TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Harald Astok

Java turvaaukude leidmine staatilise analüüsiga: millest probleem?

Bakalaureusetöö (9 EAP)

Juhendaja: Vesal Vojdani, PhD

Tartu 2018

Java turvaaukude leidmine staatilise analüüsiga: millest probleem?

Lühikokkuvõte:

Käesoleva bakalaureusetöö käigus uuritakse raamistikust Apache Struts 2 aastal 2017 avastatud turvaauku CVE-2017-5638 ning kas viga oleks olnud võimalik tuvastada varem, kasutades staatilisi koodi analüsaatoreid.

Uurimise käigus kasutatakse avatud lähtekoodiga Apache Struts 2 koodi, milles on uuritav turvaauk, ning seda analüüsitakse kahe tasuta staatilise koodianalüsaatoritega - FindBugs ja PMD. Lisaks uuritakse, milline oli turvaaugu mõju tavakasutajatele, vaadates lähemalt teadaolevalt suurimat ohvrit, milleks on krediidiajaloofirma Equifax.

Võtmesõnad:
Staatiline koodi analüüs, FindBugs, PMD, Equifax, CVE-2017-5638

CERCS: P175 (Informaatika, süsteemi teooria)

Finding Java Security Vulnerabilities Using Static Analysis: whence the Problem?

Abstract:

In process of this Bachelor's Thesis, vulnerability of Apache Struts 2 framework called CVE-2017-5638 is closely examined and static code analysis is conducted in order to determine if the vulnerability could have been detected earlier.

During the process an Apache Struts 2 source code with a vulnerability is examined and analysed with two free static code analysers – FindBugs and PMD. In addition, it is researched what influence the vulnerability had to average user, while examining the biggest known victim, which is consumer credit reporting agency Equifax.

Keywords:
Static code analysis, FindBugs, PMD, Equifax, CVE-2017-5638

CERCS: P175 (Informatics, systems theory)

Sisukord

Sissejuhatus

Vähemalt paar korda aastas raputab maailma uudis mõnest järjekordsest avastatud turvaveast, mis seab ohtu miljonite inimeste isikuandmed, turvalisuse või privaatsuse. Aastal 2017 avaldati 13250 turvaviga, mille CVE (Common Vulnerabilities and Exposures)[1] skoor oli vähemalt 4.0 ehk tegemist on keskmise või kõrgema astme veaga [2]. Enamasti parandatakse leitud turvaaugud kiiresti ning tihti isegi enne, kui viga on avalikkusele teada. Kui vigadest enamasti õpitakse ning nende avastamise süsteeme täiustatakse, siis kerkib küsimus, miks praegusel ajal üldse saavad tekkida turvaaugud, millel on katastroofilised tagajärjed? Erinevad staatilised koodi analüsaatorid lubavad, et annavad arendajatele võimaluse kirjutatud koodi testida, et enne avalikkusele avaldamist oleks see võimalikult ohutu ning usaldusväärne.

Käesoleva bakalaureusetöös uuritakse Apache Struts 2 turvauku CVE-2017-5638 ning kas seda oleks saanud kuidagi ennetada. Lisaks uuritakse turvaaugu tagajärgi, keskendudes kõige suuremale ohvrile – Equifaxile. Tegemist on ühega kolmest peamisest firmast USAs, mis tegeleb inimeste krediidiajaloo hoidmisega ning selle müümisega [3].

Töö esimeses peatükis tutvustatakse rünnakut krediidifirma Equifax vastu ning mis tagajärjed olid turvaaugul CVE-2017-5638 nende süsteemidele. Teises peatükis tutvustatakse Apache Struts 2-te ning kirjeldatakse detailselt uuritavat turvaauku CVE-2017-5638. Kolmandas peatükis antakse ülevaade staatiliste koodi analüsaatorite olemusest. Lisaks võetakse vigane Struts 2 kood ning kasutatakse kaht staatilist analüsaatorit, et tuvastada, kas uuritavat turvaauku oleks saanud nende tööriistadega kuidagi ennetada. Neljandas ehk viimases peatükis analüüsitakse analüsaatorite tulemusi ja arutletakse, kuidas oleks võimalik tulemusi parandada.

[1] Avastatud turvaaukudele viitamise süsteem, kus igale veale omistatakse ID [1]

1 Rünnak Equifaxile

Teadaolevalt suurima küberrünnaku osaliseks langes Equifax, mis on üks suurimatest USA krediidiinfo firmadest. Aastal 2017, vahemikus mai kuni juuli, tabas Equifaxi kaks suurt küberrünnakut. Federal Trade Commissioni andmete põhjal vähemalt 143 miljoni inimese isikuandmed (isikukood, krediidiinfo, krediitkaardi andmed ja palju muud) langesid küberkurjategijate valdusesse [4]. Kui alguses arvati, et on toimunud vaid üks rünne, siis hiljem selgus, et tegelikult tuvastati kaks erinevat rünnakut – esimene toimus märtsis ning teine vahemikus mai – juuli [5].

USA-s kehtiva süsteemi kohaselt hoitakse eraisikute krediidiandmeid, sealhulgas isikukood, krediitkaardi andmed, juhiloaandmed ning palju muud erafirmade serverites. Kliendid saavad firmadest tellida oma krediidiajaloo või krediidiskoori, mis on tihti vajalik laenu taotlemiseks või tööle kandideerimiseks. Ameerika Ühendriikides on see info peamiselt kolme suurfirma käes – Experian, TransUnion ning Equifax [3].

1.1 Esimene rünnak

Esimese rünnaku kohta tulid vihjed 2017-nda aasta märtsi alguses, kui mõned kliendid kaebasid lekke üle, väites, et nende isikuandmed on varastatud. Equifax viis läbi juurdluse, millesse kaasas küberturvafirma Mandiant. Juurdlus lõpetati mais 2017, kuna usutavasti ei leitud ühtegi asitõendit, et Equifaxi vastu oleks toimunud küberrünnak. Firma pole uurimuse raportit avalikkusele avaldanud [5].

1.2 Teine rünnak

Kuigi turvaauk avastati 2017-nda aasta märtsi alguses ning teine rünne Equifaxile toimus vahemikus mai kuni juuli, tegi firma ametliku avalduse rünnete kohta alles 7. septembril 2017. Avalduses öeldi, et firma avastas lekke 29. juulil 2017, kuid andmed olid varastatud juba mitu kuud varem. Kaks päeva hiljem kaasati juurdlusesse taas küberturvafirma Mandiant, mis selgitas välja, et esmane rünne toimus mai keskel [5].

Seni ei ole teada, kes oli rünnakute taga ning kas tegemist oli mõlemal juhul samade inimestega või kahe erineva grupiga. Tumeveebi on ilmunud erinevaid lehekülgi, mis ähvardavad avalikustada kogu varastatud info, kui neile ei maksta suuri summasid. Seni on kõik taolised leheküljed kiiresti maha võetud, mille põhjal võib järeldada, et tegemist ei olnud tegelike ründajatega, vaid tavaliste petturitega, kes lootsid antud olukorrast raha teenida [6].

1.3 Tulemus

Oktoobriks 2017 oli teada umbkaudne arv ameeriklasi, keda rünnak mõjutas - 145,5 miljonit. Lisaks puudutas andmeleke veel 12,2 miljonit Suurbritannia klienti. Hiljem selgus, et lekkinud oli veel 10,9 miljoni ameeriklase juhiloa andmed [7].

Equifaxi käitumist on kritiseeritud, et nad nii hilja rünnakust teada andsid. Firma väitis, et vajas aega aru saamaks rünnaku ulatusest ning selleks, et hinnata kahjusid. Lisaks lõi firma lehekülje equifaxsecurity2017.com. Lehe eesmärk oli aidata Equifaxi kliendil leida kogu vajamineva info rünnaku kohta ning kuidas edasi käituda. Näitamaks, kui jabur idee on firma pealehest equifax.com luua eraldi domeeniga lehekülg, tegi Equifaxi oma töötaja lehe securityequifax2017.com. Leheküljel küsiti kasutajalt tema andmeid, mida oleks saanud kurjategija pahaaimamatu kliendi vastu ära kasutada. Peale andmete sisestamist tuli kasutaja ekraanile teade, et tegemist on vale leheküljega. Lisaks paluti, et inimesed annaksid mikroblogi võrgustiku Twitteri teel Equifaxile teada, et nad oma lehekülge muudaksid [8].

Lisaks on seatud suure kahtluse alla firma juhtide tegevus, kes müüsid plaaniväliselt endale kuuluvaid aktsiaid väärtuses 1,8 miljonit USD. Müük toimus 1. ja 2. augustil 2017 ehk vahetult peale rünnaku avastamist. Peale Equifaxi septembris tehtud avaldust, firma aktsia väärtus langes koheselt. Equifax väidab, et firma juhtidel puudes teave infolekkest aktsiate müügi hetkel [5].[2]

[2] Antud teemal koostatud Vikipeedia artikli autoriks on käesoleva bakalaureusetöö autor

2 Rünnakut võimaldava turvanõrkuse kirjeldus

Järgnevas keskendutakse konkreetsele koodis olevale nõrkusele, mis võimaldas rünnakut läbi viia. Apache Struts 2 on Apache 2.0 litsentsiga, vabavaraline Java-põhine veebiraamistik, mida tihti kasutatakse suuremahuliste veebirakenduste loomiseks. Struts 2 on laialdaselt levinud nii erasektoris kui ka valitsusetasemel, meditsiini- või finantsteenuseid osutatavate asutuste poolt [9].

2.1 Turvaauk CVE-2017-5638 ning rünne

Turvaaugu CVE-2017-5638 olemus seisneb selles, et viga lubab ründajal Apache Struts 2-s käivitada oma koodi. Turvaaugust andis esimesena teada Hiina arendaja Nike Zheng 7. märtsil 2017 [10]. Ründel kasutatakse ära failide üleslaadimisel tekkivat erindit, mis asub Jakarta Multipart parseris, mis eeldab, et erind ei sisalda kasutajapoolset sisendit. Selle kaudu on ründajal võimalus pääseda ligi Content-Type päisesse (ingl *header*), kus ta saab käivitada oma koodi. Üheks rünnet võimaldavaks tehnoloogiaks on veel OGNL (Object Graph Navigation Language)[3] [12].

Haavatavad on ka Apache Struts vanemad versioonid, mis ulatuvad kuni aastasse 2008 [13]. Rünnakute eest kaitsmiseks tuleks uuendada süsteem vähemalt versioonile 2.3.32 või 25.10.1 [14]. Vanemad Struts 2 versioonid on siiani haavatavad. Joonisel 1 (järgmisel leheküljel) on näha ründel kasutatavat koodi.

Turvaugu avastamisel registreeris küberturvafirma Imperva järgneva kuue päeva jooksul tuhandeid rünnakuid oma klientide vastu, mis tulid üle maailma kokku 40-st erinevast riigist [16]. Peamiseks sisestatud käskudeks olid „#cmd=whoami“ või „#cmd=echo klss“, millega saab tuvastada, kas süsteem on haavatav või mitte [17]. Apache Struts 2 raamistik on väga levinud ning seda kasutavad paljud suurfirmad, nende seas ka näiteks sõjatööstuse hiid Lockheed Martin, lennufirma Virgin Atlantic ning telekommunikatsiooni hiid Vodafone [18]. See näitab, et lisaks isikuandmetele, on turvaugul CVE-2017-5638 potentsiaalne oht ka inimeste turvalisusele.

[3] Keel avaldiste väärtustamiseks, mille abil saab Java programmeerimiskeeles objektide omadusi määrata [11]

```
def exploit(url, cmd):
    payload = "%{(#_='multipart/form-data')."
    payload += "(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS)."
    payload += "(#_memberAccess?"
    payload += "(#_memberAccess=#dm):"
    payload += "((#container=#context['com.opensymphony.xwork2.ActionContext.container'])."
    payload += "(#ognlUtil=#container.getInstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class))."
    payload += "(#ognlUtil.getExcludedPackageNames().clear())."
    payload += "(#ognlUtil.getExcludedClasses().clear())."
    payload += "(#context.setMemberAccess(#dm))))."
    payload += "(#cmd='%s')." % cmd
    payload += "(#iswin=(@java.lang.System@getProperty('os.name').toLowerCase().contains('win')))."
    payload += "(#cmds=(#iswin?{'cmd.exe','/c',#cmd}:{'/bin/bash','-c',#cmd}))."
    payload += "(#p=new java.lang.ProcessBuilder(#cmds))."
    payload += "(#p.redirectErrorStream(true)).(#process=#p.start())."
    payload += "(#ros=(@org.apache.struts2.ServletActionContext@getResponse().getOutputStream()))."
    payload += "(@org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros))."
    payload += "(#ros.flush())}"

    try:
        headers = {'User-Agent': 'Mozilla/5.0', 'Content-Type': payload}
        request = urllib2.Request(url, headers=headers)
        page = urllib2.urlopen(request).read()
    except httplib.IncompleteRead, e:
        page = e.partial

    print(page)
    return page
```

Joonis 1. Ründel kasutatav kood [15]

Tegemist ei ole esimese korraga, kui OGNL keele tõttu on Strutsi kasutavad süsteemid haavatavad. Küberturvafirma Imperva avaldas 2017-nda aasta jaanuari alguses artikli, analüüsides nelja erinevat RCE (Remote/Arbitrary Code Execution) [4] tüüpi rünnakuid, milles kõik kasutasid OGNL keelt, et teostada rünnet [19]. Lisaks tuuakse artiklis välja, et selleks ajaks teadaolevast 68-st haavatavusest Apache Strutsile, 12 on seotud OGNL keele ära kasutamisega. 2017. aasta septembris avastati järjekordne turvaauk, mis kasutab ära REST (Representational State Transfer) [5] pistikprogrammi (ingl *plugin*) viga, millega ründaja saab teostada RCE tüüpi rünnet. See näitab, et Struts 2 raamistik võib endas peita veel seni avastamata vigu.

[4] Rünnakutüüp, kus ründaja käivitab võõras süsteemis oma koodi

[5] Tarkvaraarhidektuuri stiil, mis lubab arvutisüsteemidel omavahel lihtsamini suhelda [20]

2.2 Vigane kood

Järgnev analüüs põhineb Eric Rafaloffi uurimisel [21] ning piltidel olev kood pärineb Struts 2 avalikust GitHubi repositooriumist [22]. Peamine probleem seisneb selles, et Struts 2 kasutas vaikimisi Jakarta Multipart süntaksianalüsaatorit (ingl *parser*) juhul, kui kasuliku koormuse (ingl *payload*) päise tüübiks on määratud „multipart/form-data“, mida on ka näha Joonisel 1, real 9. Päises olev OGNL avaldis põhjustab Jakarta Multipart süntaksianalüsaatoris erindi, mis seejärel suunatakse edasi veateadet moodustavasse meetodisse koos OGNL sisendiga. Seal kasutatakse OGNL avaldist, mille tulemusel saab ründaja sisestada enda koodi. Vaatame nüüd vigast koodi natukene lähemalt. Kõik saab alguse klassist JakartaMultiPartRequest, milles kasutatakse meetodit parse, mida on näha Joonisel 2 real 64.

Real 67 kutsutakse välja meetod processUpload, mis on ka Joonisel 2, real 90. Meetod processUpload kutsub real 91 välja meetodi parseRequest, mida näeb Joonisel 2, real 144.

Meetodis kontrollitakse, kas tuleks visata erind või mitte. Antud juhul erind visatakse. Sealt minnakse tagasi real 64 olevasse parse meetodisse, kus tekkinud erindi tõttu antud juhul käivitub real 68 olev erindi püüdmise plokk. Kuna faili suurusega probleeme pole, siis minnakse real 75 meetodisse buildErrorMessage, mis on Joonisel 3, real 98. Nii kaua kui parseRequest meetod viskab erindit, ei oma väga suurt tähtsust, mis tüüpi on see erind, sest igal juhul jõuab parse meetodi kood välja buildErrorMessage meetodisse. Meetod asub klassis AbstractMultiPartRequest.

```
64 public void parse(HttpServletRequest request, String saveDir) throws IOException {
65     try {
66         setLocale(request);
67         processUpload(request, saveDir);
68     } catch (FileUploadException e) {
69         LOG.warn("Request exceeded size limit!", e);
70         LocalizedMessage errorMessage;
71         if(e instanceof FileUploadBase.SizeLimitExceededException) {
72             FileUploadBase.SizeLimitExceededException ex =
                    (FileUploadBase.SizeLimitExceededException) e;
73             errorMessage = buildErrorMessage(e, new Object[]{ex.getPermittedSize(),
                    ex.getActualSize()});
74         } else {
75             errorMessage = buildErrorMessage(e, new Object[]{});
76         }
77
78         if (!errors.contains(errorMessage)) {
79             errors.add(errorMessage);
80         }
81     } catch (Exception e) {
82         LOG.warn("Unable to parse request", e);
83         LocalizedMessage errorMessage = buildErrorMessage(e, new Object[]{});
84         if (!errors.contains(errorMessage)) {
85             errors.add(errorMessage);
86         }
87     }
88 }


90 protected void processUpload(HttpServletRequest request, String saveDir) throws
       FileUploadException, UnsupportedEncodingException {
91     for (FileItem item : parseRequest(request, saveDir)) {
92         LOG.debug("Found file item: [{}]", item.getFieldName());
93         if (item.isFormField()) {
94             processNormalFormField(item, request.getCharacterEncoding());
95         } else {
96             processFileField(item);
97         }
98     }
99 }


144 protected List<FileItem> parseRequest(HttpServletRequest servletRequest, String
       saveDir) throws FileUploadException {
145     DiskFileItemFactory fac = createDiskFileItemFactory(saveDir);
146     ServletFileUpload upload = createServletFileUpload(fac);
147     return upload.parseRequest(createRequestContext(servletRequest));
148 }
```

Joonis 2. parse, processUpload ning parseRequest meetodid klassis JakartaMultiPartRequest

```
98      protected LocalizedMessage buildErrorMessage(Throwable e, Object[] args) {
99          String errorKey = "struts.messages.upload.error." + e.getClass().getSimpleName();
100         LOG.debug("Preparing error message for key: [{}]", errorKey);
101
102         return new LocalizedMessage(this.getClass(), errorKey, e.getMessage(), args);
103     }
```

Joonis 3. buildErrorMessage meetod klassis JakartaMultiPartRequest

Meetod tagastab LocalizedMessage'i, mis on loomult konteiner, ning sisaldab loodud sõne errorKey, mille väärtuseks on "struts.messages.upload.error.InvalidContentTypeException". Lisaks on isendi defaultMessage väärtuseks kasutaja poolt päises määratud „multipart/form-data“. Järgnevalt kutsutakse välja MultiPartRequestWrapperi klassi konstruktoris meetod parse, mis on Joonisel 4, real 86.

Real 88 kutsutakse välja meetod addError, mis asub samas klassis ning on näha real 247.

```
77  public MultiPartRequestWrapper(MultiPartRequest multiPartRequest, HttpServletRequest request,
78                                  String saveDir, LocaleProvider provider,
79                                  boolean disableRequestAttributeValueStackLookup) {
80          super(request, disableRequestAttributeValueStackLookup);
81          errors = new ArrayList<>();
82          multi = multiPartRequest;
83          defaultLocale = provider.getLocale();
84          setLocale(request);
85          try {
86              multi.parse(request, saveDir);
87              for (LocalizedMessage error : multi.getErrors()) {
88                  addError(error);
89              }
90          } catch (IOException e) {
91              LOG.warn(e.getMessage(), e);
92              addError(buildErrorMessage(e, new Object[] {e.getMessage()}));
93          }
94      }


247 protected void addError(LocalizedMessage anErrorMessage) {
248         if (!errors.contains(anErrorMessage)) {
249             errors.add(anErrorMessage);
250         }
251     }
```

Joonis 4. MultiPartRequestWrapperi klassi konstruktor ja addError meetod

Meetodi eesmärk on kontrollida, kas antud erindit on varem nähtud ning kui ei ole, lisab selle muutujasse, milles on kõik LocalizedMessage objektid. Järgnevalt kutsutakse klassis

Dispatcher välja meetod wrapRequest, mida saab näha Joonisel 5. Meetod kontrollib, kas päringu Content-Type on „multipart/form-data“, mida see ka on.

```
794 public HttpServletRequest wrapRequest(HttpServletRequest request) throws IOException {
795         // don't wrap more than once
796         if (request instanceof StrutsRequestWrapper) {
797             return request;
798         }
799 
800         String content_type = request.getContentType();
801         if (content_type != null && content_type.contains("multipart/form-data")) {
802             MultiPartRequest mpr = getMultiPartRequest();
803             LocaleProvider provider = getContainer().getInstance(LocaleProvider.class);
804             request = new MultiPartRequestWrapper(mpr, request, getSaveDir(), provider,
805                     disableRequestAttributeValueStackLookup);
806         } else {
807             request = new StrutsRequestWrapper(request,
808                     disableRequestAttributeValueStackLookup);
809         }
810 
811         return request;
812     }
```

Joonis 5. wrapRequest meetod klassis Dispatcher

Kontroll toimub real 801 ning kuna on teada, et Content-Type sisaldab „multipart/form-data“, luuakse real 804 uus MultiPartRequestWrapper, mis lisab selle erindi LocalizedMessage'isse. Selleks hetkeks on HTTP (Hypertext Transfer Protocol)[6] päring läbi töötatud ning kasutusele võetakse filtrid (ingl *interceptor*), mida Struts kasutab päringute töötlemiseks. Üks filter, mida Struts iga päringu puhul kasutab, on klass FileUploadInterceptor. Klassis olev meetod intercept, mis on Joonisel 6, kontrollib real 242, kas päring on MultiPartRequestWrapperi objekt, mida see on, kuna see lisati (Joonis 5, rida 804).

[6]Protokoll, mis aitab veebilehitsejatel suhelda veebis serveritega

```
237 public String intercept(ActionInvocation invocation) throws Exception {
238         ActionContext ac = invocation.getInvocationContext();
239
240         HttpServletRequest request = (HttpServletRequest)
                ac.get(ServletActionContext.HTTP_REQUEST);
241
242         if (!(request instanceof MultiPartRequestWrapper)) {
243             if (LOG.isDebugEnabled()) {
244                 ActionProxy proxy = invocation.getProxy();
245                 LOG.debug(getTextMessage("struts.messages.bypass.request", new
                        String[]{proxy.getNamespace(), proxy.getActionName()}));
246             }
247
248             return invocation.invoke();
249         }


261 if (multiWrapper.hasErrors()) {
262             for (LocalizedMessage error : multiWrapper.getErrors()) {
263                 if (validation != null) {
264                     validation.addActionError(LocalizedTextUtil.findText(error.getClazz(),
265                             error.getTextKey(), ActionContext.getContext().getLocale(),
266                             error.getDefaultMessage(), error.getArgs()));
267                 }
268             }
269         }
```

Joonis 6. intercept meetod klassis FileUploadInterceptor

Samas meetodis, Joonisel 6, real 261, kontrollitakse, kas MultiPartRequestWrapper sisaldab veateateid, mida ta teadaolevalt sisaldab. Edasi kutsutakse ridadel 264 – 266 välja klassi LocalizedTextUtil meetod findText, millega antakse kaasa mitmeid argumente, nende seas muudetud textKey ning defaultMessage.

Järgnevalt liigutakse edasi klassi LocalizedTextUtil, kus kutsutakse välja meetod findText, mille kirjeldust võib näha Joonisel 7. Meetodi eesmärk on leida sobiv veateade, mis vastab etteantud parameetritele. Meetod kutsutakse välja, kui aClassName on AbstractMultiPartRequest, aTextName väärtus on textKey, mis sai kasutaja poolt seatud kui "struts.messages.upload.error.InvalidContentTypeException". Lisaks on muutuja locale seatud kui ActionContext's locale ning args tühi massiiv. Meetodis findText tekkiv muutuja valueStack on ActionContexti valueStack.

```
Finds a localized text message for the given key, aTextName. Both the key and the message itself
     is evaluated as required. The following algorithm is used to find the requested message:

  1. Look for message in aClass' class hierarchy.
    1. Look for the message in a resource bundle for aClass
    2. If not found, look for the message in a resource bundle for any implemented interface
    3. If not found, traverse up the Class' hierarchy and repeat from the first sub-step
  2. If not found and aClass is a {@link ModelDriven} Action, then look for message in the model's
       class hierarchy (repeat sub-steps listed above).
  3. If not found, look for message in child property. This is determined by evaluating the
       message key as an OGNL expression. For example, if the key is user.address.state, then it
       will attempt to see if "user" can be resolved into an object. If so, repeat the entire
       process fromthe beginning with the object's class as aClass and "address.state" as the
       message key.
  4. If not found, look for the message in aClass' package hierarchy.
  5. If still not found, look for the message in the default resource bundles.
  6. Return defaultMessage
When looking for the message, if the key indexes a collection (e.g. user.phone[0]) and a message
     for that specific key cannot be found, the general form will also be looked up (i.e.
     user.phone[*]).

If a message is found, it will also be interpolated. Anything within ${...} will be treated as an
     OGNL expression and evaluated as such.
```

Joonis 7. findText meetod ning selle kirjeldus kirjeldus klassis LocalizedTextUtil

Kuna erindi sõnumile "struts.messages.upload.error.InvalidContentTypeException"ei leita vastet, kutsutakse real 573 välja meetod getDefaultMessage, mis on näha Joonisel 8.

```
570 // get default
571         GetDefaultMessageReturnArg result;
572         if (indexedTextName == null) {
573             result = getDefaultMessage(aTextName, locale, valueStack, args, defaultMessage);
574         } else {
575             result = getDefaultMessage(aTextName, locale, valueStack, args, null);
576             if (result != null && result.message != null) {
577                 return result.message;
578             }
579             result = getDefaultMessage(indexedTextName, locale, valueStack, args, defaultMessage);
580         }
```

Joonis 8. findText meetodi osa klassis LocalizedTextUtil

Meetod getDefaultMessage asub samas klassis ning teeb viimase katse leida sobiv veateade, mis ebaõnnestub. Meetod on näha Joonisel 9. Real 729 antakse klassi TextParseUtil meetodisse translateVariables edasi muudetud argument message.

```
714 private static GetDefaultMessageReturnArg getDefaultMessage(String key, Locale locale, ValueStack
        valueStack, Object[] args,
715                                                                 String defaultMessage) {
716       GetDefaultMessageReturnArg result = null;
717       boolean found = true;
718
719       if (key != null) {
720           String message = findDefaultText(key, locale);
721
722           if (message == null) {
723               message = defaultMessage;
724               found = false; // not found in bundles
725           }
726
727           // defaultMessage may be null
728           if (message != null) {
729               MessageFormat mf = buildMessageFormat(TextParseUtil.translateVariables(message,
                        valueStack), locale);
730
731               String msg = formatWithNullDetection(mf, args);
732               result = new GetDefaultMessageReturnArg(msg, found);
733           }
734       }
735
736       return result;
737   }
```

Joonis 9. getDefaultMessage meetod klassis LocalizedTextUtil

Joonisel 10 olev meetodi kirjeldus ning meetod translateVariablesi eesmärk on väärtustada vajaminev OGNL keel. Kuigi sama nimega meetodeid on klassis palju, on neil kõigil erinevad argumendid, millest viimast, kuhu ka lõpuks jõutakse, võib näha Joonisel 10, real 154.

```
36 /**
37      * Converts all instances of ${...}, and %{...} in <code>expression</code> to the value
            returned
38      * by a call to {@link ValueStack#findValue(java.lang.String)}. If an item cannot
39      * be found on the stack (null is returned), then the entire variable ${...} is not
40      * displayed, just as if the item was on the stack but returned an empty string.
41      *
42      * @param expression an expression that hasn't yet been translated
43      * @param stack value stack
44      * @return the parsed expression
45      */


154 public static Object translateVariables(char[] openChars, String expression, final ValueStack
        stack, final Class asType, final ParsedValueEvaluator evaluator, int maxLoopCount) {
155
156         ParsedValueEvaluator ognlEval = new ParsedValueEvaluator() {
157             public Object evaluate(String parsedValue) {
158                 Object o = stack.findValue(parsedValue, asType);
159                 if (evaluator != null && o != null) {
160                     o = evaluator.evaluate(o.toString());
161                 }
162                 return o;
163             }
164         };
165
166         TextParser parser =
                ((Container)stack.getContext().get(ActionContext.CONTAINER)).getInstance(TextParser.class);
167
168         return parser.evaluate(openChars, expression, ognlEval, maxLoopCount);
169     }
```

Joonis 10. meetodi translateVariables kirjeldus klassis LocalisedTextUtil ning viimane translateVariables meetod klassis LocalizedTextUtil

Real 168 kasutatakse meetodit evaluate OGNL välja arvutamiseks. Probleem seisnebki selles, et meetodid, mis aluskoodi analüüsivad, ei tohiks analüüsida kasutajapoolset sisendit, mida rünnakus ära kasutatakse. Kuigi turvaaugu peamiseks süüdlaseks peetakse Jakarta päringu pakendit (ingl *wrapper*), siis tegelikult on probleem hoopis filtris, mis eeldab, et erindi sõnum ei sisalda kasutajapoolset sisendit.

2.2.1 Lahendus

Turvaaugu parandamiseks muudeti failides JakartaMultiPartRequest.java, JakartaStream-MultiPartRequest.java ning MultiPartRequestWrapper.java read, mis on näha Joonisel 11, kus roheline tähistab lisatud ning punane kustutatud rida. Kui errorKey'le puudus

väärtus, liiguti vigase versiooni puhul edasi kasutajapoolse sisendiga. Nüüd antakse väärtuse mitteäratundmise või puudumise korral sellele vaikeväärtus, et vigase koodiga mitte edasi liikuda.

```
- return LocalizedTextUtil.findText(this.getClass(), errorKey, defaultLocale, e.getMessage(),
    args);
+ if (LocalizedTextUtil.findText(this.getClass(), errorKey, defaultLocale, null, new Object[0]) ==
    null) {
+ return LocalizedTextUtil.findText(this.getClass(), "struts.messages.error.uploading",
    defaultLocale, null, new Object[] { e.getMessage() });
+ } else {
+   return LocalizedTextUtil.findText(this.getClass(), errorKey, defaultLocale, null, args);
+ }
```

Joonis 11. Muudetud read failides JakartaMultiPartRequest.java, JakartaStreamMultiPartRequest.java ning MultiPartRequestWrapper.java [23]

Failis FileUploadInterceptor.java muudetud ridu näeb Joonisel 12.

```
- return LocalizedTextUtil.findText(this.getClass(), errorKey, defaultLocale, e.getMessage(),
    args);
+ if (LocalizedTextUtil.findText(this.getClass(), errorKey, defaultLocale, null, new Object[0]) ==
    null) {
+ return LocalizedTextUtil.findText(this.getClass(), "struts.messages.error.uploading",
    defaultLocale, null, new Object[] { e.getMessage() });
+ } else {
+ return LocalizedTextUtil.findText(this.getClass(), errorKey, defaultLocale, null, args);
+ }
```

Joonis 12. Muudetud read failis FileUploadInterceptor.java [24]

Lisaks ei kasutata Java.io.File isendeid ning LocalizedTextUtil.java klassi, kus asub getDefaultMessage meetod, mida ründel ära kasutati.

3 Vigase koodi uurimine staatiliste analüsaatoritega

Käesolevas bakalaureusetöös otsustas autor kasutada kahte tuntud ning tasuta staatilist koodianalüsaatorit: FindBugs ning PMD. Valiti need kaks, kuna mõlema puhul on tegemist tasuta programmidega, mida on suhteliselt lihtne kasutada ning need otsivad ka turvaauke. Arenduskeskkonnana kasutatakse IntelliJ IDEA-d.

3.1 Tööpõhimõte

Staatiline koodi analüüs või lihtsalt staatiline analüüs on koodi testimise ja hindamise viis ilma koodi käivitamata [25] ning on võimeline avastama paljusid paljusid probleeme: erinevaid turvaaugu liike (näiteks võõra käsu sisestamine, tundmatute HTTP päiste kontroll [26]), kasutamata koodi, kas sõnede võrdluses on arvestatud null väärtusega ja palju muud. Suurbritannial on kaitseministeeriumi poolt väljastatud nõuded, mis puudutavad kaitsetööstuses kasutusel olevat tarkvara: kogu tarkvara peab olema kontrollitud kasutades staatilist koodi analüüsi [27]. Staatilise koodi analüüsi eeliseks on, et seda saab vajadusel kasutada lokaalselt oma arvutis ilma interneti ühenduseta. Populaarsemad, näiteks Marylandi ülikooli poolt loodud avatud lähtekoodiga FindBugs, on jooksutatavad enimlevinumates arenduskeskkondades ning analüüsi programmid suudavad suhteliselt kiiresti üle vaadata ning tuvastada levinumad või triviaalsemad turvaaugud. Analüsaatorite peamisteks negatiivseteks külgedeks võib pidada keerulisemate turvaaukude mitteleidmist, tulemustes on tihti palju müra ning on oht väärpositiivsele või –negatiivsele tulemusele. Peamine meetod staatiliseks analüüsiks on teadaolevate vigade mustrite võrdlus lähtekoodiga, mille põhjal programm saab otsustada, kas kasutajat peaks hoiatama või mitte. Teine viis, mida kasutatakse, on kompileeritud baitkoodi kasutamine mustrite võrdluseks [28]. Kuna staatilise analüsaatori tulemustes esineb valepositiivseid ning –negatiivseid tulemusi, tuleks iga veateade inimese poolt üle kontrollida.

Staatilist analüüsi on mõistlik kasutada, kuna vigade parandamine on tunduvalt lihtsam ja ka odavam, kui need tulevad välja arenduseprotsessi algfaasis. Seda on tõestanud ka Struts 2 turvaauk, mida selles uurimistöös käsitletakse. Kuna turvaauk on eksisteerinud pikalt, siis on haavatava süsteemi kasutajaid palju. Struts 2 puhul on tegemist raamistikuga, mitte mingi väiksema programmiga, mida saaks lihtsalt uuendada, mistõttu võivad

paljudel vigase Struts 2 kasutajatel puududa teadmised või muu ressurss, et oma süsteeme uuendada.

Üks põhjus, mis võis mängida olulist rolli turvaaugu CVE-2017-5638 tekkel, on analüsaatorite võime RCE rünnakuid mitte tuvastada. Väga üksikud analüsaatorid väidavad, et on võimelised leidma RCE tüüpi turvaauke.

3.2 FindBugsi analüüs turvaauku sisaldava Struts 2 koodile

FindBugs on Marylandy ülikooli poolt välja arendatud vabavaraline programm, mis otsib staatilise analüüsiga Java koodist vigu. Selleks kasutab programm erinevaid vigade mustreid, mida võrdleb kasutaja baitkoodiga. Oma kodulehel lubavad FindBugsi arendajad, et valede hoiatuste arv jääb alla 50% [29]. Find Security Bugs on omakorda FindBugsi pistikprogramm, mis keskendub ainult turvavigade leidmisele ning on avatud lähtekoodiga [30]. Lisaks on FindBugs pistikprogrammina olemas peamistes arenduskeskkondades, näiteks IntelliJ IDEA, Android Studio ning Eclipse.

Peale FindBugsi käivitamist on tulemusi võimalik uurida kahel viisil. Esimene võimalus on uurida analüüsi tulemusi kasutatavas arenduskeskkonnas, kus FindBugs annab kasutajale mitu võimalust vigadega tutvuda: koodi klasside või pakkide (ingl *package*), vigade kategooriate või liigi põhjal. Kui valida liigi põhjal, on kasutajal kolm võimalust: murettekitav, tuleks vaadata ning hirmus. Valides mõne konkreetse teate, avaneb vea kirjeldus ning kood, mida see puudutab. Teine võimalus kuidas analüüsi tulemusi vaadata on eksportida xml või html failid, mis sisaldavad kogu infot.

Praeguses bakalaureusetöös käsitletakse FindBugsi analüüsi tulemusi, mis on saadud eksporditud core mooduli html failist, kuna kõik vigased failid asuvad selles. Esiteks on kohe näha, et programm lahterdab vigu kahte kategooriasse – keskmise ja kõrge prioriteediga hoiatused. Kokku toob programm välja 596 hoiatust, millest hetkel oluliseks peetakse 78 pahatahtliku koodi haavatavuse hoiatust ning 73 turvalisuse hoiatust. Kõikide hoiatuste liigid on välja toodud Joonisel 13.

Warning Type	Number
Bad practice Warnings	85
Correctness Warnings	83
Experimental Warnings	1
Internationalization Warnings	13
Malicious code vulnerability Warnings	78
Multithreaded correctness Warnings	10
Performance Warnings	145
Security Warnings	73
Dodgy code Warnings	107
Total	**595**

Joonis 13. FindBugsi loodud raportis kõikide leitud veatüüpide tabel

Esiteks vaadatagu, mis tüüpi vigu klassifitseeritakse keskmise ning mis tüüpi kõrge prioriteediga hoiatuste alla. Keskmise prioriteediga vigade alla loetakse näiteks pea kõik halva tava, jõudluse ning turvalisuse hoiatused. Kõrge prioriteediga vigadest üle veerandi moodustavad küsitava väärtusega koodi (ingl *dodgy code*). Eksporditud html failist on hea otsida huvipakkuvaid teateid, kuna seal on välja toodud faili nimi ning vastav rida, mida viga puudutab. Kuna on teada turvaauku põhjustavad failid ning read, saab veateateid lihtsalt otsida.

Seega otsitakse relevantseid faili nimesi ning ridu. Ainsad raportis välja toodud teated, mille kirjeldus sisaldab ka turvaauguga CVE-2017-5638 seotud ridu, ei ole tegelikult kuidagi seotud huviobjektiks oleva turvaauguga. Välja on toodud faili JakartaMultiPartRequest.java meetodi processUpload ning meetodi buildErrorMessage väljakutsumine. Kuigi need meetodid on olulised, et CVE-2017-5638 saaks toimida, on antud teate põhjuseks Potential Path Traversal. Teade on kategoriseeritud kui keskmise prioriteediga hoiatus. See pole praegu oluline, kuna uurimise all oleva rünnaku tüüp on Remote Code Execution.

3.3 PMD analüüs turvaauku sisaldava Struts 2 koodile

PMD on avatud lähtekoodiga staatiline koodi analüsaator, mis on loodud peamiselt Java koodi kontrollimiseks. Uurides, mille taga seisavad PMD tähed, saab teada, et tegelikkuses need ei oma need mingit tähendust, arendajate arvates kõlavad need tähed koos hästi [31]. Programmi on sisse ehitatud reeglid, mille põhjal see vigu otsib. Lisaks on kasutajal võimalik lisada ja kirjutada oma reegleid, mille põhjal programm saab vigu otsida. Programm on olemas eraldi allalaadimiseks, kui ka pistikprogrammina peamistes arenduskeskkondades, nagu näiteks Eclipse, Android Studio ning IntelliJ IDEA.

Nagu ka FindBugsil, on ka PMD-l võimalik tulemusi vaadata kahel viisil. Kõigepealt saab neid vaadata arenduskeskkonnas, kus programm on jaotanud tulemused erinevatesse kategooriatesse, mis tüüpi vigu on otsitud. Näiteks on erinevad kategooriad koodistiili, veateadete, lõimtöötluse (ingl *multithreading*) ning jõudluse osas. Avades mõne kausta, avanevad konkreetsed vead, mille peale vajutades avaneb seotud kood. Teine võimalus on eksportida kõik teated html failina, kus on välja toodud relevantne fail, selles olev rida ning vea tüüp.

Kuna html failist on relevantseid veateateid oluliselt lihtsam leida, kasutab autor ekspordi valikut. Esiteks on näha, et kuna fail avaneb väga kaua, siis leitud viga hulk on väga suur. Kokku tuvastas PMD 105174 viga. Kuna selles failis ei saa koheselt vaadata, mis raskusastme liigitustega on vead, hakatakse otsima failidega seotud vigu. Sorteerides välja turvaauguga CVE-2017-5638 seotud failid, saadakse kokku 9 teadet. Neist 4 puudutavad Demeteri seadust, mis ütleb, et programmi osad peaksid suhtlema ainult neile lähedaste osadega [32]. Kuna tegemist on pigem koodi stiili tüüpi veaga, siis need teated praegu ei huvita. Ülejäänud välja toodud vead klassifitseeruvad ka kui koodistiili või halva tava tüüpi vead.

Vaadates PMD dokumentatsiooni, on näha, et viga kategoriseeritakse ka numbri järgi, skaalal 1-10, kus 1 on kõige kõrgema astmega viga. Otsides kõige kõrgema astmega viga, ehk numbriga 1 ning keskmine-kõrge astmega viga, numbriga 2, ei ole analüüsis ühtegi taolise numbriga veateadet.

3.4 Tulemuste analüüs

Antud uurimuse põhjal võib tõdeda, et kumbki analüsaator ei leidnud ning isegi ei andnud ühtegi vihjet uuritava turvaaugu olemasolu kohta.

Üldiselt oli mõlema programmi puhul väga keeruline otsida või saada mingit ülevaadet analüsaatori tulemustest. Vaadatagu natukene lähemalt FindBugsi analüüsi tulemusi kogu Struts 2 projektile. Segadust tekitab asjaolu, et näiteks IntelliJ IDEA-s kõige hirmsamate (ingl *scariest*) tulemuste kategoorias toodi välja kaks leidu ning mõlemal juhul oli tegemist equals meetodi kasutamisega, mis asusid test klassis. Lisaks olid mõlemad vead keskmise prioriteediga. Taoline prioriteedi ja kategooria määramine võib kasutajas tekitada pigem segadust kui luua selget ülevaadet tulemustest. FindBugsi puhul tuleks kasuks vea tähtsuse täpsem liigitamine, näiteks skaalal 1 - 10.

Lisaks oli keskmise prioriteediga vigade seas päris palju teateid, mille tegelik risk oli palju väiksem. Näiteks peaaegu kõik halva koodi tüübiga vead olid keskmise prioriteediga. Raporteid oleks oluliselt lihtsam analüüsida, kui sellist tüüpi vigade jaoks oleks olemas näiteks madala prioriteediga vigade kategooria. Sellisel juhul oleks raportites oluliselt vähema müra, mis raskendab lugemist ning adekvaatse ülevaate saamist.

PMD tulemusi on veel keerulisem uurida. Kui lasta programmil otsida kõiki vigu, mida see suudab, on leitud vigade arv väga suur, antud juhul 105174. Kuigi PMD kodulehel pole välja toodud, kui suur protsent leitud tulemustest võivad olla valepositiivsed, siis isegi kui võtta eelduseks, et näiteks 30%, peaks tõeseid vigu olema umbes 73621, mis on ikkagi väga suur arv. Veateatele väljastatakse lisaks ainult vea pealkiri, mille peale hiirega vajutades avaneb selgitav interneti lehekülg.

Mõlema prgrammi tulemustega saab lähemalt tutvuda töö autori GitHubi kontol, aadressil `https://github.com/HaraldAstok/Bachelors`

4 Kokkuvõte

Käesolevas bakalaureusetöö eesmärgiks oli uurida, kas Apache Struts 2 turvaauku CVE-2017-5638 oleks olnud võimalik ennetada, kasutates staatilisi analüsaatoreid. Töös uuriti sügavuti turvaauku ning vigasele koodile rakendati kahte erinevat staatilist analüsaatorit: FindBugs ning PMD. Lisaks uuriti, milline mõju oli turvaaugul tavainimesele.

Leiti, et kuigi mõlemat programmi oli lihtne kasutada, olid väljastatavate tulemuste uurimine keeruline. Mõlema programmi puhul polnud tulemuste väljastamise viis väga loogiline, kuna esines vastuolusi vea kategooria ja proriteedi määramisel. Sellest sõltuvalt oli tulemustes palju ebavajalike teateid, mis suurendasid uuritavate andmete hulka. Lisaks raskendas analüüsi vigade hulk - FindBugsil 595 ning PMD-l 105174. Taolise hulga läbi vaatamine ja iga vea analüüsimine ei ole mitte mingil määral mõistlik ega efektiivne. Kui oleks olemas vigade kategooriate filtreerimise variant, muudaks see analüüsi tunduvalt kiiremaks.

Uurides mõlema programmi leide, saab järeldada, et kasutades FindBugsi või PMD-d, ei oleks olnud võimalik ennetada turvaauku CVE-2017-5638. Mõlemad programmid leidsid küll suurel hulgal vigu, kuid ükski neist ei viidanud konkreetsele turvaaugule. Seega on saadud vastus peamisele probleemile ning võib järeldada, et tööriistad on liiga ebatäpsed, et oleks saanud ennetada turvaauku.

Antud bakalaureusetöö edasiarendusena oleks võimalik uurida tasulisi staatilisi analüsaatoreid, näiteks lgtm või Klockwork. Selliste analüsaatorite kasutamine võimaldaks näha, mis vahe on tasulistel ja tasuta saada olevatel staatiliste analüsaatorite tulemustel.

Viidatud kirjandus

[1] Rouse M. Common Vulnerabilities and Exposures (CVE). TechTarget. https://searchfinancialsecurity.techtarget.com/definition/Common-Vulnerabilities-and-Exposures. Kasutatud: 10. mai 2018.

[2] CVE Details. https://www.cvedetails.com/vulnerability-list.php?vendor_id=0&product_id=0&version_id=0&page=265&hasexp=0&opdos=0&opec=0&opov=0&opcsrf=0&opgpriv=0&opsqli=0&opxss=0&opdirt=0&opmemc=0&ophttprs=0&opbyp=0&opfileinc=0&opginf=0&cvssscoremin=0&cvssscoremax=0&year=2017&month=0&cweid=0&order=3&trc=14712&sha=815cc1a3d2c4b72bf23b8a2fa85939f5ab0041c0. Kasutatud: 26. märts 2018.

[3] Irby L. Who Are the Major Credit Reporting Agencies? The Balance. https://www.thebalance.com/who-are-the-three-major-credit-bureaus-960416. Kasutatud: 16. märts 2018.

[4] Gressin S. The Equifax Data Breach: What to Do. Federal Trade Commission. https://www.consumer.ftc.gov/blog/2017/09/equifax-data-breach-what-do. Kasutatud: 17. märts 2018.

[5] Riley M., Sharpe A., and Robertson J. Equifax Suffered a Hack Almost Five Months Earlier Than the Date It Disclosed. Bloomberg Technology. https://www.bloomberg.com/news/articles/2017-09-18/equifax-is-said-to-suffer-a-hack-earlier-than-the-date-disclosed. Kasutatud: 17. märts 2018.

[6] Paganini P. Hackers are offering Equifax data for sale, but they are scammers. Security Affairs. http://securityaffairs.co/wordpress/63047/cyber-crime/equifax-scammers.html. Kasutatud: 17. märts 2018.

[7] Andriotis A. and Glazer E. Equifax Hack Disclosed Driver's License Data for More Than 10 Million Americans. The Wall Street Journal. https://www.wsj.com/articles/equifax-hack-disclosed-drivers-license-data-for-more-than-10-million-americans-1507664315. Kasutatud: 17. märts 2018.

[8] Astor M. Someone Made a Fake Equifax Site. Then Equifax Linked to It. The New York Times. https://www.nytimes.com/2017/09/20/business/equifax-fake-website.html. Kasutatud: 17. märts 2018.

[9] Apache Struts. https://struts.apache.org/. Kasutatud: 16. märts 2018 a.

[10] Chirgwin R. Apache Struts 2 needs patching, without delay. It's under attack now. The Register. https://www.theregister.co.uk/2017/03/09/apache_under_attack_patch_for_zero_day_available/. Kasutatud: 16. märts 2018.

[11] Apache Commons. https://commons.apache.org/proper/commons-ognl/. Kasutatud: 16. märts 2018.

[12] Bhatt I. An analysis of the Apache Struts 2 Vulnerability CVE-2017-5638. Flexera Blogs. https://blogs.flexera.com/sca/2017/09/an-analysis-of-the-apache-struts-2-vulnerability/. Kasutatud: 23. märts 2018.

[13] Goodin D. Critical vulnerability under "massive" attack imperils high-impact sites [Updated]. Ars Technology. https://arstechnica.com/information-technology/2017/03/critical-vulnerability-under-massive-attack-imperils-high-impact-sites/. Kasutatud: 13. märts 2018.

[14] Lenart L. and Gielen R. S2-045. Confluence. https://cwiki.apache.org/confluence/display/WW/S2-045. Kasutatud: 16. märts 2018.

[15] Vex Woo. Apache struts 2.3.5 < 2.3.31 / 2.5 < 2.5.10 - remote code execution. Exploit Database. https://www.exploit-db.com/exploits/41570/. Kasutatud: 16. märts 2018.

[16] Joskowicz M. and Avital N. CVE-2017-5638: New Remote Code Execution (RCE) Vulnerability in Apache Struts 2. Imperva. https://www.imperva.com/blog/2017/03/cve-2017-5638-new-remote-code-execution-rce-vulnerability-in-apache-struts-2/. Kasutatud: 17. märts 2018.

[17] Biasini N. Content-Type: Malicious - New Apache Struts2 0-day Under Attack. Talos. https://blog.talosintelligence.com/2017/03/apache-0-day-exploited.html. Kasutatud: 13. märts 2018.

[18] Pittenger M. "Easy"to Hack Apache Struts Vulnerability CVE-2017-9805. Black Duck. https://blog.blackducksoftware.com/hack-apache-struts-cve-2017-9805. Kasutatud: 13. märts 2018.

[19] Ajay Uggirala and Gilad Yehudai. Remote Code Execution (RCE) Attacks on Apache Struts. Imperva. https://www.imperva.com/blog/2017/01/remote-code-execution-rce-attacks-apache-struts/. Kasutatud: 19. aprill 2018.

[20] What is REST?. Codecademy. https://www.codecademy.com/articles/what-is-rest. Kasutatud: 10. mai 2018.

[21] Rafaloffi E. An Analysis Of CVE-2017-5638. Gotham Digital Science. https://blog.gdssecurity.com/labs/2017/3/27/an-analysis-of-cve-2017-5638.html. Kasutatud: 30. märts 2018.

[22] Mirror of apache struts. github. https://github.com/apache/struts/tree/55fed537646ba33d6789deb15f0d8ef99b870594. Kasutatud: 14. jaanuar 2018.

[23] Lenart L. Uses default error key if specified key doesn't exist. GitHub. https://github.com/apache/struts/commit/352306493971e7d5a756d61780d57a76eb1f519a?diff=unified. Kasutatud: 30. märts 2018.

[24] Lenart L. Uses default error key if specified key doesn't exist. GitHub. https://github.com/apache/struts/commit/6b8272ce47160036ed120a48345d9aa884477228?diff=unified. Kasutatud: 30. märts 2018.

[25] Louridas P. Static code analysis. IEEE Software. https://ieeexplore.ieee.org/document/1657940/#full-text-section. Kasutatud: 14. märts 2018.

[26] Bug Patterns. Find Security Bugs. http://find-sec-bugs.github.io/bugs.htm. Kasutatud: 18. märts 2018.

[27] Requirements for safety related software in defence equipment. http://www.software-supportability.org/Docs/00-55_Part_2.pdf. Kasutatud: 20. märts 2018.

[28] Static Analysis Tools for security. Careerdrill. http://careerdrill.com/blog/uncategorized/static-analysis-tools-security/. Kasutatud: 26. märts 2018.

[29] Findbugs fact sheet. http://findbugs.sourceforge.net/factSheet.html. Kasutatud: 30. märts 2018.

[30] Find security bugs. https://find-sec-bugs.github.io/. Kasutatud: 30. märts 2018.

[31] Pmd source code analyzer project. https://pmd.github.io/pmd-6.2.0/pmd_projectdocs_trivia_meaning.html. Kasutatud: 30. märts 2018.

[32] Lieberherr K. Law of demeter: Principle of least knowledge. http://www.ccs.neu.edu/home/lieber/LoD.html. Kasutatud: 30. märts 2018.

Lisad

I. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Harald Astok**,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

   Java turvaaukude leidmine staatilise analüüsiga: millest probleem?

   mille juhenda on Vesal Vojdani

   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, 14.05.2018