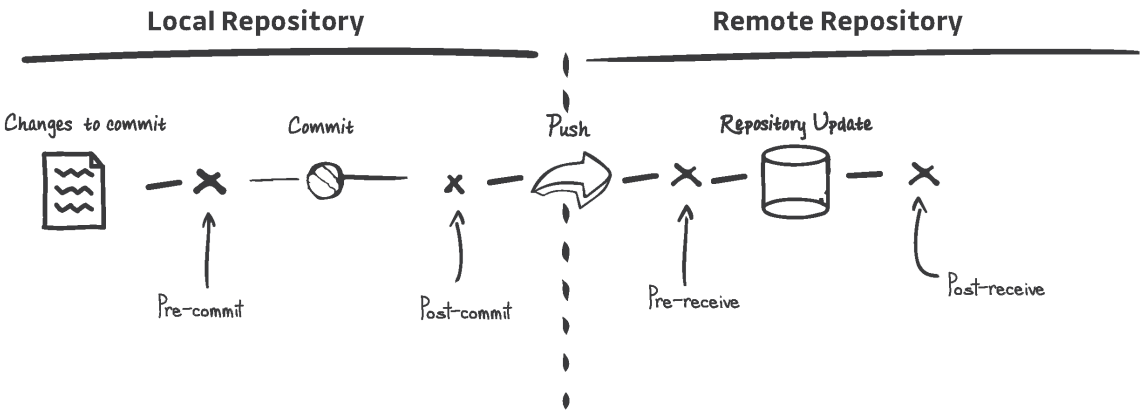
- Sõnapõhine tuvastamine. Eesmärk on filtreerida välja sõned, mis koosnevad keelelistest sõnadest, et vastupidiselt tuvastada ainult sõned, mis on juhuslikud. Sellise tuvastuse loomiseks läheb vaja vastava keele sõnastikku, et kontrollida keelelisust [5].

- Statistiliste mudelite põhine tuvastamine. Eesmärk on tuvastada sõne, mille tähemärkide kombinatsioon on statistiliselt eristatav. Näiteks tähepaaride järjestusindeks ja *Gibberish* indeks [4].

4.2 Tuvastuse arhitektuur

Arhitektuuriliselt on võtmete tuvastamiseks mitmeid lahendusi. Kõige ideaalsem lahendus on kliendipoolne arhitektuur, mille tulemusel on võimalik ennetada ja piirata võtmete talletamist lähtekoodi. Serveripoolse arhitektuuri puhul on tegemist rohkem järeltegevusega, kus lähtekoodi on juba võti talletatud ning seejärel on võimalik piirata või teavitada võtmete jõudmisest koodivaramusse.

Automatiseerimise vaatest on kõige efektiivsem kasutada *Git Hook* skripte. *Git Hook* on funktsionaalsus, mis lubab käivitada kohandatud skripte kindla tegevuse toimumisel [6]. Kohaliku koodivaramu *Git Hook* käivitatavad toimingud on *pre-commit* ja *post-commit* ning koodivaramus käivitatavad *Git Hook* toimingud on *pre-receive* ja *post-recieve*. Näited *Git Hook* skriptide rakendatavatest faasidest on toodud joonisel 1 [7].



Joonis 1 *Git Hook* skriptide näide [7]

Kohaliku koodivaramu puhul on tegemist kliendipoolse arhitektuuriga, sest nimetatud skriptid käivitatakse kliendi enda arvutis. Kaugkoodivaramu puhul on tegemist serveripoolse arhitektuuriga, kus skriptid käivituvad serveri poolel ning kliendil puudub võimalus skripte muuta. Järgnevates alapeatükkides on välja toodud kirjeldused kliendi- ja serveripoolse arhitektuuri kohta.

4.2.1 Kliendipoolne arhitektuur

Kliendipoolse arhitektuuri puhul kasutatakse kohaliku koodivaramu *Git Hook* skripte, mida käivitatakse kliendi arvutis. Automatiseerides võtmete tuvastamise *pre-commit* ja *post-commit* tegevustele, on võimalik anda kliendile vahetu tagasiside võtmete talletamisest kohalikku koodivaramusse. Neist esimese, *pre-commit* tegevusega, on võimalik ka piirata võtmete talletamist *commit* käsuga kohalikku koodivaramusse.

Selline arhitektuur on kõige kuluefektiivsem, kuna võtmete talletamine tuvastatakse juba arendamise staadiumis ning võtmed ei jõua koodivaramusse. Kuna kliendipoolse arhitektuuri puhul on võimalik kliendil endal skripti toimimist muuta siis põhineb lahendus kliendi usaldusväärsusel.

Tabelis 1 on välja toodud kliendipoolse arhitektuuri eelised ja puudused.

Eelised	**Puudused**
• Võtmete salvestamise piiramine kohalikku koodivaramusse • Kuluefektiivsem kui tuvastatakse võtmed enne lekkimist	• Klient saab muuta skripte, mis võimaldab vaigistada ja keelata võtmete tuvastust • Puudub keskne ülevaade koodivaramutele, kuna skripti väljund on ainult kliendi arvutis • Keskne skriptide ja reeglite uuendamine on keeruline • Valepositiivsete vaigistamine on keeruline ja häirib koodivaramu klientide tööd

Tabel 1 Kliendipoolse arhitektuuri eelised ja puudused

4.2.2 Serveripoolne arhitektuur

Serveripoolse arhitektuuri puhul on võtmete kontrollimine viidud kliendist eemale, kus võtmeid kontrollitakse koodivaramus *Git Hook* funktsionaalsusega või eraldiseisva serveriga, mis perioodiliselt käib koodivaramust kontrollimas võtmete esinemist.

Git Hook lahendus kasutab koodivaramu sündmusi, et käivitada võtmete tuvastuse skript. Nendeks sündmusteks on *pre-receive* ja *post-receive*. Mõlemad sündmustepõhised funktsioonid käivitatakse siis kui klient laeb koodivaramusse muudatusi, *pre-receive* käivitatakse koodivaramus vahetult enne kui muudatused salvestatakse aga *post-receive* käivitatakse koodivaramus vahetult pärast muudatuste salvestamist. Kui *pre-receive*

käivitatakse enne muudatuste salvestamist siis on võimalik selle funktsiooniga piirata võtme salvestamist koodivaramusse.

Keskne serverilahendus kasutab eraldiseisevat serverit, mis on seadistatud ja ajastatud ühenduma koodivaramusse, et kontrollida võtmete esinemist koodivaramu projektide muudatustest.

Git Hook ja keskse serverlahenduse peamine arhitektuuri erinevus on see, et *Git Hook* funktsionaalsust kasutades toimub võtmete tuvastamine koodivaramu serveris, kuid keskse serveri puhul toimub võtmete tuvastamine eraldiseisvas keskses serveris. Mõlema serveripoolse arhitektuuri lahenduse eelised ja puudused on välja toodud Tabel 2.

	Eelised	**Puudused**
Koodivaramu *Git Hook* lahendus	• Võtmete salvestamise piiramine koodivaramusse • Keskne võimalus hallata võtmete talletamist kõikides koodivaramu projektides • Klient ei saa õiguste puudumisel võtmete tuvastust vaigistada ega keelata • Võtmete tuvastusprotsesside ja reeglite keskne haldus • Võtmete tuvastuse protsess käivitatakse koheselt pärast igat koodivaramusse tehtavat muudatust	• Valepositiivsete vaigistamine on keeruline ja häirib koodivaramu klientide tööd
Keskse serveri lahendus	• Võtmete salvestamise keskne tuvastamine kõikides koodivaramu projektides • Võtmete tuvastuse protsesside ja reeglite keskne haldus • Valepositiivsete vaigistamine on lihtsam • Klient ei saa õiguste puudumisel võtmete tuvastust vaigistada ega keelata	• Puudub võimalus piirata võtmete salvestamist • Võtmete kontrollimine teostatakse perioodiliselt

Tabel 2 Serveripoolsete lahenduste eelised ja puudused

4.3 Vajadused ja nõuded

Antud töös kajastatav suurettevõte on mitme tuhande töötajaga rahvusvaheline ettevõte, mis tegeleb tarkvaraarendusega. Ettevõttes arendatakse kümneid erinevaid tooteid, kus iga toote kohta on mitmeid arendusmeeskondi. Sellise suurettevõtte puhul tähendab see seda, et arendusmeeskonnad erinevad üksteisest nii struktuuri ja protsesside vaatest kui ka kasutatavate tehnoloogiate poolest. Kuna töö keskendub automatiseeritud võtmete tuvastamisele koodivaramutest, siis tuleb arvestada, et antud suurettevõttes on kasutuses erinevaid koodivaramuid, näiteks GitLab, BitBucket, Stash jne.

Võtmete tuvastamisel peavad olema täidetud järgnevad tingimused:

- Regulaaravaldistepõhine tuvastamine.
- Shannoni entroopiapõhine tuvastamine.
- Populaarsemate rakendusliideste võtmete tuvastamine, näiteks Google, Amazon jne.
- Asümmeetriliste võtmete tuvastamine, näiteks RSA, ECC jne.
- Võtmete tuvastamine peab toimuma versioonihalduse logi (*git commit history*) põhjal.
- Võtmete tuvastamine peab olema võimalik versioonihalduse erinevatest harudest (*branch*).
- Võtmete tuvastamine peab olema võimalik muudatuste põhiselt, et tuvastada võtmeid sellest koodi osast, mida ei ole eelnevalt skaneeritud.
- Võtmete tuvastamisel ei tohi lähtekood olla kättesaadav kolmandatele isikutele.

Loodav lahendus peab vastama eelnevalt seatud tingimustele ja olema automatiseeritud nii, et võtmete tuvastamine oleks võimaldatud kõikidest kasutatud koodivaramutest ning suutma tuvastada uusi lisatavaid võtmeid.

Kuna füüsilisi koodivaramuid on umbes kümmekond ja koodiprojekte enam kui tuhat siis peab lahendus omama kasutajaliidest, et hallata tuvastatud võtmeid. Tuvastatud võtmete

hulgas võib kindlasti olla palju valepositiivseid, siis kasutajaliides peaks võimaldama kuvada leitud võtmeid koos asukohaga ning valepositiivsete ilmnemisel eemaldama leiud nimekirjast.

Samuti on automatiseerimise juures oluline saada teavitusi uute võtmete tuvastamisest koodivaramutes. Nõutavaks teavituste kanaliks on e-post kuid lisaväärtust annab ka teavituste saatmine meeskonnatöö rakendustesse.

4.4 Olemasolevate lahenduste võrdlus

Järgnevas peatükis on toodud ülevaade olemasolevatest lahendustest, mis suudavad tuvastada koodiprojektidesse salvestatud võtmeid. Iga lahenduse kohta on välja toodud plussid ja miinused ning võrdlustabel lähtetingimistega. Viimases alapeatükis on lahenduse valik olemasolevatest lahendustest.

Kuna lähtetingimustes oli nõue, et skaneeritav lähtekood ei tohi olla kättesaadav kolmandatele isikutele, siis on olemasolevate lahenduse hulgast on välja arvatud kommertslahendused, sest valdavalt tuleb nende puhul saata lähtekood teenusepakkuja juurde skaneerimiseks.

Võtmete tuvastamisel on kindel nõue, et toetatud oleks Shannoni entroopia ja regulaaravaldistepõhine võtmete tuvastamine. Seetõttu on antud töö võrdlusesse toodud ainult lahendused, mis toetavad mõlemaid võtmete tuvastuse võimalust. Samuti ka lahenduste liiasuse tõttu ei ole neid lahendusi välja toodud võrdluse tabelis. Lahendused, mis on töö autori poolt uuritud kuid siinsest võrdlusest välja jäetud, on GitRob, Git-secrets, repo-supervisor ja Crass.

4.4.1 TruffleHog

TruffleHog suudab tuvastada võtmeid git koodivaramust, otsides võtmeid kõikidest redaktsioonidest ja projekti harudest.

Trufflehog otsib võtmeid Shannoni entroopia ja regulaaravaldiste põhjal, vastavat otsingut teostab programm iga koodiprojekti haru redaktsioonidel kuni jõuab kõige vanema redaktsioonini. Shannoni entroopia põhisel tuvastamisel kasutatakse kõiki vähemalt 20 tähemärgiga tekstipõhiseid binaarobjekte, mis vastavad *base64* ja

hexadecimal tähemärkide komplektile. Ning kui nende hulgast leitakse kõrge entroopiaga sõnu siis kuvatakse tulemus ekraanile. [6]

4.4.2 Talisman

Talisman on programm, mis paigaldab *git hook* laienduse, et tuvastada potentsiaalseid võtmeid ja tundliku teavet viisil, et neid ei salvestataks enne vastavuse kontrollimist koodivaramusse.

Talisman töötab põhimõttes, kus iga uue koodi *git commit* ja *push* tegevusega kontrollitakse vastavas muudatuses võtmete esinemist ja seda vahetult enne *git commit* ja *push* käsu lõpetamist, et piirata võtme sattumist koodivaramusse. Positiivse leiu puhul tühistatakse tegevus ning kuvatakse raport tuvastatust. [7]

4.4.3 Gitleaks

Gitleaks on SAST (*Static Application Security Testing*) programm, mis suudab tuvastada koodivaramutes olevaid salasõnu, rakendusliidese võtmeid ja võtmetalonge (*key token*). Gitleaks ise ütleb, et nad tahavad luua programmi, mida on lihtne kasutada ja toetada kõiki arhitektuurilisi lahendusi, sealhulgas võtmete leidmist minevikus kui ka olevikus. [8]

4.4.4 Detect-secrets

Detect-secrets suudab tuvastada võtmeid lähtekoodist samuti nagu teised võtmete tuvastuse programmid, kuid arendajate endi sõnul fokuseerib nende programm rohkem ettevõtete vajadustele. Seetõttu on programmist arendatud kaks eraldi versiooni, üks neist kliendi- [9] ja teine serveripoolseks [10] võtmete tuvastuseks.

4.4.5 Võrdlus

Eelnevates peatükkides kirjeldatud võtmete tuvastuse lahenduste omadused on välja toodud tabelis Tabel 1.

	TruffleHog	**Talisman**	**Gitleaks**	**Detect-secrets**
SVN tugi	Ei	Ei	Ei	Ei
Git tugi	Jah	Jah	Jah	Jah
Entroopiapõhine tuvastus	Jah	Jah	Jah	Jah

Regulaaravaldise põhine tuvastus	Jah, 38 regulaaravaldist	Jah, 8 regulaaravaldist	Jah, 25 regulaaravaldist	Jah, 22 regulaaravaldist
Muud tuvastuse võimalused	Ei	Faili nimi, faili suurus	Ei	Ei
Valepositiivsete tuvastus	Ei	Ei	Ei	Ei
Skaneerib versioonihalduse kogu ajalugu	Jah	Jah	Jah	Jah
Skaneerib versioonihalduse ajalugu osadena	Jah	Ei	Jah	Jah
Kasutajaliides	Ei	Ei	Ei	Ei
Enda regulaaravaldiste lisamise võimalus	Jah	Ei	Jah	Ei
Kliendipoolne skaneerimine	Ei	Jah	Jah	Jah
Serveripoolne skaneerimine	Osaline[1]	Ei	Jah	Jah
SSH-põhine autentimine	Ei	Ei	Ei	Ei
Mitme varamu skaneerimise võimalus	Ei	Ei	Ei	Osaline[2]
Automatiseerimine	Ei	Ei	Ei	Osaline
Teavitused	Ei	Ei	Ei	Jah[3]
Aktiivne arendus	Ei	Jah	Jah	Jah

Tabel 3 Võtmete tuvastuslahenduste võrdlus

[1] Skaneerimine on võimalik kesksest serverist, kuid puudub võimekus koodivaramu *Git Hook* funktsionaalsuse kasutamiseks

[2] Detect-secrets pakub automatiseerimise ja mitme koodivaramu toetamiseks genereeritud skaneerimise käsklust, mida lisada *cron* ajastatud toimingutesse.

[3] Detect-secrets pakub võimalust saata teavitusi Sensu monitooringu rakendusse ja lisada enda kohandatud skripte

4.4.6 Lahenduse valik

Kõikide võrdluses olevate lahenduste puhul ilmnes lähtetingimustes kirjeldatud omadusi, kuid ükski lahendus ei sisaldanud kõiki vajalikke omadusi. Nendeks omadusteks on kasutajaliidese olemasolu, SSH-põhine autentimine, mitme varamu skaneerimise võimekus koos automatiseerimisega.

Lahenduse valiku juures üheks olulisemaks valikutingimuseks on võtmete tuvastuse võimekus. Lähtetingimustes kirjeldatud *Shannoni* entroopia ja regulaaravaldistepõhine võtmete tuvastamine on kõigil neljal lahendusel olemas, kuid erinevus on regulaaravaldiste reeglite mahus. TruffleHog omab kõige enam regulaaravaldiste reegleid, milleks on 38 ning kõige vähem reegleid on Talismani programmis, milles on 8 reeglit. Töö autor tugineb 2019. aastal RadareAPI teostatud eksperimendile, mis võrdles erinevate võtmetuvastuse programmide võimekusi koos valepositiivsete ja õigete positiivsete hulgaga ning tulemus näitas, et programmid, millel oli rohkem regulaaravaldiste reegleid, omas paremat tulemust võtmete tuvastamisel [13]. Sellest eksperimendist tuli välja, et TruffleHog on parima tuvastusvõimekusega vabavaraliste programmide hulgast. Tuginedes RadareAPI eksperimendile ja regulaaravaldiste arvule, peab autor ilma uue eksperimendita parimaks tuvastusvõimekusega lahenduseks TruffleHog programmi.

Muude valikutingimuste hulgast, mis ei ole seotud tuvastusvõimekusega, sai välistatud valikust koheselt Talisman, sest programm on suunatud ainult kliendipoolsele skaneerimisele. Kui võrrelda alles jäänud kolme lahendust nendel kriteeriumitel, mis ei ole seotud tuvastusvõimekusega, siis on kõik need lahendused üsnagi sarnased. Märkimisväärne erinevus on Detect-secrets programmil, mis ainukesena omab osalist võimekust mitme koodivaramu skaneerimisel ja automatiseerimisel. Põhjalikumalt uurides selgus, et programm pakub võimekust lisada programmi käsklus koos parameetriga ajastatud toimingutesse (ingl. *Cron*) iga koodivaramu eraldi, pakkudes seeläbi funktsionaalust mitme koodivaramu skaneerimisel ja automatiseerimisel. Sama võimekus on tegelikult nii TruffleHog kui ka Gitleaks programmil, kui lisada iga koodivaramu skaneerimise käsklus koos parameetritega ajastatud toimingutesse. Detect-secrets pakub lihtsalt automaatselt loodud väljundit, mida koheselt lisada ajastatud toimingutesse.

Tuginedes eeltoodud võrdlusele, valis autor parimaks võimalikuks lahenduseks TruffleHog programmi. Parimaks võimalikuks lahenduseks TruffleHog seetõttu, et programm ei suuda täita kõiki lähtetingimustele vastavaid kriteeriumeid, kuid mis tuleb täita selle töö raames.

4.5 Määratletud tingimused käesoleva töö ulatuseks

Olemasolevatest lahendustest valis töö autor parima võimaliku lahendusena TruffleHog programmi. Eelnevas peatükis selgus, et ükski lahendus ei võimalda kõiki funktsionaalsusi, mis olid kirjeldatud töö lähtetingimustes, seetõttu valis töö autor parima võimaliku lahenduse ning täiendab valitud lahendust, et täita töö lähtetingimused.

Lähtetingimused, mida TruffleHog ei täida ning tuleb antud tööga juurde täiendada:

- Kasutajaliides
    - Kasutajaliides peab võimaldama kuvada ja otsida võtmete skaneerimise tulemusi.
    - Lisaks tulemuste kuvamisele ja otsimisele peab olema võimaldatud valepositiivsete tulemuste markeerimine või kustutamine.
    - Kasutajaliides peab võimaldama loogiliselt struktureerida koodivaramuid, tooteid ja arendusüksusi.
    - Tulemustele ligipääs peab olema autoriseeritud ja ligipääs tulemustele rollipõhine
- Mitme koodivaramu tugi - programm peab suutma jälgida ja skaneerida mitmeid koodivaramuid ja koodiprojekte ning hoidma staatust eelneva skaneerimise kohta.
- Automatiseeritud skaneerimine - programm peab perioodiliselt ühenduma jälgitavatesse koodivaramutesse ning kontrollima uusi lisatud võtmeid.
- Teavitused - uute lisatavate võtmete korral tuleb saata selle kohta emaili teavitusi.
- SSH-põhine koodivaramu autentimine - koodivaramutesse ühendumisel peab programm võimaldama kasutada SSH-põhist autentimist.

5 Tehniline lahendus

Järgnev peatükk keskendub lahenduse tehnilisele kirjeldusele. Peatükis kirjeldatakse, kuidas lähtetingimused muudetakse tehnilisteks kirjelduseks ning kuidas süsteem peaks toimima.

5.1 Koodivaramute määratlemine

Selleks, et hoida koodivaramute sätteid, olekut ning skaneeritavaid koodiprojekte, tuleb kogu see teave määratleda ja talletada. Samuti peaks lahendus olema lihtsasti paigaldatav ja käivitatav käsurealt. Seetõttu on autor otsustanud, et koodivaramute määratlemiseks ei looda eraldi andmebaasi, vaid kogu vajalik teave talletatakse vastavates sätete- ja olekufailides.

Töös kajastatud suurettevõttes on kasutusel kümneid koodivaramuid ning kokku on üle tuhande koodiprojekti ning samuti on koodivaramud ja projektid erineva kasutusega. Näiteks võivad koodiprojektid sisaldada tarkvara lähtekoodi, paigalduse skripte, konfiguratsiooni jms. Seetõttu on oluline, et koodiprojekte ja -varamuid oleks võimalik loogiliselt rühmitada. Loogiline rühmitamine tähendab seda, et ühe füüsilise koodivaramu koodiprojekte on võimaik rühmitada mitmeks loogiliseks koodivaramuks.

Näiteks üks arendusmeeskond arendab mitut toodet kuid omab ühte koodivaramut koos kõikide toodete tarkvara lähtekoodi ja paigalduse skriptidega, mis on eraldatud mitmeteks koodiprojektideks. Loogiline rühmitamine võimaldaks luua loogilise koodivaramu ühe toote tarkvara lähtekoodi jaoks ja lisada sinna kõik seotud koodiprojektid. Loogiline rühmitamine lubaks samuti rakendada sätteid vastavalt koodivaramule, mis rakenduks kõikidele sealses varamus olevatele koodiprojektidele.

Loodava lahendusega on võimalik koodiprojekte loogiliselt rühmitada üheks koodivaramuks, vastavalt otstarbele ning rakendada sätteid loogilise koodivaramu põhiselt.

5.2 Tulemuste haldus

Skaneerimise tulemuste maht võib koodiprojektides erineda. Kui koodiprojektides kasutatakse näiteks binaarobjekte või suure juhuslikkusega sõnu, siis valepositiivsete tulemuste hulk on palju suurem. Samuti kui skaneerida suures koguses koodiprojekte siis tulemuste hulk mitmekordistub. Selleks, et suures koguses tulemusi vaadata ja analüüsida läheb vaja kasutajaliidest. Seetõttu peab lahendus pakkuma kasutajaliidest, kus oleks võimalik tulemusti vaadata, otsida ja seeläbi analüüsida.

Lahendusele annaks rohkem lisaväärtust kui kasutajaliides pakub järgnevaid vajalikke protsesse:

- Tulemustele oleks võimalik määrata olekuid, näiteks uus/avatud, valepositiivne, kinnitatud ja lahendatud. Olek "uus" näitab, et tegemist on uue tulemusega ning ei ole veel kinnitatud, kas tulemus on valepositiivne või mitte. Olek "valepositiivne" näitab, et tegemist on valepositiivsega. Olek "kinnitatud" ütleb, et tegemist on asjakohase tulemusega. Kui tulemus parandatakse ehk võti eemaldatakse koodist, siis määratakse tulemuseks "lahendatud".

- Tulemused peaks olema rühmitatud vastavalt loogilisele koodivaramule.

- Kasutajaliides võimaldaks määrata igale loogilisele koodivaramule eraldi ligipääsuõiguseid. Seeläbi oleks võimalik igale koodivaramu omanikule anda ligipääsuõigused, et näha enda vastuses olevate varamute olekut ja vajadusel reageerida.

Olemasolevatest lahendustest, mis pakuks rakenduste turvanõrkuste haldusliidest, on vabavaralistest lahendustest saadaval DefectDojo. See on rakenduste turvanõrkuste haldusliides, mis võimaldab importida tulemusi teistest turvatestimise tööriistadest, hallata tulemusi, luua pileteid, seadistada meetrikaid ning raporteerida tulemusi [14].

Analüüsides eelpool kirjeldatud protsesse tulemuste haldamisel ning võrreldes neid DefectDojo võimalustega, selgus, et DefectDojo toetab kõiki võimalusi hallata automatiseeritud TruffleHog testimisega saadud tulemusi.

5.3 Automatiseerimine

Automatiseerimisel on oluline, et programm käiks perioodiliselt koodivaramus, kontrolliks koodiprojektidesse lisatud uusi võtmeid ning teavitaks uutest leidudest. Automatiseerimisel oleks kindlasti efektiivsem kui perioodilise kontrollimise asemel koodivaramu ise teavitaks uutest muudatustest läbi *Git Hook* funktsiooni ning seejärel alustatakse võtmete skaneerimist, kuid mahukuse tõttu jääb koodivaramu poolt algatatud skaneerimise protsess praeguse töö ulatusest välja.

Kogu protsess pärast koodivaramute ja -projektide määratlemist peab olema automatiseeritud. See tähendab, et programm peab lugema sätted koos koodivaramute ja -projektidega, seejärel küsima koodivaramust koodiprojektide muudatused, skaneerima muudetud koodi ning laadima tulemused DefectDojo tulemuste haldusliidesesse. Muudatuste põhisel skaneerimisel peab lahendus omama iga koodiprojekti kohta olekut, vastasel juhul hakkaks lahendus tuvastama samu tulemusi.

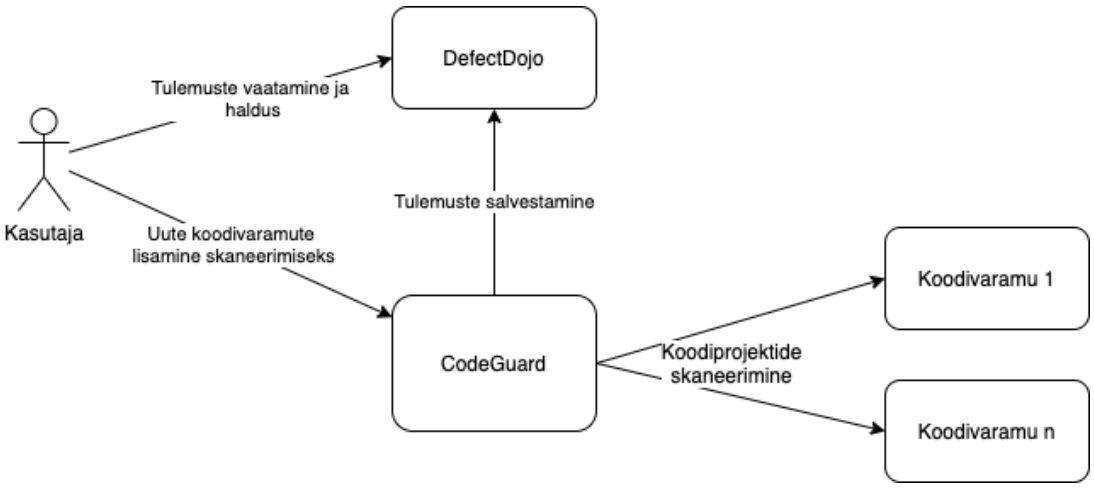
Koodiprojekti oleku hoidmiseks peab pärast igat automatiseeritud perioodi talletama viimase skaneeritud versioonihalduse redaktsiooni räsi (*git commit hash*). Skaneerimine ehk võtmete tuvastamine koodiprojektist, toimub redaktsioonide põhiselt, alates uuemast kuni kõige vanemani. Pärast igat skaneerimist tuleb salvestada esimene skaneeritud redaktsioon ning järgmisel skaneerimise perioodil tuleb skaneerimine peatada eelnevalt salvestatud redaktsioonini jõudmisel. Sellisel viisil on võimalik skaneerida koodiprojektide redaktsioone muudatuste põhiselt ning vältimaks, et ei skaneerita rohkem ega ka vähem redaktsioone kui vajalik. Viimase skaneeritud versioonihalduse redaktsiooni räsi salvestatakse koodivaramute määratlemise peatükis kirjeldatud olekufaili.

Vältimaks liiga tihedat või harva automatiseeritud skaneerimist, peab lahendus võimaldama seadistada skaneerimise perioodilisust. Perioodilisuse all on nimetatakse ühte tsüklit kui programm suudab skaneerita kõiki määratletud koodivaramute projekte.

5.4 Lahenduse arhitektuur

Loodava lahenduse arhitektuuris on kaks eraldi koostisosa. Esimeseks osaks on DefectDojo, mida kasutatakse tulemuste haldusliidesena ja teine on CodeGuard, mis arendatakse käesoleva tööga, et automatiseerida võtmete tuvastust koodivaramutest.

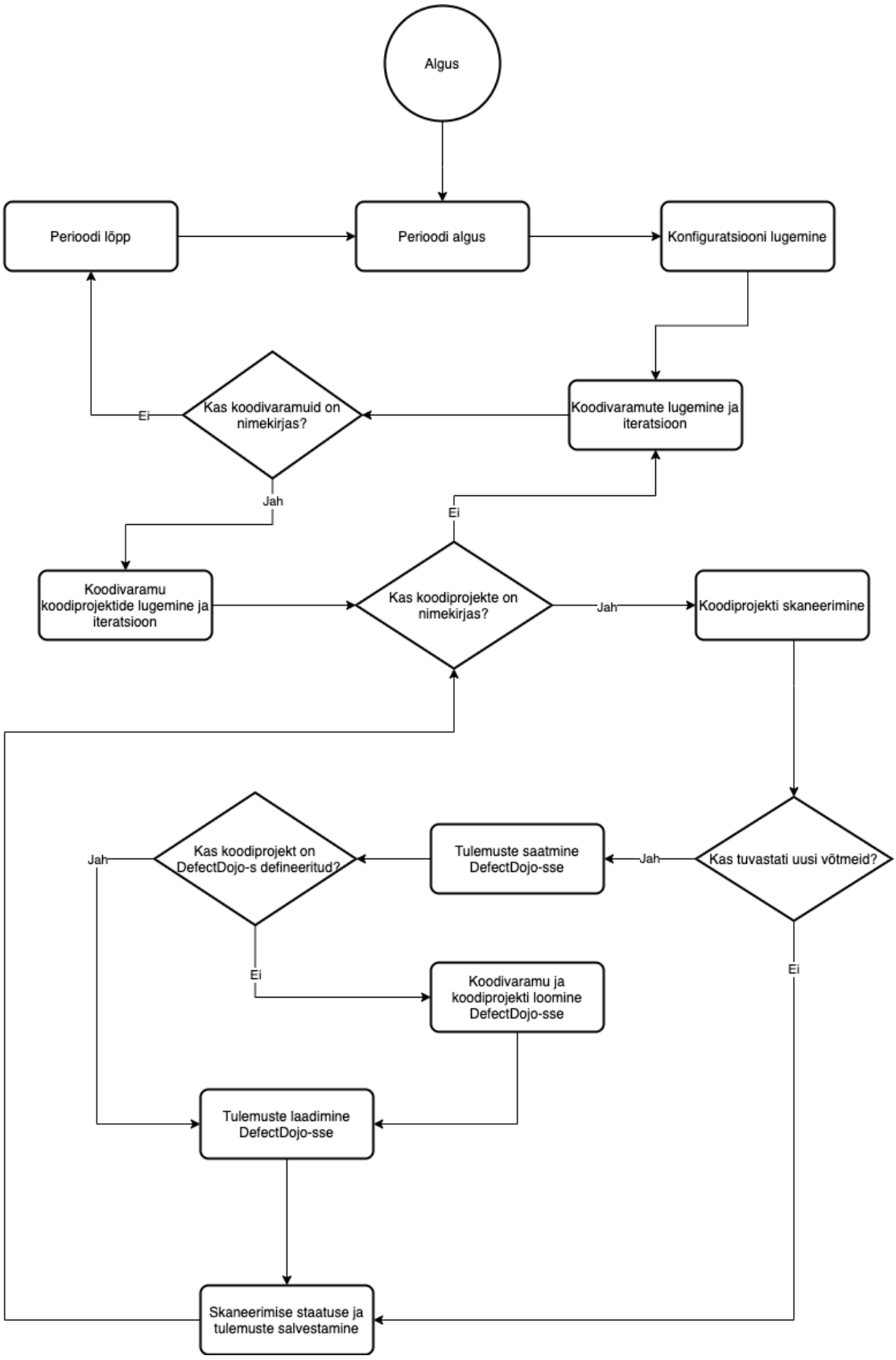
CodeGuard kasutab TruffleHogi võtmete tuvastuse programmi skaneerimise loogikat ning ülejäänud lähtetingimused, mida TruffleHog ei võimaldanud töö analüüsis osas, arendatakse CodeGuard programmi. Automatiseeritud koodivaramute võtmete tuvastuse arhitektuur on nähtav joonisel Joonis 2.



Joonis 2 Automatiseeritud koodivaramute võtmete tuvastuse arhitektuur

CodeGuardi programmis tuleb kirjeldada kõik koodivaramud ja koodiprojektid, mida soovitakse skaneerida ning programm ise pöördub kirjeldatud koodivaramutesse ja sealsetesse koodiprojektidesse skaneerimaks lekkinud võtmeid. Lekkinud võtmete tulemused saadetakse DefectDojo rakendusse, kus on kasutajal võimalik tulemusi vaadata ja hallata. CodeGuard programmis leitud tulemused saadetakse DefectDojo andmebaasi, kasutades DefectDojo rakendusliidest. Antud lahenduse juures ei pea eelnevalt seadistusi DefectDojo rakenduses tegema, mis võimaldab kasutada juba olemasolevat rakendust, kirjeldades sätteid rakendusliidese aadressi ja võtme abil. CodeGuard loob automaatselt DefectDojo rakendusse vastava struktuuri koodivaramute ja -projektide jaoks.

Lihtsustatud automatiseeritud protsess koos skaneerimise ja tulemuste saatmisega DefectDojosse on kirjeldatud joonisel Joonis 3.



Joonis 3 Automatiseeritud võtmete tuvastuse protsessi diagramm

6 Teostamine

Teostamise peatükk kirjeldab loodava lahenduse arendamist vastavalt eelnevates peatükkides kirjeldatud lähtetingimustele ja tehnilistele nõuetele. Lisaks kirjeldab peatükk lahenduse ülesehitust, struktuuri ja teostamist koos autori põhjenduste- ja selgitustega.

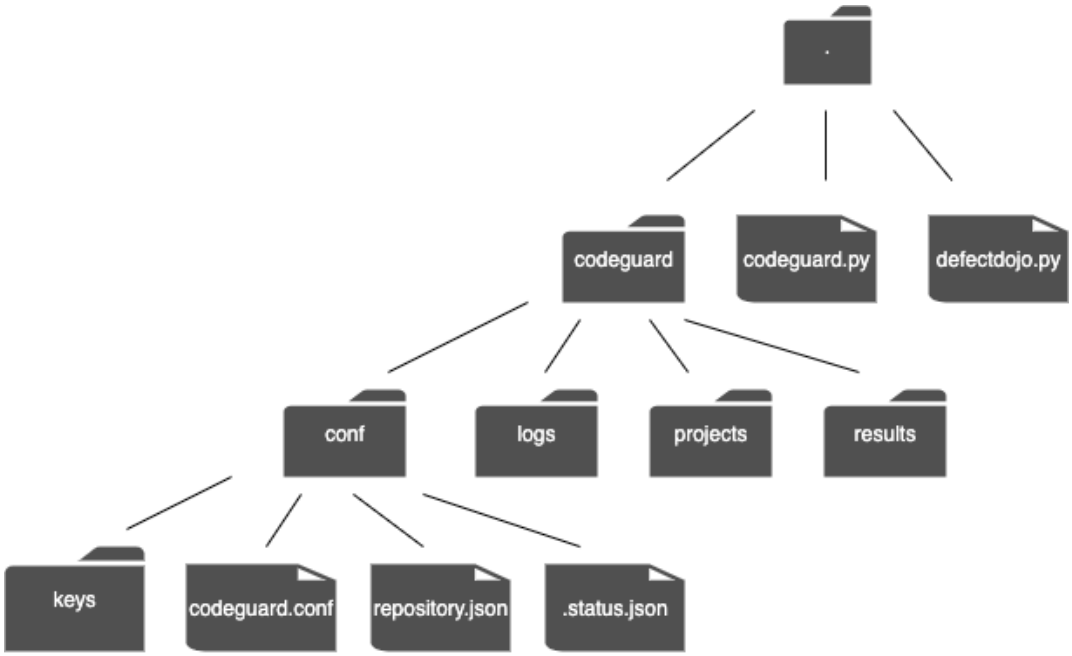
6.1 Lahenduse ülesehitus

Loodud lahendusele on autor andnud nimeks CodeGuard. CodeGuard on sisuliselt TruffleHog programmi edasiarendus, millele on koodivaramus sisalduvate võtmete tuvastusele lisaks juurde arendatud mitme koodivaramu tugi ning võimekus automatiseerida tuvastust ja edastada tulemusi DefectDojo turvanõrkuste haldusrakendusse.

Lahenduse failipuu on edasiarendus, mis on loogiliselt ülesehitatud ja nähtav jooniselt Joonis 4. Lahenduse juurkataloogis asetseb kataloog *codeguard* ning failid *codeguard.py* ja *defectdojo.py*. Kataloogis *codeguard* asetsevad kõik rakenduse toimimiseks vajalikud failid. Fail *codeguard.py* on rakenduse programmifail ja seal asetseb kogu programmi loogika. Fail *defectdojo.py* on DefectDojo teek, et kasutada sealse rakendusliidese võimekust.

Kataloogis *codeguard* asetsev *conf* kataloog on rakenduse sätete hoidmiseks, *logs* kataloog on rakenduse logi jaoks, *projects* kataloog on skaneeritavate koodiprojektide hoidmiseks ning *results* kataloog on skaneerimise tulemuste salvestamiseks.

Kataloogis *conf* asetseb kolm sätete faili – *codeguard.conf*, *repository.json* ja *.status.json*. *Codeguard.conf* on rakenduse üldine sätete fail, mis sisaldab rakenduse toimimiseks vajalikke sätteid, näiteks parameetrid DefectDojo rakendusliidesega integreerimiseks, skaneerimise perioodi muutmiseks jms. *Repository.json* fail on skaneeritavate koodivaramute ja -projektide kirjeldamiseks. *Status.json* on punktiga algav peidetud fail, mille eesmärgiks on hoida skaneerimise olekut, et saavutada muudatustel põhinevat skaneerimist. Kataloog *keys* on vajalik privaatvõtmete hoidmiseks, mida kasutatakse SSH-põhise autentimismeetodiga kaitstud koodivaramutesse ühendumiseks.



Joonis 4 CodeGuard lahenduse failipuu

6.2 Lahenduse arendus

Eelnevas peatükis kirjeldatult on CodeGuard rakendus, mis baseerub Trufflehog rakendusele ning millele on edasi arendatud mitme koodivaramu tugi, automatiseerimise võimekus ja tulemuste edastamine DefectDojo turvanõrkuste rakendusse. Kuna rakendus põhineb TruffleHogi rakendusel ja kasutab sealset skaneerimise loogikat, mis on juba varasemalt kirjutatud Pythoni programmeerimiskeeles, siis otsustas töö autor mitte hakata tõlkima funktsionaalsust ühest programmeerimiskeelest teise, vaid jätkata arendust samaskeeles. Loodud rakendus on arendatud Pythoni versioonile 3.7.

Loodud rakendus on edasiarendus TruffleHog rakendusele, siis on oluline välja tuua, milline rakenduse osa on autori enda panus ja milline mitte. Seetõttu on tabelis Tabel 4 välja toodud autori enda panus uue rakenduse loomisel. Tabelis on välja toodud loodud rakenduse kõik meetodid ning jaotatud vastavalt, et näidata autori enda panust. Nimetatud meetodid on jaotatud kolme kategooriasse – kasutatud meetodid muutmata kujul, kasutatud ning täiendatud meetodid ja täiesti uued meetodid.

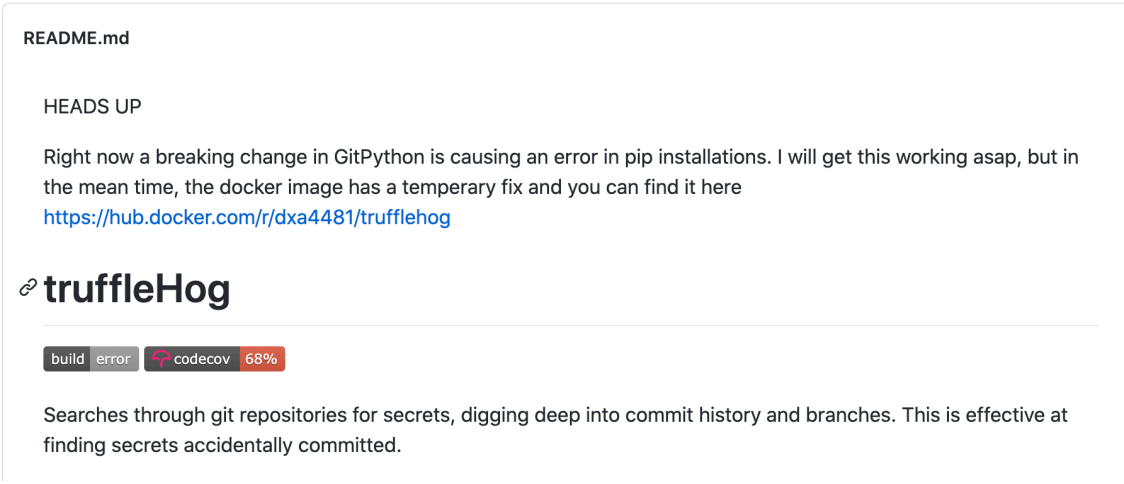
Kasutatud muutmata kujul meetodid	**Kasutatud ja täiendatud meetodid**	**Täiesti uued meetodid**
• shannon_entropy • get_strings_of_set	• find_strings • clone_git_repo	• upload_to_defectdojo • update_repository

• path_included • clean_up • b_color • del_rw	• print_results • find_entropy • regex_check • diff_worker • handle_results	• update_status • update_branch • get_scan_branches • scan_branch • scan_project • init_scan • run_job

Tabel 4 TruffleHog rakenduse täienduste skoop

Arenduse mahult on sisaldab TruffleHog rakendus 365 koodirida ning CodeGuard 951 koodirida. 236 koodirida TruffleHog rakendusest on muutmata kujul kasutatud CodeGuard rakenduses. Ülejäänud 715 koodirida on CodeGuard rakenduses uus osa, millest maksimaalselt 129 koodirida on muudetud osa TruffleHog rakendusest. Ülevaatlikult võib hinnata, et CodeGuard rakenduses on 13,5% muudetud, 25% muutmata ning 61,5% uut osa.

TruffleHog rakendusele ei ole olnud aktiivset arendust ligikaudu 2 aastat, vahepealsel ajal on küll teostatud rakendusele mõningaid parandusi, kuid need on olnud enamjaolt seotud rakenduse sõltuvustega. Praegusel hetkel TruffleHog rakendust paigaldada ei ole võimalik, sest rakendusel on probleeme teekide sõltuvustega. Rakenduse GitHubi arendusprojekti lehel on teade, et rakenduse paigaldusega on probleeme ning selle parandamisega tegeletakse niipea kui võimalik. Teade on lisatud 9 kuud tagasi kuid siiani ei ole probleem parandust leidnud. Selleks, et TruffleHogi funktsionaalsust kasutada CodeGuard rakenduses, tuli seal parandada teekide sõltuvused. Ekraanitõmmis TruffleHog teade paigalduse probleemist on näha joonisel Joonis 5



Joonis 5 TruffleHog projekti teavitus paigalduse puudustest

Lisaks teekide sõltuvuse probleemile ilmnes TruffleHogi rakenduses ka mitmeid muid probleeme, mida tuli töö autoril parandada CodeGuardi rakenduses. Näiteks ei toiminud *master* arenduseharu skaneerimine ning samuti muudatustel põhinev skaneerimine oli vigane. Lisaks on praegusel hetkel TruffleHogi arendusprojekti GitHub lehel rakenduse kohta raporteeritud 74 avatud küsimust[1], millest suur osa on rakendusega seotud probleemid. Mitmed probleemid on ka korduvad, sest keegi ei ole probleemide registreerimise ja nimekirjaga tegelenud.

TruffleHog rakendus on avaldatud GPL 2.0 litsentsiga, mis lubab kasutada loodud tarkvara muuta, levitada ning kasutada privaatselt ja kommerts eesmärkidel. Kuna CodeGuard rakendus põhineb TruffleHog rakendusel ning rakenduse lähtekood on avaldatud ka avalikuks kasutamiseks siis CodeGuard rakendus on avaldatud GPL 2.0 litsentsiga ning viidatud TruffleHog rakendusele.

CodeGuardi rakenduse arendusega on kasutatud TruffleHogi rakenduse funktsionaalsust ning kõik teadaolevad probleemid on autoril CodeGuardi rakenduses parandatud. Kõik kopeeritud funktsioonid ja meetmed on parandatud ning muudetud CodeGuardi rakenduse jaoks sobivaks ning õigesti viidatud TruffleHogi rakenduse funktsioonide kasutuse kohta. Samuti ei ole TruffleHogi rakendus eraldiseisvana seotud CodeGuardi rakendusega, mis tähendab, et kõik arendused, mis tehakse eraldi TruffleHogi rakendusele, ei ole kuidagi sõltuvad CodeGuardi rakendusest.

[1] https://github.com/dxa4481/truffleHog/issues

6.3 Mitme koodivaramu toe teostamine

TruffleHogi programm ise on käsurea rakendus, millega on võimalik tuvastada võtmeid ühest koodivaramu koodiprojektist. CodeGuardi rakendusele on juurde lisatud mitme koodivaramu tugi, et oleks võimalik ühe käivitusega tuvastada võtmeid mitmetest erinevatest koodivaramutest ja -projektidest.

Ühekordsel võtmete skaneerimisel on oluline ette anda koodivaramu, sealne skaneeritav koodiprojekt ning *git* redaktsioon ehk kui kaugele redaktsioonide ajaloos skaneerimine toimub. Need kolm väärtust on olulised iga koodivaramu projekti skaneerimisel. Selleks, et mitme koodivaramu tuge saavutada, tuleb iga skaneeritava koodiprojekti jaoks talletada need väärtused. Koodiprojektide skaneerimiseks vajalikud väärtused salvestatakse eraldi faili nimega *repository.json,* mille andmestruktuur on kirjeldatud JSON-vormingus. Töö autor otsustas kasutada andmete talletamiseks failipõhist lähenemist, sest käsurea rakenduse puhul on andmete hoidmine failides mugavam ja lihtsam. Samuti puudub rakendusel kasutajaliides, mis võimaldaks mugavalt andmete salvestamist ja taasesitamist andmebaasist.

Skaneerimiseks vajalike väärtuste talletamiseks on oluline struktuur, et võimaldada lisasätete määramist vastavalt koodivaramule või -projektile. Samuti oli töö tehniliseks lähtetingimuseks võimalus loogiliselt rühmitada koodiprojekte. Loogilise rühmitamisega on võimalik koodiprojekte üheks loogiliseks koodivaramuks siduda olenevalt sellest, kas need koodiprojektid on erinevates koodivaramutes või mitte.

Koodivaramute ja -projektide kirjeldusel on lähtutud selle seotusest koodivaramuga. Samuti on koodivaramud ja -projektid kirjeldatud sõnastik (ingl. *dictionary*) andmetüübina, mis tähendab, et vastavas sõnastiku objektis on kirjeldatud iga elementi kirjeldada võtme ja väärtusega. Sõnastiku puhul peab olema võtme väärtus unikaalne kõikide elementide hulgas. Sõnastiku kasutamine aitab samuti vältida korduvaid koodivaramuid ja nendes olevate projektide kordumist. Koodivaramute ja -projektide kirjelduse struktuur on nähtav jooniselJoonis 6. Joonisel kirjeldatud *parameters* ehk väärtused, on lisasätted, mida on võimalik lisada koodivaramutele ja -projektidele. Näiteks väärtused, mis on lisatud koodivaramule, rakenduvad kõikidele seal asetsevatele koodiprojektidele ning väärtused, mis on lisatud koodiprojektile, rakendub ainult kirjeldatud koodiprojektile.

Parameetrid, mida on võimalik eraldi seadistada, on praegusel hetkel SSH-põhiseks autentimiseks vajalik seadistus ja koodiprojekti asukohta kirjeldav link *(URL)*. Kui koodivaramu on privaatne, siis on SSH-põhise autentimise seadistus kohustuslik. Koodiprojekti asukohta kirjeldav link on alati vajalik, et rakendus saaks skaneerida koodivaramus asetsevat koodiprojekti.

```
{
	"repository1_name":
	{
		<parameters>
		"projects":
		{
			"project1_name":
			{
				<parameters>
			},
			"project2_name":
			{
				<parameters>
			}
		}
	},
	"repository2_name":
	{
		<parameters>
		"projects":
		{
			"project1_name":
			{
				<parameters>
			}
			"project2_name":
			{
				<parameters>
			}
		}
	}
}
```

Joonis 6 Koodiprojektide ja -varamute kirjelduse struktuur

6.4 Automaatika teostamine

Võtmete tuvastamine on automatiseeritud nii, et programmi tsükkel käivitatakse perioodidena ning igal perioodil toimub võtmete tuvastamine koodivaramute projektidest eelneva käivitatud perioodi muudatuste põhjal. Programmi üheks perioodiks on aeg kui

programm kontrollib võtmeid kõikidest kirjeldatud koodivaramu projektidest ühe korra. Aeg, mis jääb kahe perioodi vahele, on programmi ajavahemik. See määrab aja kui kaua peab programm ootama uuesti käivitust. Vaikimisi on programmi ajavahemik 1 minut, mis tähendab, et programm ootab pärast perioodi 1 minuti enne kui käivitatakse uus programmi periood. Programmi ajavahemikku on võimalik eraldi rakenduse sätetes seadistada.

Muudatustel põhineval automatiseerimisel tuleb talletada iga koodiprojekti skaneerimise olek. Seetõttu on rakendusse loodud eraldi *.status.json* fail, mis kirjeldab iga koodiprojekti olekut. Koodiprojekti olek salvestatakse iga perioodi lõpus ning laetakse iga perioodi alguses. Kuna olek on salvestatud faili, siis programmi katkestuse tõttu ei kao programmi seis.

Olekufaili struktuur on sarnaselt *repository.json* failile kirjeldatud JSON-andmevormingus ja objektid kirjeldatud sõnastik-andmetüüpidega. Peamine erinevus on koodivaramute ja koodiprojektide kirjeldamisel. Olekufailis ei ole kirjeldatud koodivaramute ja -projektide vahelist struktuuri, mistõttu on kõik koodiprojektid samal tasemel. Vältimaks sama nimega koodiprojekti esinemist, on nime ette lisatud ka koodivaramu nimi, sellisel juhul ei ole võimalik korduv koodiprojekti nime esinemine. Iga koodiprojekti *branches* alamelemendis on kirjeldatud koodiprojekti harud, et hoida olekut iga skaneeritud koodiprojekti haru seisu kohta. Koodiprojekti haru parameetriteks on *latest_commit_hash* ja *results_file*. Näitaja *latest_commit_hash* kirjeldab viimase skaneerimise perioodis skaneeritud redaktsiooni ja *results_file* kirjeldab skaneerimise tulemuste faili asukohta ja nime. Olekufaili struktuuri koos elementidega on kirjeldatud joonisel Joonis 7.

```
{
	"<repository1>_<project1>":
	{
		"branches":
		{
			"branch1":
			{
				<parameters>
			},
			"branch2":
			{
				<parameters>
			}
		}
	},
	"<repository2>_<project1>":
	{
		"branches":
		{
			"branch1":
			{
				<parameters>
			}
		}
	}
}
```

Joonis 7 Olekfaili kirjelduse struktuur

Olekufail *.status.json* on peidetud fail, mille tekitab rakendus ise ning mida ei tohiks kasutaja muuta. Rakendus kasutab olekufaili loomiseks koodivaramute kirjeldusfaili *repository.json*. Rakenduse käivitamisel loetakse kõik koodivaramud ja koodiprojektid koos sätetega ning lisaks loetakse olekufail eelmise skaneerimise perioodi kohta. Muudatustel põhinevaks automatiseerimiseks on kõige olulisem iga koodiprojekti haru viimase skaneerimise redaktsiooni viide, et järgmisel korral skaneerida ainult neid redaktsioone, mis on uuemad kui olekufailis kirjeldatud.

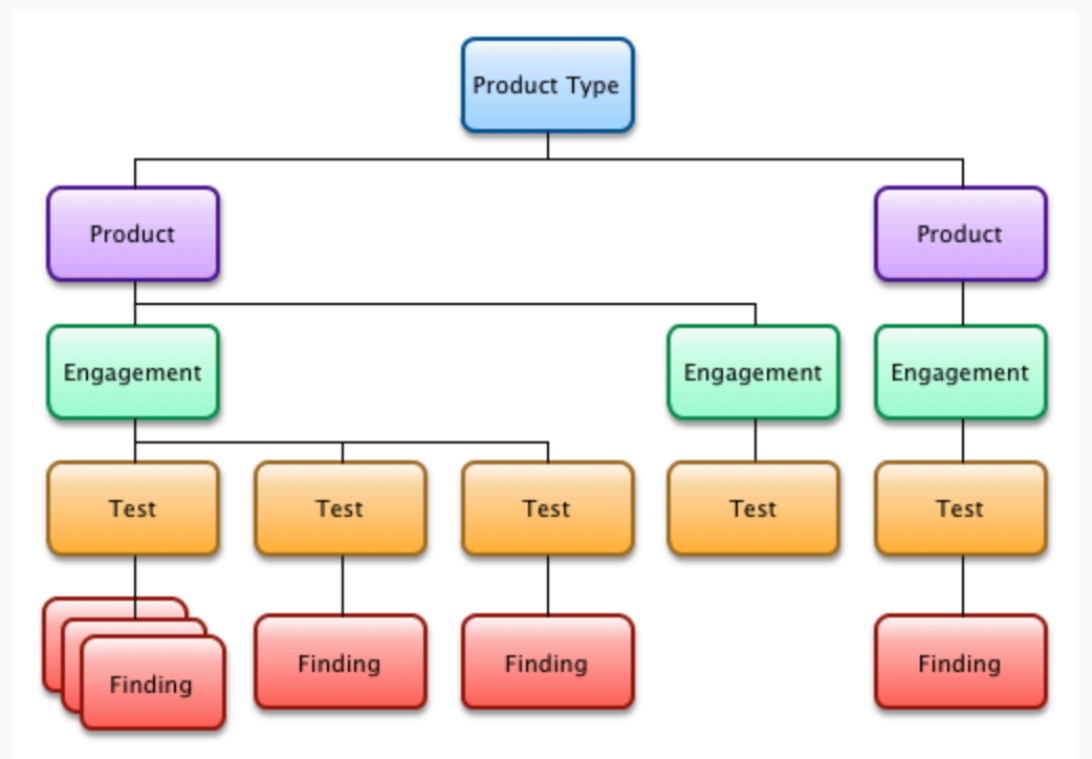
6.5 Tulemuste halduse teostamine

Tulemuste halduse teostamise peatükk kirjeldab DefectDojo turvanõrkuste halduse rakenduste kasutuselevõtmist CodeGuardi rakenduse arhitektuuris. Integreerimise alapeatükis on ülevaade selle teostamisest DefectDojo ja CodeGuardi rakenduse vahel. Tulemuste halduse alapeatükis on ülevaade DefectDojo tulemuste haldamise võimekusest pärast tulemuste edastamist CodeGuardi rakendusest.

6.5.1 Integratsioon

TruffleHogi rakenduse skaneerimise tulemusi on võimalik salvestada faili või kuvada käsurealt otse ekraanile. Tulemuste analüüsimine käsurealt või mitmest failist on mitmete koodivaramute puhul üsna keeruline. Selleks, et oleks võimalik tulemusi kuvada, hallata ja luua protsesse tulemuste lahendamiseks, on töö autor automatiseerinud CodeGuardi rakenduse tulemuste saatmise DefectDojo rakendusse. DefectDojo on rakendus, mis võimaldab hallata rakendustega seotud turvanõrkusi.

DefectDojo andmemudel on üles ehitatud toodetele ehk rakendustele. Andmemudel algab nimetusega *Product Type* ehk toote tüüp, seejärel *Product* ehk toode, igal tootel võib olla mitu tootega seotud tegevust ehk *Engagement*, iga tegevusega on seotud testid ehk *Tests* ning lõpuks testidega on seotud tulemused ehk *Findings*. DefectDojo andmemudel on nähtav joonisel Joonis 8 [14].



Joonis 8 DefectDojo andmemudel [14]

CodeGuardi rakendus peab looma kogu DefectDojo andmemudeli automaatselt, seetõttu on selle andmemudelit ühtlustatud DefectDojo andmemudeliga. DefectDojo andmemudelis olev toode on võrdsustatud CodeGuardi rakenduse koodivaramuga, tegevuse objekt on omakorda kirjeldatud koodiprojektina, testimise objekt on seotud

koodiprojekti skaneerimisega ning tulemused on nimetatud koodiprojekti skaneerimise tulemused. CodeGuardi rakendus lubab koodivaramuid loogiliselt rühmitada, mis lubab ka andmemudelit rohkem kohandada. Näiteks on võimalik määratleda enda toode kui rakendus ning siduda rakendusega seotud koodiprojektid erinevatest koodivaramutest või kui määratleda toode kui üks koodivaramu, on võimalik hallata tulemusi koodivaramupõhiselt. Kogu struktuuri saab kirjeldada CodeGuardi rakenduse *repository.json* failis ning rakendus ise loob automaatselt skaneerimise käigus vastavad objektid DefectDojo rakendusse.

CodeGuardi rakendus integreeritakse DefectDojo rakendusega läbi rakendusliidese ning andmevoog on ühesuunaline, ehk andmeid saadetakse ühesuunaliselt CodeGuard rakendusest DefectDojo rakendusse. Integreerimiseks tuleb CodeGuardi rakenduse *codeguard.conf* sätetefailis kirjeldada DefectDojo rakendusliidese aadress, kasutaja ja võti. Seejärel toimub andmete saatmine automaatselt DefectDojo rakendusse.

Järgnevalt on kirjeldatud DefectDojo rakendusliidese päringud, mida CodeGuardi rakendus kasutab:

- *products/* - GET-meetodiga küsitakse teavet olemasolevate toodete või toote kohta, POST-meetodiga lisatakse puuduv toode.
- *engagements/* - GET-meetodiga küsitakse teavet olemasolevate tegevuste või tegevuse kohta, POST-meetodiga luuakse puuduv tegevus.
- *importscan/* - POST-meetodiga laetakse üles skaneerimise tulemused.

6.5.2 Tulemuste haldamine

Kogu tulemuste haldus alates tulemuste registreerimisest kuni nende sulgemiseni on võimaldatud DefectDojo enda võimekustega. Antud töö puhul lahendab DefectDojo rakendus järgnevad tulemustega seotud tingimused:

- Kasutajaliides tulemuste analüüsimiseks
- Tulemuste teavituste saatmine
- Ligipääsuõiguste võimalus

- Tulemuste haldamise töövoogude võimekus

Tulemuste halduse juures on oluline tulemuste jagamine toodete või koodivaramute omanikega, kuid seejuures piirata ligipääsu ainult selleks ettenähtud isikutega. Tulemuste jagamine on võimaldatud DefectDojo rollipõhise lahendusega, kus iga toote õigusi on võimalik piirata.

Tulemuste saatmine DefectDojo rakendusse ja teavituste registreerimine seal on automatiseeritud CodeGuardi rakendusega. Teavitusi on võimalik saata erinevate kanalite kaudu, näiteks rakenduses endas, e-postiga või edastada rakendustesse nagu Microsoft Teams ja Slack. Teavitusi on võimalik rakendada tootepõhiselt ning kirjeldada igal tootel erinev teavituste saaja ja teavituste kanal.

DefectDojo rakenduses on võimalik kasutada mitmeid erinevaid võimalikke töövooge. Töövooge on võimalik luua vastavalt organisatsiooni vajadusele. Kuna antud töö eesmärk ei olnud töövoogude analüüsimine ja kavandamine, seetõttu on kasutatud lahenduses kasutusele võetud kõige tavapärasem töövoog. Uute tulemuste ilmnemisel lisatakse tulemused avatud olekuga ning seejärel tulemuste läbivaatusega tuleb kinnitada nende olek, kas siis on tegemist kinnitatud õige-või valepositiivse tulemusega. Kinnitatud positiivsete tulemuste korral tuleb tegeleda edasi puuduste kõrvaldamise- ja analüüsiga, näiteks kõrvaldada võtmed lähtekoodist, vahetada võtmed uute vastu ning analüüsida lekkinud võtme(te) mõju. Pärast puudus(t)e kõrvaldamist määratakse positiivne tulemus parandatuks ehk lahendatuks.

7 Tulemused

Tingimustele vastavuse peatükk kirjeldab loodud lahenduste vastavust töös püstitatud lähtetingimustele. Töö lisaväärtuse loomiseks on loodud lahendust testitud lisaks töös kajastatud suurettevõttele ka teises valiktestimise ulatuse jaoks sobilikus organisatsioonis, mille ülevaade on välja toodud valiktestimise tulemuse peatükis. Tuginedes loodud lahenduse tulemuste testimisele ja vastavuse kontrollile, tuuakse edasiarenduse peatükis välja võimalikud lahendused rakenduse parendamiseks, mida teostada tulevikus.

7.1 Tingimustele vastavus

Töö tulemusena valmis automatiseeritud koodivaramutes asetsevate võtmete tuvastamise esialgne lahendus, mis vastab kõikidele töö analüüsi osas kirja pandud lähtetingimustele. Töö analüüsi osas kirja pandud lähtetingimusi võrreldi olemasolevate võtmete tuvastuse lahendustega, mille tulemusena selgus, et ükski olemasolev lahendus ei vasta kõikidele seatud lähtetingimustele. Seetõttu tuli töö autoril valida olemasolevate lahenduste hulgast parim võimalik ning täiendada seda, et viia vastavusse töö lähtetingimustega.

Parimaks võimalikuks lahenduseks osutus võrdluse käigus TruffleHogi võtmete tuvastamise rakendus, kuid valitud lahendus vastas vaid osaliselt kõikidele seatud lähtetingimustele. TruffleHogi lahendus vastas kõikidele tuvastuse võimekuste tingimustele, kuid puudusid järgnevad tingimused:

- Automatiseerimise võimekus
- Mitme koodivaramu tugi
- SSH võtmepõhine autentimine koodivaramutesse
- Kasutajaliides tulemuste haldamiseks
- Uute tuvastuste teavituste loomine

Töö autor arendas uue lahenduse nimega CodeGuard, mis on kasutab edasiarendatud ja parandatud TruffleHogi võtmete tuvastuse loogikat. Lisaks on edasiarendustele ja parandatud võtmete tuvastuse loogikale, on rakendusele juurde arendatud

lähtetingimustele vastamiseks automatiseerimise võimekus, mitme koodivaramu tugi, SSH võtmepõhine autentimine koodivaramusse sisselogimisel ning integreerimise võimekus DefectDojo turvanõrkuste haldusrakendusega. Kuna DefectDojo omab kasutajaliidest tulemuste haldamiseks ja samuti võimaldab luua teavitusi uute tulemuste ilmnemisel, siis integreerimise tulemusena on võimaldatud CodeGuardi rakendusega tuvastatud tulemuste haldamine ja teavituste edastamine läbi DefectDojo rakenduse.

7.2 Valiktestimise tulemus

Valiktestimise ulatusse oli kaasatud üks organisatsioon, et testida lahenduse kasutatavust läbi tõhususe, efektiivsuse ja rahulolu. Valiktestimise küsimustik oli jaotatud kaheks, esimene osa hindas CodeGuard lahenduse kasutatavust, kasutades SUS (*System Usability Scale*) ehk süsteemi kasutatavuse skaalat ning teine osa hindas autori enda loodud küsimustega lahenduse aktuaalsust. Koostatud valiktestimise küsimustik on välja toodud lisade hulgas (vt. Lisa 2). Küsimustiku vastuste statistika ei ole eraldi välja toodud, sest valiktestimisse kaasati üks organisatsioon, millest küsimustele vastas kaks inimest ning seetõttu ei ole võimalik vastuste anonüümsust saavutada. Samuti SUS küsimustiku hindamisega võiks olla valimis rohkem vastajaid, et saada paremat ülevaadet, kuid antud töö raames annab ka kaks vastajat piisava tagasiside.

SUS küsimustikuga tuli välja, et süsteem on asjakohane ning võimalik kasutada eesmärgipäraselt. Vastustest eristus välja ka puudus, et lahenduse kasutajad vajaksid tehnilise inimese tuge. See tulened sellest, et puudus täielik lahenduse dokumentatsioon kasutamise kohta ning paigaldus ning koodivaramute ja -projektide kirjeldamine on osadele inimestele keerulisem.

Süsteemi tagasiside vabas vormis vastused ühtegi puudust välja ei toonud, kuid tuli välja arenduse ettepanek, et liidestada CodeGuard lahendus koodivaramute rakendusliidesega ning saada nimekiri kõikidest koodiprojektidest automaatselt ning ei peaks neid ise eraldi kirjeldama. Selline funktsionaalsus hoiaks kokku ajakulu koodivaramute ja koodiprojektide kirjeldamisel, samuti eeldab funktsionaalsuse loomine iga koodivaramu rakendusliidese uurimist ja arendust kuna koodivaramute rakendusliidese päringud on erinevad või mõnel juhul isegi puuduvad.

Lahenduse paigalduse ja seadistuse käigus tuli ka valiktestimisel välja ettepanek koodivaramute kirjeldamisel kasutada JSON andmevormingu asemel YAML andmevormingut, mis on kasutaja seisukohast mugavam. Ettepanekut mainiti paigalduse käigus, kuid eraldi küsimustikus seda välja ei olnud toodud.

Lisaks süsteemi kasutatavuse hindamisele, otsustas koguda vastajatelt informatsiooni ka lahenduse aktuaalsuse kohta. Lahenduse aktuaalsuse küsimustikust tuli üheselt välja, et võtmete lekkimine koodivaramust on organisatsiooni jaoks risk, mida võivad põhjustada tarkvaraarendajad ja süsteemihaldurid talletades võtmeid koodivaramusse teadmatusest või ka mugavusest ning, mille vähendamiseks peab organisatsioon kasutama meetmeid, et riski vähendada. Samuti oli vastajate jaoks automatiseeritud koodivaramutes talletavate võtmete tuvastuse lahendus uudne.

Valiktestimise tulemusega sai lahenduse tagasisideks palju olulist informatsiooni ning testimisse kaasatud organisatsioon avaldas soovi lahendust kasutada ka pärast valiktestimise perioodi.

7.3 Edasiarendused

Valminud prototüüp lahendus on vastav töö lähtetingimustes kirjeldatud nõuetele ning lahendab ettevõtte vajadused tuvastamaks võtmete ilmnemist koodivaramutes. Lahenduse testimise ja kasutuse käigus ilmnesid mõned kasutajamugavuse asjaolud, mida võiks tulevikus rakenduse edasiarendusega parendama.

Järgnevad kasutusjuhud, mis vajaks tulevikus parendamist:

- Võtme tuvastusel kuvab DefectDojo turvanõrkuste rakendus leiu kohta *git* redaktsiooni räsi, tuvastuse kuupäeva ja kellaaja, koodiprojekti ning selles tuvastatud võtme ja faili, millest võti leiti. Ainult nende väärtuste alusel ei ole võimalik tuvastada, kas tegemist on vale- või õigepositiivse tuvastusega. Tuvastuse kinnitamiseks oleks vaja näha ka ülejäänud faili sisu, millest võti tuvastati. DefectDojo ei kuva ülejäänud faili sisu, ülejäänud sisu nägemiseks tuleb koodivaramus otsida vastavat redaktsiooni ning seejärel on näha vastav fail ja muudatustega sisu.

- DefectDojo lubab filtreerida ja otsida tuvastusi näiteks koodivaramute, -projektide, tuvastuse kuupäeva, kategooria ja oleku alusel, kuid otsimine ja filtreerimine puudub failide ja tuvastatud võtmete põhiselt.

- Koodivaramute ja koodiprojektide määratlemine JSON-vormingus on töö autorile mugav ja lihtne kuid valiktestimise käigus selgus, et teistele ei pruugi selline meetod olla kõige sobilikum. JSON-vormingu keerukuse korral on soovitatud seda vahetada ümber YAMLi vorminguks, mida peetakse rohkem kasutajasõbralikumaks [15]. Samuti oleks mugavam määratleda koodivaramuid ja -projekte veebirakenduses ning ei peaks selleks alati serverisse sisse logima.

- Võimalus liidestada koodivaramute rakendusliidesega, et saada koodivaramute ja koodiprojektide nimekiri CodeGuard rakendusse automaatselt ning ei peaks ise neid eraldi varamute failis määratlema. Funktsionaalsuse loomine eeldaks iga koodivaramu rakendusliidese uurimist ja eraldi liidestuse arendamist, sest iga koodivaramu rakendusliidese päringud on erinevad või isegi puudub rakendusliides.

- Praegusel hetkel on võimalik tuvastada võtmeid kõikidest Git tüüpi koodivaramutest kuid SVN tüüpi koodivaramud ei ole toetatud. SVN tüüpi koodivaramuid kasutatakse järjest vähem kuid tulevikus võiks lahendus toetada võtmete leidmist ka SVN tüüpi koodivaramutest.

Praegune lahendus keskendub lekkinud võtmete tuvastusele kuid tulevikus võiks lahendus toimida ka ennetaval põhimõttel ning takistada võtmete jõudmist koodivaramusse. Sellise lahenduse puhul peaks olema tuvastuse agent, mis töötab koodivaramus ning kasutab *git hook* funktsiooni *pre-receive*. Näiteks kui koodivaramusse üritatakse muudatusi salvestada siis enne salvestamist käivitatakse *pre-receive* funktsioon, mis käivitab võtmete tuvastuse agendi, kogub koodimuudatused ja saadab need võtmete tuvastuse serverisse skaneerimiseks. Skaneerimise tulemused saadetakse tagasi agenti ning seejärel teab agent võtmete ilmnemisel takistada koodimuudatuste jõudmise koodiprojekti. Valepositiivse tulemuse puhul saaks tuvastuse serveris kirjeldada tulemuse valepositiivseks ning järgneva *pre-receive* agendi käivitamisel saab agent serverist vastuse, et tegemist on valepositiivsega ning lubab muudatusi salvestada koodiprojekti.

Lahendus on kättesaadav avatud lähtekoodina ja kõigile kasutatav, kuid valiktestimisest tuli välja, et dokumentatsioon ja paigaldusjuhend võiks olla põhjalikum. Tulevikus peaks kindlasti täiendama dokumentatsiooni ja paigaldusjuhendit, et kasutajatel oleks lihtsam lahendust paigaldada ja kasutada.

Töö autor plaanib lahendust edasi arendada ning lisada GodeGuardi rakendusele enda arendatud kasutajaliidese koodivaramute ja -projektide määratlemiseks. Samuti võimekuse selliste tulemuste otsimiseks, filtreerimiseks ja kuvamiseks, mis praegusel hetkel puuduvad DefectDojo rakenduses. DefectDojo integreerimise võimekus jääks alles, sest kindlasti võib olla kasutajaid, kes kasutavad DefectDojo rakendust ning tahaksid kõiki turvanõrkusi hoida ühes rakenduses. Samuti oleks arhitektuuriliselt keerulisem või isegi võimatu luua ennetavat rakendust DefectDojo liidesega, mis võimaldaks takistada võtmete salvestamist koodivaramusse.

8 Kokkuvõte

Käesoleva diplomitöö eesmärk oli luua suurettevõttele automatiseeritud lahendus, mis võimaldaks tuvastada koodivaramutesse talletatavaid võtmeid.

Töös kirjeldati võtmete lekkimise probleemi olemust, koguti võtmete tuvastuse võimalused ja lähtetingimused vastavalt töös kajastatud suurettevõttele. Kogutud lähtetingimuste alusel võrreldi olemasolevaid lahendusi, et valida välja parim võimalik, kuid ükski olemasolev lahendus ei vastanud kõikidele kirjeldatud lähtetingimustele ning seetõttu tuli autoril arendada uus lahendus, mille võtmete tuvastamise loogika on võetud valitud olemasolevast lahendusest. Töös on välja toodud uue loodud lahenduse tehniline kirjeldus, koos ülesehituse ja arhitektuuriga ning seejärel kirjeldatakse loodud lahenduse teostus vastavalt tehnilisele kirjeldusele ning autori selgituste ja kirjeldustega. Samuti tuuakse töö tulemuste osas välja lahenduse vastavuse kontroll lähtetingimustele, valiktestimise tulemused ja edasiarendused tulevikuks.

Diplomitöö tulemusena valmis lahendus, mis on automatiseeritud ning võimaldab jälgida ettevõtte kõiki koodivaramuid ja tuvastada uusi lisatavaid võtmeid. Lahendus vähendab ettevõttele võtmete lekkimisest tulenevat riski, tuvastades võtmete talletamist koodivaramusse võimalikult varakult kui sellega on võimalik ettevõttele kahju tekitada.

Kasutatud kirjandus

[1] M. Kotadia, „AWS urges developers to scrub GitHub of secret keys," [Võrgumaterjal]. Available: https://www.itnews.com.au/news/aws-urges-developers-to-scrub-github-of-secret-keys-375785.

[2] D. Cameron, „Amazon Engineer Leaked Private Encryption Keys. Outside Analysts Discovered Them in Minutes," [Võrgumaterjal]. Available: https://gizmodo.com/amazon-engineer-leaked-private-encryption-keys-outside-1841160934.

[3] C. E. Shannon, „A Mathematical Theory of Communication," [Võrgumaterjal]. Available: http://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf.

[4] A. D. Diego, „Automatic extraction of API Keys from Android applications," [Võrgumaterjal]. Available: https://drive.google.com/file/d/0B59pyODQqCxiUVkzb3diU0JNVzA/view.

[5] M. Michael, R. M. Matthew ja R. Bradley, „How Bad Can It Git? Characterizing Secret Leakage in Public GitHub Repositories," [Võrgumaterjal]. Available: https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_04B-3_Meli_paper.pdf.

[6] „Customizing Git - Git Hooks," [Võrgumaterjal]. Available: https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks.

[7] J. Thomas, „Git hooks - pre-commit, post-commit, post-receive and more. Automated secrets detection in your software development lifecycle," [Võrgumaterjal]. Available: https://blog.gitguardian.com/git-hooks-automated-secrets-detection/.

[8] „TruffleHog," [Võrgumaterjal]. Available: https://github.com/dxa4481/truffleHog.

[9] „Talisman," [Võrgumaterjal]. Available: https://github.com/thoughtworks/talisman.

[10] „Gitleaks," [Võrgumaterjal]. Available: https://github.com/zricethezav/gitleaks/.

[11] „detect-secrets," [Võrgumaterjal]. Available: https://github.com/Yelp/detect-secrets.

[12] „detect-secrets-server," [Võrgumaterjal]. Available: https://github.com/Yelp/detect-secrets-server.

[13] Nightfall, „Introducing Radar API: Detect Credentials & Secrets in Code via Machine Learning," [Võrgumaterjal]. Available: https://medium.com/@watchtowerai/introducing-radar-api-detect-credentials-secrets-in-code-via-machine-learning-fe402b818bf1.

[14] „DefectDojo's Documentation," [Võrgumaterjal]. Available: https://defectdojo.readthedocs.io/en/latest/index.html.

[15] M. Eriksson ja V. Hallberg, „Comparison between JSON and YAML for data serialization,“ [Võrgumaterjal]. Available: http://www.csc.kth.se/utbildning/kth/kurser/DD143X/dkand11/Group2Mads/Rapport_Malin_Eriksson_Viktor_Hallberg.pdf.

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Tanel Peep

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Automaatne võtmete tuvastamine koodivaramutest suurettevõtte näitel“, mille juhendaja on Edmund Laugasson
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

02.12.2020

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Valiktestimise küsimusik

1. Süsteemi kasutatavuse hindamine *

Küsimustiku eesmärk on hinnata CodeGuard süsteemi kasutatavust. Peamised mõõdetavad aspektid on efektiivsus, tõhusus ja rahulolu.

Küsimustele tuleb vastata 5 punktisel skaalal: 1 - ei nõustu, 2 - pigem ei ole nõus, 3 - ei oska öelda, 4 - pigem olen nõus, 5 - nõustun täielikult

	1 (ei nõustu)	2 (pigem ei ole nõus)	3 (ei tea)	4 (pigem olen nõus)	5 (nõustun täielikult)
Ma arvan, et tahaksin seda süsteemi kasutada sageli	◯	◯	◯	◯	◯
Ma arvan, et süsteem on asjatult keeruline	◯	◯	◯	◯	◯
Ma arvan, et süsteemi on lihtne kasutada	◯	◯	◯	◯	◯
Ma arvan, et süsteemi kasutamiseks vajan tehnilise inimese tuge	◯	◯	◯	◯	◯
Ma leian, et süsteemi erinevad funktsioonid on hästi integreeritud	◯	◯	◯	◯	◯
Ma arvan, et selles süsteemis on liiga palju vasturääkivusi	◯	◯	◯	◯	◯
Ma leian, et enamik inimesi õpiks seda süsteemi kiirelt kasutama	◯	◯	◯	◯	◯
Ma leian, et süsteemi kasutamine on väga kohmakas	◯	◯	◯	◯	◯
Ma tundsin end süsteemi kasutamisel väga enesekindlalt	◯	◯	◯	◯	◯
Mul oli vaja enne süsteemi kasutamist õppida väga palju uusi asju	◯	◯	◯	◯	◯

2. Tagasiside süsteemi kasutamise kohta *

Tagasiside süsteemi toimimisest, ettepanekud edasiarenduseks, jne.

3. Aktuaalsuse hindamine *

Küsimustiku eesmärk on hinnata koodivaramutes olevate võtmete lekkimise aktuaalsust

Küsimustele tuleb vastada 5 punktisel skaalal: 1 - ei nõustu, 2 - pigem ei ole nõus, 3 - ei oska öelda, 4 - pigem olen nõus, 5 - nõustun täielikult

	1 (ei nõustu)	2 (pigem ei ole nõus)	3 (ei tea)	4 (pigem olen nõus)	5 (nõustun täielikult)
Võtmete lekkimine koodivaramust on organisatsiooni jaoks risk	◯	◯	◯	◯	◯
Automatiseeritud võtmete tuvastamine koodivaramust aitab piirata võtmete lekkimisest tulenevat ohtu	◯	◯	◯	◯	◯
Tarkvaraarendajad ja süsteemihaldurid võivad teadmatusest talletada võtmeid koodivaramutesse	◯	◯	◯	◯	◯
Tarkvaraarendajad ja süsteemihaldurid võivad mugavusest talletada võtmeid koodivaramutesse	◯	◯	◯	◯	◯
Automatiseeritud võtmete tuvastamine koodivaramust oli minu jaoks uudne	◯	◯	◯	◯	◯
Võtmete lekkimisel koodivaramust võib organisatsioon saada nii maine kui ka varalist kahju	◯	◯	◯	◯	◯
Organisatsioon peab kasutama meetmeid vähendamaks riski võtmete lekkimiseks	◯	◯	◯	◯	◯