VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Automatizuotas programų saugumo spragų aptikimas

Automatic Detection of Software Vulnerabilities

Magistro baigiamasis darbas

Atliko: Piotr Petunov (parašas)

Darbo vadovas: lekt. Vaidas Jusevičius (parašas)

Recenzentas: doc. Kristina Lapin (parašas)

Vilnius – 2016

SANTRAUKA

Kiekvienoje sistemoje yra spragų ir kiekviena sistema naudoja daug trečiųjų šalių programinės įrangos, todėl svarbu kiekvienai konkrečiai sistemos konfigūracijai sugebėti nustatyti jos saugumą. Visą reikalingą informaciją apie rastas saugumo spragas galima paimti iš laisvai pasiekiamos NVD duomenų bazės. Linux OS įdiegtų paketų sąrašą su jų versijomis galima nesudėtingai gauti iš paketų valdyklės. Duomenų apie sistemos saugumą formalizavimui geriausiai tinka ontologijos. Darbe buvo sukurta ontologija, kuri yra tokio ekspresyvumo, kad ją praplėtus veikimo greitis su samprotautoju būtų pakankamas. Prototipo testavimo rezultatai parodė, kad visos populiariausios atnaujintos Linux distribucijos turi saugumo spragų.

Raktiniai žodžiai: saugumas, saugumo spraga, ontologija, ekspresyvumas.

SUMMARY

It’s important for each system configuration to be able to determine it’s security, because each system has vulnerabilities and each system uses a lot of third party software. All necessary information can be loaded from an openly available NVD database. Package manager can easily return full list of installed packages with their versions in a Linux OS. Ontologies are well suited to formalize data about system security. Reasoners work fast enough with ontologies of such expressivity. Testing of prototype showed that all popular Linux distributions have security vulnerabilities.

Keywords: security, vulnerability, ontology, expressivity.

TURINYS

ĮVADAS

Neįmanoma sukurti programinės įrangos, kurioje užtikrintai nebūtų spragų. Sistemoje vienu metu gali veikti daug įvairios programinės įrangos, vadinasi kiekvienoje sistemoje potencialiai yra spragų. Šiomis spragomis pasinaudoję įsilaužėliai gali pasiekti arba modifikuoti sistemoje esančią konfidencialią informaciją. Tokios spragos yra reguliariai aptinkamos ir sukelia didelius nuostolius verslui.

Atlikto empirinio tyrimo duomenimis, vidutiniškai atskleidus spragą programinės įrangos tiekėjas prarasdavo 0,86 milijardo dolerių kapitalo[TW07]. NIST duomenimis JAV ekonomika per metus praranda 60 milijardų dolerių, dėl išlaidų susijusių su kūrimu ir platinimu programinės įrangos pataisų ir pažeistų sistemų perdiegimo, taip pat dėl prarasto produktyvumo, kuris atsirado dėl žalingų kompiuterinių programų (*angl. malware*) ir kitų problemų[ZC09].

Svarbu tiksliai apibrėžti ką laikome spraga. Deja, vieno plačiai priimto ir tikslaus saugumo spragos apibrėžimo nėra:

- Pažeidžiamumas (*angl. vulnerability*) sistemoje, kuris atsiranda dėl sistemos architektūros, įgyvendinimo, priežiūros arba palaikymo[JWX+08].
- Pažeidžiamumas sistemoje, kuriuo pasinaudojant galima pakeisti numatytą sistemos veiklą. Spragos gali būti saugumo, vientisumo, pasiekiamumo ir kitų tipų[NRC91].
- Vidinė klaida, dėl kurios išorinė klaida gali pakenkti sistemai[ALR+04].
- Klaida arba defektas technologijoje arba jos panaudojime, kuris sukelia pažeidžiamumą sistemoje, dėl kurio gali įvykti veikla, kuri įtakos sistemos saugumą arba gyvybingumą.

Aptiktos spragos įprastai pirmiausia yra užfiksuojamas programinės įrangos tiekėjo puslapyje. Vadinasi, jei norime patikrinti visą naudojamą programinę įrangą, informacijos apie aptiktas spragas turėtume ieškoti kiekvieno atskiro tiekėjo puslapyje, o tai nėra patogu. Kai kurios saugumo spragos gali atsirasti vienoje programinėje įrangoje, o jų poveikis gali būti kitoje, pavyzdžiui kokioje nors bibliotekoje. Tokios situacijos dar labiau apsunkina informacijos apie spragas radimą. Kita problema, kad kiekvienas programinės įrangos tiekėjas spragoms aprašyti gali naudoti skirtingą formatą. Vadinasi, net jei ir galėtume efektyviai surinkti informaciją apie spragas iš visų šaltinių, tokią informaciją būtų sudėtinga apdoroti. Efektyviau yra pasinaudoti spragų duomenų bazėmis.

Saugumo spragų duomenų bazių yra daugiau nei viena. Kiekvienoje duomenų bazėje yra aprašomi skirtingi duomenys, todėl reikia išnagrinėti sąryšius tarp atskirų bazių. Be to, skirtingos duomenų bazės gali kaupti duomenis apie skirtingą programinę įrangą. Todėl svarbu teisingai

pasirinkti duomenų bazę, nes nuo to priklausys korektiškas saugumo patikrinimo priemonės veikimas.

Praktikoje dažniausiai siekiama patikrinti įvairių, o ne kažkokios konkrečios sistemos, saugumą, todėl negalime iš anksto tiksliai nusakyti sistemos charakteristikų (operacinės sistemos, įdiegtos programinės įrangos versijų) ir pagal tai pritaikyti priemonę. Vadinasi, reikės nepriklausomai nuo naudojamos operacinės sistemos rasti sąrašą naudojamos programinės įrangos ir jos versijų.

Darbo tikslai ir uždaviniai

Darbo tikslas yra sukurti metodą ir jį realizuojantį prototipą, kuris gebėtų automatizuotai surinkti informaciją apie egzistuojančias saugumo spragas, rastų sistemoje įdiegtą programinę įrangą bei pagal šią informaciją nurodytų ar sistemoje yra neištaisytų saugumo spragų.

Darbo uždaviniai:

1. Pasirinkti tinkamiausius informacijos apie saugumo spragas šaltinius.
2. Išanalizuoti informaciją esančią informacijos apie saugumo spragas šaltiniuose.
3. Sukurti būdą formalizuotai nuspręsti ar sistema yra saugi.
4. Sukurti metodą, kuriuo galima būtų patikrinti sistemos saugumą pagal nustatytus reikalavimus.
5. Sukurti prototipą realizuojantį darbe pateiktą metodą.
6. Patikrinti prototipo veikimą sąlygose artimose realioms.

1. SPRAGŲ DUOMENŲ BAZĖS

Norint užtikrintai pasakyti, kad sistema yra saugi, reikia turėti aktualią ir pilną informaciją apie tuo metu aptiktas saugumo spragas visoje sistemoje įdiegtoje programinėje įrangoje. Kadangi kiekvienoje sistemoje tipiškai yra programinės įrangos iš įvairių trečiųjų šalių, būtų nepatogu ieškoti informacijos apie spragas kiekvieno atskiro programinės įrangos tiekėjo šaltiniuose. Be to, programinės įrangos tiekėjas gali būti šališkas ir gali neatskleisti visos informacijos apie realų programinės įrangos saugumą. Objektyvią, nešališką ir pilną informaciją galime gauti iš saugumo spragų duomenų bazių. Tokių bazių yra daugiau nei viena, todėl reikia pasirinkti tinkamiausią arba naudoti kelias duomenų bazes. Pastaruoju atveju būtina nustatyti ar duomenys jose yra suderinami, t.y. ar negali atsirasti prieštaringų ar persiklojančių duomenų (pvz.: jei abiejose duomenų bazėse yra užfiksuota ta pati spraga, bet nurodytos skirtingos paveikto produkto versijos).

1.1. National Vulnerability Database

National Vulnerability Database[NVD15] (NVD) yra National Institute of Standards and Technology programinės įrangos saugumo spragų duomenų bazė. Svarbiausias informacijos apie saugumo spragas šaltinis šioje duomenų bazėje yra CVE duomenų rinkinys. CVE yra įvardinama ne kaip spragų duomenų bazė, o kaip spragų identifikacijos sistema, kurios tikslas yra suteikti tokius bendrus vardus viešai žinomoms saugumo spragoms, kad spragų duomenų bazes ir kitus duomenis galima būtų apjungti tarpusavyje. Kiekvienas CVE įrašas turi unikalų identifikatorių, būseną (spragos įrašas arba kandidatas tapti įrašu), bendrą aprašymą ir vieną ar daugiau nuorodų į išorinius informacijos šaltinius apie spragą. Informacija apie spragas yra priimama kaip spragos forma (*angl. vulnerability submission*). MITRE[MIT15] jai priskiria CVE identifikatorių ir kandidato būseną. Spraga tampa įrašu, kai ją peržiūri CVE redaktorių taryba.

Pateikiami tokie svarbiausi duomenys apie spragą:

- spragos tipas pagal Common Weakness Enumeration[CWE15] (CWE) klasifikacijos sistemą;
- paveiktos programos vardas, versijų numeriai ir platintojas nurodyti vienu Common Platform Enumeration identifikatoriumi;
- spragos pavojingumas pagal Common Vulnerability Scoring System[CVS+15] standartą;
- išoriniai informacijos šaltiniai apie spragą (pvz.: nuoroda į programinės įrangos tiekėjo tinklapyje esantį spragos aprašymą);

- nuorodos į kitas saugumo spragų duomenų bazes (pvz.: OSVDB);
- spragos aprašymas, kuriame nurodyta kokiu principu veikia spraga.

Nuo 2002 metų NIST publikuoja NVD spragų duomenų bazes XML failų pavidalu, pavadinimu nvdcve-2.0-metai.xml, kur metai reiškia skaičių nuo 2002 iki 2015. 2008 metais XML schema buvo atnaujinta iš 1.2.1 į 2.0 versiją, bet iki šiol yra palaikoma ir atnaujinama ir senesnio formato duomenų bazė. Šie failai yra laisvai prieinami atsisiųsti oficialiame CVE tinklapyje. Apie naujausias rastas spragas galima sužinoti taip pat ir iš RSS srautų, kurie pateikia duomenis apie spragas, kurios buvo rastos, arba apie spragas, kurios buvo išanalizuotos, per paskutines aštuonias dienas.

Ši duomenų bazė yra palaikoma JAV vyriausybės ir turi daugiau nei 70000 spragų įrašų, todėl gerai tinka automatiniam spragų tikrinimui, tačiau dėl santykio su valstybinėmis organizacijomis gali kilti abejonių dėl saugumo spragų pateikimo proceso skaidrumo. Sunku įrodyti, kad saugumo organizacijos negali įtakoti duomenų bazės duomenų ir yra visiškai nešališkos. Patikimiau būtų naudoti duomenų bazę, kuri nėra valdoma vienos organizacijos.

1.2. Open Sourced Vulnerability Database

Open Sourced Vulnerability Database[OSV15] (OSVDB) yra nepriklausomos atviro kodo spragų duomenų bazės projektas, kurio tikslas yra suteikti tikslią, detalią, atnaujintą ir objektyvią informaciją apie saugumo spragas. Projektas ne tik suteikia išsamią duomenų bazę, bet ir suteikia išplėstas paieškos, klasifikacijos ir nuorodų funkcijas.

Atviro kodo statusas kalba apie pačio projekto veikimą, o ne apie programinę įrangą, kurios spragos yra fiksuojamos. Projekte yra fiksuojamos tiek ir atviro, tiek ir uždaro kodo programinės įrangos spragos.

Projektas buvo įkurtas 2002 metų rugpjūčio mėnesį Black Hat ir Defcon konferencijų metu, bet pradėjo įsibėgėti tik po 2003 metų Defcon konferencijos ir atviram naudojimui buvo publikuotas 2004 metų kovo 31 dieną. Ilgalaikiam projekto stabilumui užtikrinti buvo įkurta ne pelno siekianti organizacija Open Security Foundation (OSF).

2012 metais projekto pavadinime „Open Source“ buvo pakeista į „Open Sourced“ norint parodyti, kad projektas nutolo nuo tipiško atviro kodo projekto modelio. Tai reiškia, kad atviro kodo principai galioja projekto sukurtai programinei įrangai, o ne projekto surinktiems duomenims.

2013 metų pabaigoje OSVDB buvo užfiksuota 100000 spragų. Dauguma spragų įvedė Risk Based Security[RBS15] įmonės darbuotojai. Projekto metu buvo rasta 23000 spragų, kurios

nėra užfiksuotos CVE duomenų bazėje. Šiuo metu duomenų bazėje yra daugiau nei 120000 spragų. Šios duomenų bazės neatstovauja jokia svarbi organizacija, todėl duomenų pateikimas yra skaidresnis nei NVD atveju.

OSVDB duomenų bazėje suteikiama tokia svarbiausia informacija apie spragą:

- paveikta programinė įranga yra identifikuojama laisva forma spragos pavadinime ir aprašyme;
- pateikiamos nuorodos į spragos aprašymą kituose resursuose (gali būti ir kitos duomenų bazės, spragos aprašymas tiekėjo puslapyje arba tiesiog nuoroda į tiekėjo puslapį);
- tekstinis spragos veikimo principo aprašymas;
- gali būti pateiktas spragos sprendimas (pvz.: nurodyti žingsniai kaip išvengti arba parašyta iki kokios minimalios versijos reikia atsinaujinti).

Skirtingai nei NVD atveju, OSVDB nesuteikia laisvos prieigos prie visų duomenų. Galima tik paieška duomenų bazėje, o prieiga prie visų duomenų yra leidžiama tik gavus atskirą sutikimą iš Risk Based Security. Toks sutikimas nesuteikia teisės pasidalinti duomenimis, todėl jei kuriamas įrankis naudotų šią duomenų bazę, kiekvienas naudotojas turėtų atskirai gauti leidimą. Dėl šios priežasties buvo atsisakyta toliau darbe naudoti šią duomenų bazę.

1.3. CAPEC

Dar vienas būdas nagrinėti saugumo spragas yra pasinaudojant atakų šablonais. Atakų šablonai (*angl. attack patterns*) yra būdas fiksuoti ir aprašyti populiarius įsilaužimo metodus iš įsilaužėlio perspektyvos. Atakos šablonas yra paplitusių elementų ir technikų aprašymai, kurie yra naudojami įsilaužiant. Jie aprašo iššūkius su kuriais susiduria įsilaužėlis ir būdus, kuriais pasinaudodamas jis išsprendžia tuos iššūkius. Jie yra išvesti iš projektavimo šablonų, tik naudojami ne konstruktyviai, o destruktyviai ir yra sugeneruoti pasinaudojant realiais įsilaužimo pavyzdžiais.

Kiekvienas šablonas užfiksuoja žinias, kaip atskiros atakos dalys yra suprojektuotos ir vykdomos, taip parodydamas problemą ir jos sprendimą iš priešininko pozicijos, ir suteikia patarimų kaip sušvelninti atakos efektyvumą. Atakos šablonai padeda apsiginti nuo atakų, nes leidžia geriau suprasti atskirus atakos elementus ir kaip sustabdyti juos.

Common Attack Pattern Enumeration and Classification[CAP+15] yra atvirai prieinamas paplitusių atakos šablonų katalogas su išsamia informacija apie ataką ir kitas susijusias atakas. Projektas buvo pradėtas 2007 metais JAV saugumo departamento (*angl. U.S. Department of Homeland Security*) kaip dalis programų sistemų užtikrinimo (*angl. software assurance*)

iniciatyvos, kurią pradėjo programinio saugumo ir komunikacijų biuras (*angl. Office of Cybersecurity and Communications*). Dalis NVD CVE įrašų turi nuorodas į CAPEC, todėl esant reikalui darbo metu sukurtas įrankis galės išvesti tokią informaciją.

1.4. CVSS

Common Vulnerability Scoring System[CVS+15] (CVSS) suteikia atvirą karkasą, kuris leidžia pranešti programinės įrangos spragų charakteristikas ir poveikį. Kiekybinis modelis užtikrina pakartojamus ir tikslius matavimus ir leidžia vartotojams pamatyti sudedamąsias charakteristikas, kurios yra naudojamos generuojant spragos vertę[Gal11]. Naujausia šios sistemos versija yra 3.0.

Sistema suteikia tokius privalumus:

- Standartizuotas spragos įvertis. Kai organizacija naudoja vieną spragų vertinimo algoritmą visuose savo produktuose, ji gali taikyti tokias pačias spragos valdymo taisykles, tą patį maksimalų laiką suteikiamą spragai išanalizuoti ir ištaisyti.
- Kadangi sistemos karkasas yra atviras, nekyla abejonių, kodėl spraga buvo įvertinta tokia verte, nes vertinimo kriterijai yra matomi.
- Sistema leidžia suteikti prioritetus rizikoms. Skaičiuojant organizacijos aplinkos vertes, kiekviena spraga įgauna organizacijos kontekstą, tai leidžia suprasti spragos rizikos poveikį organizacijai.

Sistema buvo įvesta 2004m. O 2007m. rugsėjo mėnesį sistemos versija 2.0 tapo Payment Card Industry Data Security Standard[PCI15] dalimi. Norėdami atitikti šį standartą, prekeiviai apdorodami mokėjimo korteles turi parodyti, kad nei viena jų naudojama kompiuterinė sistema neturi spragų, kurių vertė pagal CVSS būtų lygi arba didesnė nei 4.0. 2007 metais NIST padarė CVSS v2.0 jų Security Contrent Automation Protocol dalimi. 2011 metų Balandį CVSS v2.0 tapo tarptautinio spragų vertinimo standarto dalimi.

Naujausia išleista sistemos versija yra 3.0. Naujoje versijoje yra įvesti šie pakeitimai:

- poveikis yra matuojamas ne komponente, kuriame yra spraga, o komponente, kurį spraga paveikia;
- sukurta nauja srities metrika, kuri apima tokia spragas, kai spragai jautrus komponentas ir spragos veikiamas komponentas yra skirtingi vienetai;
- atskirtos atakos, kurioms reikalinga vietinė sistemos prieiga ir fizinės kompiuterio techninės dalies atakos;
- atskirtas vartotojo poveikis atakos sėkmingumui nuo sistemos konfigūracijos;

- autentifikacijos metriką pakeitė reikalingų privilegijų metrika;
- poveikio metrikos pervadintos pagal poveikio lygį;
- aplinkos metrikos tikslo pasiskirstymas ir potencialus šalutinis poveikis buvo pakeistos pakeistais faktoriais (*angl. modified factors*);
- spragų jungimas (*angl. vulnerability chaining*) leidžia pridėti kelias atakoje naudojamas spragas;
- skaitiniai spragos įvertčiai yra paverčiami į 5 taškų kokybinę skalę.

NVD CVE ir X-Force[XF16] įrašuose yra CVSS įvertčiai, todėl darbo metu sukurtas įrankis išves kiekvienos aptiktos spragos vertę ir taip pat vidutinę bendrą sistemos rastų spragų CVSS vertę. Tai leis tiksliau nustatyti sistemos saugumą.

1.5. X-Force

IBM X-Force Exchange[XFE16] yra saugumo žinių platforma, kuri leidžia greitai tyrinėti naujausias globalias saugumo grėsmes, jungti žinias ir bendradarbiauti. Šią platformą palaiko IBM X-Force[XF16] komanda – viena žinomiausių komercinių saugumą tyrinėjančių komandų pasaulyje.

Šioje duomenų bazėje yra virš 96 tūkstančių įrašų apie saugumo spragas. Šioje platformoje dar yra fiksuojami ir didžiausi SPAM siuntėjai bei kenkėjiška programinė įranga.

Duomenų bazė pateikia tokią pagrindinę informaciją apie spragą:

- paveikti programinės įrangos vienetai yra identifikuojami laisva forma;
- tekstinis spragos veikimo principo aprašymas;
- spragos pasekmės (pvz.: suteiktos privilegijos arba gauta informacija);
- žingsniai kaip išvengti spragos;
- pateikiamos nuorodos į spragos aprašymą kituose resursuose (gali būti ir kitos duomenų bazės, spragos aprašymas tiekėjo puslapyje arba tiesiog nuoroda į tiekėjo puslapį);
- spragos poveikio įvertis pagal CVSS.

Suteikiama teisė laisvai ieškoti duomenų, bet neleidžiama atsisiųsti visų duomenų naudoti lokaliai, todėl šiame darbe tokių duomenų panaudoti nepavyks.

1.6. Exploit Database

Exploit Database[ED16] yra CVE suderinamas archyvas viešai žinomų spragų, su spragas išnaudojančiu kodu ir tų spragų pažeidžiama programine įranga. Ši duomenų basė yra skirta saugumo specialistams ir saugumo spragų tyrinėtojams. Kiekvienas įrašas leidžia praktiškai

išbandyti spragą.

Duomenų bazėje yra pateikiami tokie svarbiausi duomenys apie spragą:

- tekstinis spragos aprašymas su pažeidžiamos programinės įrangos pavadinimu;
- kai kurioms spragoms yra pateikta nuoroda, kur galima atsisiųsti pažeidžiamą programinę įrangą;
- kodo pavyzdys demonstruojantis spragos veikimą.

Duomenų bazė, kurioje yra visa informacija apie spragą ir spragą įgyvendinantis kodas, galima atsisiųsti iš oficialaus Exploit Database organizacijos GitHub[GH16] puslapio.

1.7. Apibendrinimas

Iš surinktos informacijos buvo sudaryta pirma lentelė, kurioje yra spragų skaičius ir apie įrašus pateikiama informacija kiekvienoje saugumo spragų duomenų bazėje. OSVDB ir IBM X-Force nepateikia tikslios informacijos apie įrašų skaičių, bet jie teigia, kad įrašų skaičius yra didesnis nei lentelėje nurodytas skaičius. Lentelės eilutė „pasiekiamumas“ nurodo ar naudotojas gali ir turi teisę laisvai atsisiųsti informaciją apie visus įrašus užfiksuotus duomenų bazėje be atskiro leidimo.

1 lentelė. Spragų duomenų bazių palyginimas

Pavadinimas	CVE	OSVDB	X-Force	Exploit Database
Įrašų skaičius	76909	>120000	>96000	36030
Spragos poveikio įvertis	+	-	+	-
Universalus PĮ identifikatorius	+	-	-	-
Išoriniai informacijos šaltiniai	+	+	+	-
Nuorodos į kitas duomenų bazes	+	-	-	+
Aprašymas	+	+	+	-
Pasekmės	-	-	+	-
Spragos sprendimas	-	+	+	-
Spragą išnaudojančio kodo pavyzdys	-	-	-	+
Nuoroda atsisiųsti pažeistą PĮ	-	-	-	+
Pasiekiamumas	+	-	-	+

CVE duomenų bazė turi mažiau įrašų nei OSVDB ar X-Force, bet jos duomenys yra laisvai pasiekiami ir spragų įrašai su programine įranga yra sujungti universaliais identifikatoriais. Taip pat darbe būtų naudinga turėti CVSS spragų įverčius, kuriuos pateikia ši duomenų bazė.

OSVDB bazė turi daugiau įrašų, tačiau jos duomenys nėra laisvai prieinami, todėl ne kiekvienas naudotojas galėtų juos pasiekti, ir įrašai susiejimui su programine įranga nenaudoja unikalių identifikatorių, todėl būtų sudėtinga sieti spragas su programine įranga. Paprasčiausias

būdas būtų tai padaryti per CVE įrašus, bet ne visi spragų įrašai turi nuorodas į CVE. Kitus būtų labai sudėtinga automatiškai susieti su konkrečia programine įranga.

IBM X-Force duomenų bazė taip pat turi daugiau įrašų, bet kaip ir OSVDB, nenaudoja unikalių programinės įrangos identifikatorių, todėl sudėtinga susieti spragas su kažkokia konkrečia sistemos konfigūracija. Darbe praverstų šioje duomenų bazėje fiksuojamas spragos įvertis pagal CVSS ir spragos pasekmės. Duomenų bazės trūkumas yra tai, kad pilni šios duomenų bazės duomenys nėra laisvai pasiekiami.

Exploit Database duomenų bazė turi mažiausiai įrašų apie spragas, bet jos duomenys yra laisvai prieinami. Patogu, kad įrašai turi nuorodą į pažeidžiamą programinę įrangą ir yra pridėtas spragą demonstruojantis kodas, tačiau šie duomenys nėra naudingi nustatant sistemos saugumą.

Dėl priežasčių išvardintų šiame poskyryje, buvo nuspręsta toliau darbe kaip informacijos apie saugumo spragas šaltinį naudoti CVE duomenų bazę.

2. ĮDIEGTOS PROGRAMINĖS ĮRANGOS APTIKIMAS

Norint patikrinti įdiegtos programinės įrangos saugumą, turime sužinoti kokia yra sistemos konfigūracija, kitaip sakant, turime sužinoti kokia OS ir programinė įranga yra įdiegta ir kokios versijos. Tai įvykdyti galime skirtingais būdais.

Kai kurios programos praneša kokia versija yra įdiegta, kai yra kviečiamos su specialu parametru (dažnai „-v“ ar panašiu). Šis būdas yra geras tuo, kad yra nepriklausomas nuo OS, tačiau yra labai specifiškas programinei įrangai. Kai kuri programinė įranga gali neturėti tokio parametro arba apskritai gali neturėti vykdomų failų (pvz.: įvairios bibliotekos).

Programinė įranga suteikianti kokią nors tarnybą, dažnai turi pateikti savo versijos numerį, kad tos tarnybos klientas žinotų kokią protokolo versiją naudoti bendraujant su ja. Šis būdas yra specifinis kiekvienai programai ir potencialiai gali būti klaidinantis, nes kai kuriais atvejais gali būti naudinga atvaizduoti klaidingą versiją.

Universalesnis būdas nustatyti programos versiją yra pasinaudojant operacinės sistemos suteikiamomis priemonėmis. Kadangi programinė įranga yra dažnai įdiegiama, ištrinama ir atnaujinama, visos operacinės sistemos turi priemones sekti kokia programinė įranga yra įdiegta, kokius failus ji įdiegė, kokius naudoja konfigūracijai, kokia programinės įrangos versija yra įdiegta. Tokios priemonės bus apžvelgtos kituose poskyriuose.

2.1. Windows OS

Vienas būdas Windows operacinėje sistemoje sužinoti įdiegtų programų sąrašą yra pasinaudojant įdiegtų programų sukurtais įrašais OS registre. Tai galime padaryti įvedę tokio formato kelią „HKEY_LOCAL_MACHINE\SOFTWARE\[Tiekėjo pavadinimas]\[Programos pavadinimas]“. Šio būdo trūkumas yra tas, kad negalime būti tikri, kad visos programos turės būtent tokio formato įrašus.

Microsoft operacinėse sistemose pradedant nuo Windows 2000 (kai kuriose senesnėse versijoje galima įdiegti atskirai) yra įdiegti valdymo įrankiai Windows Management Instrumentation[WMI15]. Jų tikslas yra suteikti nuo aplinkos nepriklausomą prieigą prie sistemos valdomų duomenų. Šiuos įrankius patogu pasiekti iš komandinės eilutės naudojant Microsoft programą Windows Management Instrumentation Command-line. Pavyzdžiui su šia programa galime gauti sistemoje veikiančių procesų sąrašą. Taip pat ši programa gali išspausdinti ir sistemoje įdiegtų programų ir jų versijų sąrašą, tai galima atlikti paleidus programos interaktyvų rėžimą ir įvedus tokią komandą „/output:C:\Programos.txt product get name,version“.

Galima WMI pasinaudoti ir tiesiogiai per Windows Installer Provider, kuris leidžia WMI naudojančioms programoms pasiekti informaciją apie su Windows Installer suderinamas įdiegtas programas, t.y. apie standartiniu būdu įdiegtas programas. Jis yra įdiegtas visose 32 bitų Windows OS ir gali būti įdiegtas atskirai 64 bitų OS. Jis leidžia gauti informaciją apie įdiegtą programinę įrangą per WMI klases. Visas įdiegtas programas atspindi klasė Win32_Product. Tarp kitų programos ypatybių ši klasė grąžina programos vardą per kintamąjį Name ir programos versiją per kintamąjį Version.

Kadangi sistemose, kuriose yra svarbus saugumas dažniausiai yra naudojamos kitos operacinės sistemos, buvo nuspręsta šiuo metu neįgyvendinti Windows palaikymo. Vėliau šį palaikymą galima nesudėtingai pridėti pasinaudojus šiame skyriuje aprašytais būdais.

2.2. Linux OS

Įprastai programinės įrangos diegimą Linux OS kontroliuoja programa vadinama paketų valdykle (*angl. package manger*). Ji seka, kokie programinės įrangos vienetai yra įdiegti, kokios jų versijos, kokius failus jie įdiegė. Tai paketų valdyklei leidžia efektyviai įdiegti, atnaujinti ir šalinti paketus iš sistemos. Paketais Linux OS įprasta vadinti programas arba bibliotekas, kurios yra įdiegiamos kartu su kitu paketu. Kai kurios paketų valdyklės gali palaikyti ir paketų grupes arba metapaketus. Tai tokie paketai, kurie dažnai yra naudojami kartu. Pavyzdžiui, jei sistemoje naudojame skaičiuoklę, tai tikėtina, kad sistemoje bus naudojamos ir kitos biuro paketų programos, todėl visus biuro paketus naudinga laikyti vienu didesniu biuro programų metapaketu ir leisti diegti tiek ir kartu, tiek ir atskirai.

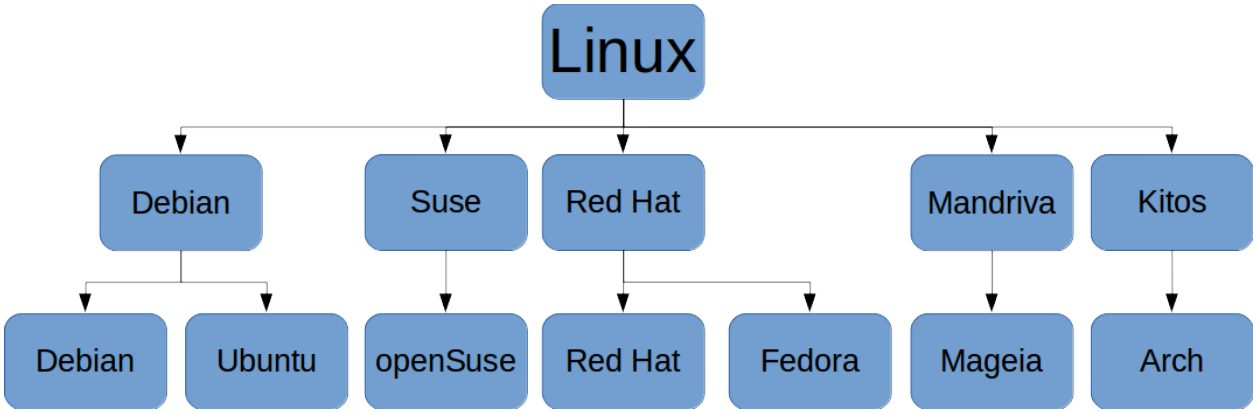
Linux yra tik operacinės sistemos branduolio pavadinimas ir kadangi ši OS įprastai yra naudojama su GNU projekto programomis, tikslesnis OS pavadinimas yra GNU/Linux. Ši operacinė sistema praktiškai visada yra pateikiama su kitomis programomis. Toks rinkinys yra vadinamas distribucija (*angl. distribution*).

Pagal tinklapio, kuris registruoja skirtingas Linux distribucijas ir informaciją apie jas, DistroWatch[UI15] duomenis, šiuo metu yra 278 aktyvios, 64 pristabdytos (*angl. dormant*) ir 458 nebeplėtojamos (*angl. discontinued*) distribucijos. Kadangi Linux branduolys yra atviro kodo nemokamas ir laisvai pasiekiamas, bet kokia organizacija ar žmogus gali sukurti savo distribuciją, todėl neįmanoma užtikrinti patikimo visų jų registravimo kokioje nors duomenų bazėje. Vadinasi, jų gali būti dar daugiau nei rodo duomenys.

Skirtingos distribucijos gali naudoti skirtingas arba vienodą paketų valdyklę. Pagal susitarimą, paketų valdyklė yra susiejama su viena distribucija, kuri pirmoji pradėjo naudoti tokią paketų valdyklę, o kitos ją naudojančios yra laikomos išvestinėmis. Kadangi, sukurti

paketų valdyklę reikia daugiau resursų nei išvestinę distribuciją, skirtingų paketų valdyklių ir taip pat naujų neišvestinių distribucijų yra daug mažiau nei išvestinių. Kadangi, išvestinės naudoja tą pačią paketų valdyklę, šio darbo apimtyje laikoma, kad išvestinės distribucijos kartu su pagrindine yra viena ir ta pati distribucija.

Toliau populiarias distribucijas padalinsime į grupes pagal paketų valdyklę. Vadinasi, kelios populiarios distribucijos galės atsidurti vienoje grupėje, o kai kurios rečiau naudojamos distribucijos nebus paminėtos.



1 pav. Aptinkamų distribucijų šeimų schema

Daugelį distribucijų galima atpažinti terminale paleidus lsb_release įrankį. Kad toks įrankis konkrečioje sistemoje yra įdiegtas, programa sužino iš jo naudojamų lsb_release failų sistemoje. Tuo atveju, kai toks įrankis nėra įdiegtas, distribucija yra atpažįstama iš pavadinimas_release arba pavadinimas_version failų. Šie failai yra saugomi distribucijos „etc“ kataloge. Debian distribuciją galima atpažinti pagal „/etc/debian_version“ failo egzistavimą. Šiuo metu pasinaudojant minėtais būdais yra aptinkamos tokios pagal šeimas išskirtos Linux distribucijos:

1. Debian tipo;
    1. Ubuntu[Can16];
    2. Debian[Deb16];
2. Suse tipo;
    1. openSuse[Sus16];
3. Red Hat tipo;
    1. Centos[CP16];
    2. Fedora[RH16a];
    3. Red Hat Enterprise[RH16b];
4. Mandriva tipo;
    1. Mageia[MAG16];
5. Kitokios;

1. Arch[AL16].

Visos populiariausios pagal DistroWatch[UI15] duomenis distribucijos yra aptinkamos, tačiau programa sukurta taip, kad būtų paprasta pridėti naujų distribucijų palaikymą. O jei jos išvestinės, tai tam užtektų pridėti vieną sąlygą, kuri nurodytų kokio tipo paketų valdyklę naudoja ši distribucija. Tolesniuose poskyriuose bus apžvelgtos palaikomos distribucijos ir bus paaiškinta kaip iš jų gaunami duomenys apie įdiegtus paketus.

2.2.1. Debian

Debian[DEB15] yra viena pirmųjų Linux distribucijų. Ji buvo paskelbta 1993. Tai viena iš priežasčių, kodėl tiek daug populiarių distribucijų yra išvestos iš Debian. Ši distribucija naudoja Advanced Package Tool (APT) paketų valdyklę ir paketus laiko DEB failuose. Dažnai patogiau distribucijos priemonėmis pasinaudoti per pagalbinį įrankį (pvz.: aptitude arba apt-get).

Sužinoti visus įdiegtus paketus ir jų versijas galime su komanda: „dpkg-query -W -f '${package} ${version}\n'“. Vietoje apt yra naudojama ši žemesnio lygio dpkg komanda, kad būtų užtikrintas veikimas su kitomis išvestinėmis iš Debian distribucijomis.

Ubuntu yra Debian paremta distribucija, kurią sukūrė Marko Shuttlewortho įkurta Canonical Ltd. Nuo naujausios versijos (16.04) ši distribucija naudoja naują paketų valdyklę snappy, tačiau vis dar palaiko ir įprastinę dpkg, todėl yra suderinama su kuriamu įrankiu ir bus toliau naudojama sukurto įrankio testavime.

2.2.2. Red Hat

Red Hat[RH15] yra Red Hat kompanijos 1995m. sukurta distribucija. Daug kitų distribucijų yra daugiau ar mažiau paremtos ja: Red Hat Enterprise, Fedora, Cent OS, Mandriva.

Naujausios distribucijos versijos (pervadintos į Red Hat Enterprise) naudoja tą pačią paketų valdyklę, kaip ir Cent OS bei Fedora, tai YUM paketų valdymo įrankis, kuris naudoja RPM formato paketus.

Nors ir Red Hat naudoja yum kaip paketų valdyklę, analogiškai kaip ir su prieš tai aprašytomis Debian distribucijomis, šiame darbe buvo pasinaudota universalesne žemesnio lygio programa „rpm“. Tai leis perpanaudoti šią komandą ir kitose RPM formato paketus naudojančiose distribucijose. Toliau bus aprašytos kitos tokios distribucijos, su kuriomis buvo testuojama sukurta programa. Pasinaudoję ja paketų sąrašą su versijomis galime sužinoti įvedę komandą:

„rpm -qa –qf '%{NAME} %{VERSION}\n'“.

OpenSuse yra populiari atvira vokiečių kompanijos „Novell“ sukurtos distribucijos

„SUSE“[SUS15] (anksčiau „SuSE“) versija[OPE15]. Ji garsi konfigūraciniu įrankiu „Yast“. Paketų valdymui ji naudoja Libzypp biblioteką, kuri gali būti pasiekiama grafiniu būdu iš „Yast“ arba per Zypper komandinės eilutės programą. Ši distribucija naudoja RPM stiliaus paketus ir todėl yra gerai palaikoma įrankio.

Centos („Community Enterprise Operating System“) yra atviro Centos projekto distribucija, tai bendruomenės kuriamas, bet verslo lygio produktas, kuris yra programiškai suderinamas su „Red Hat Enterprise Linux“. O nuo 2014 metų šis projektas buvo perduotas „Red Hat“ organizacijai, tačiau nebuvo sujungtas su pačia „Red Hat Enterprise Linux“ distribucija. Šioje distribucijoje didelis dėmesys buvo skirtas jos suderinamumui su Red Hat, todėl ji gerai palaikoma kuriamo įrankio.

Mandrake yra labai sena distribucija, kuri buvo paremta Red Hat Linux. Vėliau ji buvo pervadinta į Mandriva. Deja, šios distribucijos kūrimas yra sustabdytas, bet vis dar leidžiama atvira jos versija vadinama Mageia[MAG15]. Mageia naudoja tokį patį paketų formatą, kaip ir Red Hat distribucijos, tačiau turi savo paketų valdyklę „URPMI“. Suderinamumui palikta rpm komanda leis lengvai patikrinti ir šios distribucijos saugumą.

Fedora (ankstesnis pavadinimas Fedora Core) yra bendruomenės kuriama ir Red Hat finansuojama distribucija. Naujausios versijos naudoja Yum paremtą paketų valdyklę DNF. Nors ir šioje distribucijoje yra skiriamas didelis dėmesys naujoms technologijoms, ji palaiko ir rpm komandas, todėl yra suderinama su kuriamu įrankiu.

2.2.3. Arch Linux

Trečia paketų formatų grupė yra tgz formato paketai (tar gzip failai). Šis paketų formato tipas nurodo tik populiarų failų suspaudimo metodą, kuris dažnai naudojamas Unix stiliaus OS ir nenurodo jokio kito standartizavimo. Paketo turinys priklauso tik nuo paketų valdymo programos ypatybių. Gali būti naudojami ir kiti failų suspaudimo būdai. Pavyzdžiui populiari tokio tipo distribucija Arch Linux[AL15] anksčiau naudojo tgz paketus, tačiau vėliau perėjo prie modernesnio lzma suspaudimo (Linux dažniausiai vadinamas xz, o Windows OS – 7z) ir paketų pavadinimams naudoja .pkg.tar.xz formatą.

Arch Linux distribucija ir ja paremtos kitos distribucijos naudoja pacman paketų valdyklę. Ji išspausdins visų įdiegtų paketų sąrašą įvedus komandą „pacman -Q“. Populiariausios Arch Linux paremtos distribucijos yra: Manjaro Linux, Antergos, Netrunner, ArchBang Linux, Chakra GNU/Linux ir k.t. Visas šias distribucijas palaiko darbe kuriamas įrankis.

Kitos distribucijos naudojančios šį paketų formatą, gali naudoti ir kitokias paketų valdykles nei pacman ir įdiegtų programų sąrašą jose galima sužinoti joms pritaikytomis komandomis.

3. FORMALIZAVIMO BŪDAI

Formalizavimo būdai leidžia sukauptas neišreikštas žinias aprašyti aiškiai ir tiksliai, kad vėliau galėtume pasinaudoti jomis priimdami kažkokius sprendimus. Kitaip sakant, informaciją apie pasaulį paversti kompiuteriui suprantama forma. Sukurta skirtingų būdų žinias užrašyti formaliai.

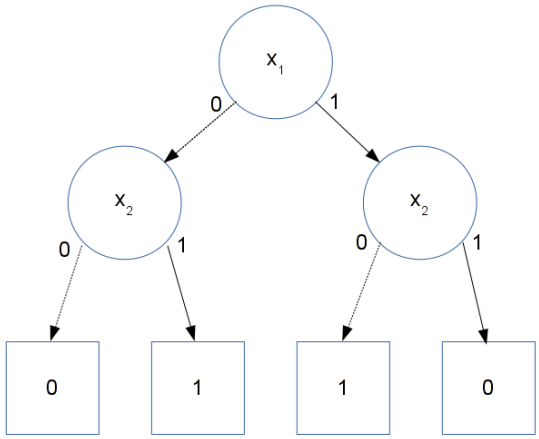
Pats paprasčiausias būdas užrašyti žinias yra grafiniu pavidalu. Tokie formalizavimo būdai yra vadinami diagraminiais ir yra paprastai skaitomi ir suprantami. Tokių formalizavimo būdų pavyzdys yra koncepciniai žemėlapiai[NG84].

Formalizavimo būdai yra artimai susiję su automatiniu samprotavimu (*angl. automated reasoning*), nes vienas svarbiausių jų tikslų yra išreikštinai užrašyti žinias, kad galėtume apie jas samprotauti, daryti išvadas, išvesti naujas žinias. Praktiškai visi formalizavimo būdai turi kokią nors samprotavimo formą.

Pasirinkę tinkamą formalizavimo būdą, galėsime sukurti gerą žinių modelį, kuris leis užtikrintai įvertinti, ar surinkti duomenys vienareikšmiškai įvertina sistemos saugumą.

3.1. Dvejetainiai sprendimų medžiai

Dvejetainiai sprendimų medžiai yra duomenų struktūra naudojama loginių funkcijų atvaizdavimui. Abstrakčiai, dvejetainiai sprendimų medžiai gali būti laikomi suspaustu aibių arba ryšių atvaizdu. Ją sukūrė Randal Bryant iš CMU ir pirmą kartą paminėjo 1986m. straipsnyje[Bry86]. Don Knuth jas įvardino kaip vieną iš tikrai fundamentinių duomenų struktūrų sukurtų per pastaruosius dvidešimt metų[Knu15]. Antrame paveiksle yra pateiktas dvejetainio sprendimų medžio pavyzdys atvaizduojantis sumos moduliu 2 (XOR) funkciją.



2 pav. Dvejetainis sprendimų medis

Dvejetainiai sprendimų medžiai yra tokie medžiai, kurie yra sudaryti iš tokių elementų:

- įvykio viršūnė rodo galimų situacijų aibę ir yra žymima apskritimu;
- sprendimo viršūnė rodo priimamą sprendimą ir yra žymima stačiakampiu;
- šaka yra elementas jungiantis dvi medžio viršūnes.

Sprendimų rezultatas – konkrečios situacijos įvertis, kuris yra gaunamas einant vieninteliu

keliu nuo medžio šaknies iki sprendimo viršūnės. Dar gali būti vadinamas konkrečia interpretacija.

Tokie medžiai dažnai turi vienodų, kitaip sakant izomorfinių, dalių. Kompaktiškesnį medį be pasikartojimų galima gauti sukūrus rikiuotą dvejetainį medį ir jam pritaikius mažinimo taisykles (*angl. reduction rules*). Tokios taisyklės yra trys. Jos yra taikomos tol, kol nebegalima pritaikyti nei vienos taisyklės. Rezultatas yra sumažintas ir surikiuotas dvejetainis sprendimų medis (*angl. reduced ordered binary decision tree*).

Redukcijos taisyklės:

- visos sprendimo viršūnės, kurių reikšmė yra vienas yra paverčiamos į vieną viršūnę, o viršūnės, kurių reikšmė yra nulis, yra paverčiamos į kitą viršūnę;
- sujungiami izomorfiniai pomedžiai;
- pašalinamos perteklinės viršūnės, tai tokios kurių abu vaikai turi vienodas reikšmes (t. y. rodo į vieną ir tą pačią viršūnę).

Pritaikius vieną taisyklę, reikia iš naujo taikyti kitas taisykles. Pašalinus perteklinę viršūnę gali atsirasti izomorfinių grafų ir atvirkščiai.

Dvejetainiai sprendimų medžiai yra kanoninis dvejetainės formulės atvaizdas. Vadinasi, dvi ekvivalenčios formulės bus atvaizduotos vienodais medžiais.

Šis būdas nėra populiarus, nes medžio dydis priklauso nuo kintamųjų tvarkos. Kai kurioms tvarkoms medis didėja polinomiškai, o kai kurioms – eksponentiškai. Kintamųjų tvarkos nustatymo uždavinys yra NP pilnas. Tipiškai jis sprendžiamas pasitelkiant euristikas. Šios euristikos gali būti statinės arba dinaminės (t.y. kinta vykdant medžio operacijas). Tipiškai dinaminės euristikos grąžina mažesnius medžius. Kadangi negalime užtikrinti, kad konkrečiu atveju medis nedidės eksponentiškai, šis formalizavimo būdas nėra tinkamas.

3.2. Deskriptyvi logika

Deskriptyvi logika (*angl. description logics*) kuria formalizmus, kurie suteikia pasaulio aukšto lygio aprašymus, kurie gali būti efektyviai panaudoti protingose programose[NB03]. Šiuo atveju protinga programa laikome programą, kurį iš išreikštinių duomenų gali padaryti neišreikštinius sprendimus. Kitaip sakant, tai programa sugebanti samprotauti. Formalizmas leidžia turėti konkrečią ir formalią sintaksę su nedviprasmiška reikšme. Aukšto lygio aprašymai reiškia tik būtinų savybių apibūdinimą, paliekant kitas. Ši logika turi būti efektyviai panaudojama praktiniuose įrankiuose. Pirma deskriptyvios logikos sistema KL-ONE buvo Brachman ir Schmolze sukurta 1985 metais[BS85].

Deskriptyvios logikos istoriją sudaro šie pagrindiniai etapai:

1. Aštuoniasdešimtaisiais deskriptyviomis logikomis paremtos žinių atvaizdavimo sistemos (Back, K-Rep, Loom, Meson) buvo kuriamos pasinaudojant nepilnais struktūrinio samprotavimo algoritmais.
2. Ankstyvais devyniasdešimtaisiais buvo sukurti tablo (*angl. tableau*) paremti algoritmai, kurie leido efektyviai samprotauti pasinaudojant ekspresyvesnėmis deskriptyviomis logikomis (Kris, Crack).
3. Nuo devyniasdešimtųjų vidurio buvo kuriami greitai veikiantys samprotautojai labai ekspresyvioms deskriptyvioms logikoms (FaCT, RACER, CEL, KAON 2).
4. Sukurta Web Ontology Language (OWL-DL) kalba paremta labai ekspresyvia deskriptyvia logika, taip pat sukurti praktiniai samprotautojai ir redaktoriai šiai kalbai.

Aprašomi santykiai tarp realaus pasaulio objektų ir jų simbolinių modelių. Simbolinės išraiškos susiejimas su pasaulio abstrakcija. Tiesos sąvoka leidžia nustatyti ar simbolinė išraiška būtų teisinga vertinamame realiame pasaulyje.

Svarbus yra adekvatus ekspresyvumas. Per mažas ekspresyvumas neleis užfiksuoti visų reikalingų apie problemą žinių, o per didelis reikš per didelį informacijos kiekį, t. y. bus nebūtinų duomenų.

Samprotavimas turi suteikti išvestinių žinių apie atvirai užrašytas žinias. Samprotavimo procedūra tūri būti: pagrįsta, pilna ir atsakymas turi būti pateiktas per baigtinį laiką.

Tam tikros srities formalizavimo procedūra:

- apibrėžti svarbiausias srities sąvokas (klasės, santykiai objektai);
- nurodyti kaip šių sąvokų interpretavimas gali būti ribojamas;
- nustatyti apibūdinimų ir ribojimų poveikį: paveldėtų klasių santykiai, egzempliorių santykiai.

Srities modelio pavyzdžiai:

- klasės gali būti Žmogus, Autorius, Dalyvis, Bilietas, Paskaita;
- santykiai (rolės) gali būti patinka, perduoda, apsilanko;
- objektai (egzemplioriai) gali būti Jonas, Petras_1416090;
- apribojimai gali būti kiekvienoje paskaitose kalbą jos autorius, į paskaitos dalyką turi būti užsirašę nei mažiau nei penki studentai.

Spragų objektai saugo įvairią informaciją apie spragą ir su ja susijusias kitas spragas (pvz.: per atakos šablonus), todėl tokia informacija yra struktūrizuota. Ši struktūra yra pakankamai laisva, nes nemaža dalis informacijos nėra būtina. Tokiai informacijai formalizuoti deskriptyvi

logika nėra patogi.

3.3. Ontologijos

Ontologijos buvo filosofijos sritis jau Aristotelio laikais. Tuo metu tai buvo egzistencinis mokslas apimantis viską kas yra pasaulyje[Gar10].

Ontologija yra žinių atvaizdavimo sistema paremta deskriptyvia logika[JWX+08]. Ši sistema leidžia ne tik vienodai aprašyti duomenų sritį, bet ir dalintis tais duomenimis tarp žmonių ir heterogeninių programų sistemų[Fen11].

Šiuo metu ontologijos plačiai naudojamos informatikoje ir dirbtinio intelekto srityse. Dirbtinio intelekto sistemose samprotavimas yra vykdomas pasinaudojant problemų sprendimo metodais, o statinės srities žinios žiniomis pagrįstose sistemose yra aprašomos pasinaudojant ontologijomis[Fen11]. Šiuo atžvilgiu, ontologijos yra srities terminų užfiksavimas jų konceptualizacijomis. Konceptualizacija yra abstraktus kažkurio pasaulio aspekto modelis, kuris yra formuojamas aprašant svarbiausias sąvokų savybes ir jų ryšius[BHS04]. Elektroninių prietaisų ontologijoje galėtų būti įvairūs koncepciniai elementai, pavyzdžiui srovės stipris, diodas, diodinis tiltas, ir ryšiai tarp šių elementų, pavyzdžiui diodinis tiltas yra elektrinis prietaisas, o diodas yra diodinio tilto komponentas. Taigi, ontologija yra žodynas terminų, kurie naudojami aprašant kažkokią sritį, ir informacijos rinkinys, kuris aprašo tą sritį pasinaudodamas terminais.

Ontologijomis yra laikomas bendras ir paplitęs supratimas apie kažkokią sritį, kurį galima perduoti tarp žmonių ir vienodai suprantamas kompiuterinių programų. Pavyzdžiui elektroninių komponentų ontologiją galėtų pakartotinai panaudoti skirtingi elektroninių komponentų gamintojai, papildydami ją konkrečių jų gaminamų komponentų informaciją, kurią užfiksuotų vienodais terminais. Tai leistų pakartotinai panaudoti žinias ir lengviau palyginti skirtingų gamintojų produktus. Ontologija yra formalus ir aiškus apibūdinimas bendrai žinomai sąvokai[SBF98].

Ontologijos palengvina pasidalinimą informacija ir jos perpanaudojimą dirbtiniame intelekte. Kadangi ontologijos suteikia bendrą supratimą apie konkrečią aprašomą sritį, jos pradedamos naudoti ir kitose su informatika susijusiose srityse.

Daugumoje šaltinių yra sutariama, dėl šių ontologijų savybių:

- Ontologijos sritį sudaro objektai.
- Objektai gali būti sudaryti iš komponentų.
- Objektai gali turėti savybių arba atributų, kuriems gali būti priskirtos reikšmės.

- Tarp objektų gali egzistuoti įvairūs ryšiai.
- Savybės ir ryšiai nėra pastovūs, jie gali kisti.
- Kažkuriuo laiko momentu, gali nutikti įvykiai.
- Kažkurį laiką gali vykti procesai, kuriuose gali dalyvauti objektai.
- Sritis ir jos objektai, gali būti skirtingose būsenose.
- Įvykiai gali iššaukti kitus įvykius arba būsenas.

Ontologijos yra labai ekspresyvios, todėl leidžia patogiai išreikšti bet kokios srities objektus ir jų savybes. Toks ekspresyvumas apsunkina samprotavimą, todėl būtų patogu objektus užrašyti formalesne logine kalba. Šiuo tikslu ontologijos naudoja deskriptyvią logiką.

3.4. Apibendrinimas

Dvejetainiai sprendimų medžiai tinka bet kokių funkcijų atvaizdavimui, tačiau didėjant kintamųjų skaičiui sunku išlaikyti medžio dydį. Blogiausiu atveju didėjant kintamųjų skaičiui, medžio dydis didės eksponentiškai. Kadangi darbe reikės nagrinėti didelius informacijos kiekius (dešimtys tūkstančių įrašų apie spragas ir tūkstančiai įdiegtų paketų), buvo atsisakyta šio formalizavimo būdo.

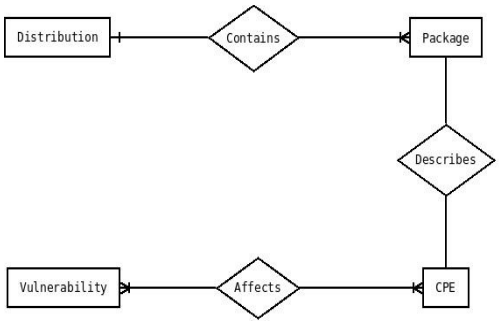
Deskriptyvi logika leidžia aprašyti reikalingas sąvokas ir jomis samprotauti. Deskriptyvios logikos sintaksė yra formali, todėl reikalingą informaciją formalizuoti pasinaudojant ją būtų nepatogu.

Ontologijos yra deskriptyvia logika pagrįstas formalizavimo būdas, kuris, dėl savo tinkamo ekspresyvumo ir patogios kalbos, leidžia patogiai formalizuoti informaciją. Taip pat vėliau patogu yra pasinaudoti samprotautojais ir pagal tai gauti naują informaciją apie sritį.

Ontologijos leidžia efektyviausiai užrašyti informaciją ir vėliau patogiai samprotauti ja, todėl ontologijos ir bus naudojamos toliau darbe informacijos formalizavimui. Kitame darbo skyriuje bus aprašoma ši ontologija, kuri buvo sukurta naudojant OWL DL kalbą, todėl bus aprašyta šios kalbos terminais.

4. ONTOLOGIJOS FORMAVIMAS

Praeituose darbo skyriuose aprašyta, kaip surinkti duomenis, kurie yra reikalingi, kad galėtume nustatyti sistemos saugumą. Šiame skyriuje ontologijos klasės aprašys duomenų struktūrą ir ryšius tarp atskirų elementų, kurie tarpusavyje suriš šiuos elementus. Kiekviename skyrelyje bus aprašomos atskiros ontologijos klasės. Ontologijos klasių ryšiai yra pavaizduoti trečiame paveiksle esančioje diagramoje. Ontologiją sudaro keturios klasės, kurias sujungia trys ryšiai. Visi šie ryšiai yra dviejų krypčių. Kaip pamatysime tolesniuose skyriuose, kai kurių ryšių pavadinimai ontologijoje skiriasi priklausomai nuo jo krypties.



3 pav. Klasių ryšių schema

4.1. Klasė „Distribution“

Distribution klasė ontologijoje atspindi distribucijos objektą. Vienoje ontologijoje gali būti daugiau nei vienas šios klasės objektas. Kiekvienas toks objektas yra susietas su bent vienu Package klasės objektu. Šis ryšis ontologijoje yra aprašomas aksioma „consistsOf some Package“, o matematine logika: $Distribution \equiv \exists consistsOf.Package$. Package klasės objektai yra surišti su atitinkamu distribucijos objektu atvirkštiniu ryšiu, kuris pavadintas „containedIn“.

4.2. Klasė „Package“

Package klasė ontologijoje atspindi paketo objektą. Vienoje ontologijoje gali būti daug tokių objektų. Kiekvienas objektas būtinai priklauso vienam distribucijos objektui. Kiekvienas Package objektas yra paveldėtas arba iš SafePackage, arba iš VulnerablePackage objekto. DL kalba sakome, kad Package klasė yra lygi reikšmei „SafePackage or VulnerablePackage“, o logikos terminais tai galima užrašyti: $Package \equiv SafePackage \cup VulnerablePackage$. Tai nereiškia, kad reikia išreikštinai nurodyti iš kokios klasės paveldi kiekvienas Package objektas, nes tam bus naudojamas samprotautojas.

Objektai, kurie pavelti iš SafePackage, negali turėti nei vieno ryšio su CPE objektais. Ši savybė yra aprašoma aksioma „isDescribedBy max 0 CPE“ arba matematiškai: $SafePackage \equiv Package \cap \nexists isDescribedBy.CPE$.

VulnerablePackage visada yra susieti su vienu CPE objektu. Ši savybė aprašoma aksioma

„isDescribedBy some CPE“ arba matematiškai:

$VulnerablePackage \equiv Package \cap \exists isDescribedBy.CPE$. Kadangi CPE yra unikalus identifikatorius, tai jis aprašys visadą tik vieną paketą ir atvirkščiai, tik vienas paketas gali atitikti vieną CPE objektą.

4.3. Klasė „CPE“

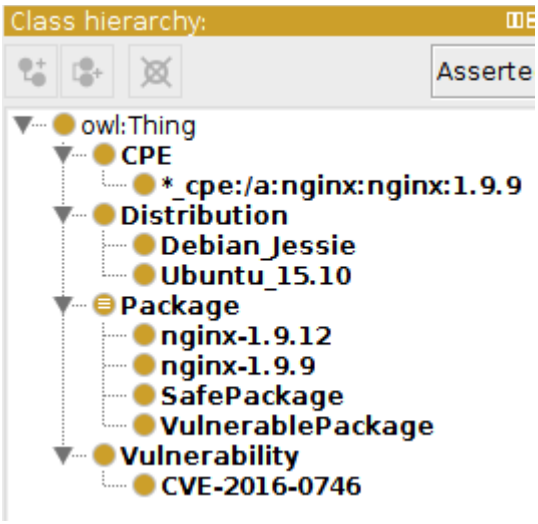
CPE atspindi NVD duomenų bazėje esančius objektus, kurie originaliai aprašo kiekvieną paketą, kuris gali turėti spragų. Šis objektas aprašo visas galimas paketo sąvybes, o ne tik jo pavadinimą ir versiją. Pavyzdžiui spraga gali įvykti tik vienai kalbai skirtoje programoje, jei klaidą iššaukia simbolis, kuris egzistuoja tik tokioje kalboje. Vienas CPE objektas gali būti surištas su keliais paketais. DL kalba CPE savybės yra aprašomos aksiomomis „isAffectedBy some Vulnerability“ ir „describes some Package“. O matematinė formulė: $CPE \equiv \exists isAffectedBy.Vulnerability \cap \leqslant 1 describes.Package$.

4.4. Klasė „Vulnerability“

Vulnerability objektas atspindi vieną saugumo spragą iš NVD duomenų bazės. Vienas toks objektas turi būti surištas su bent vienu CPE objektu. Šį objektą DL kalba aprašo aksioma „affects some CPE“ ir matematinės logikos reiškinys : $Vulnerability \equiv \exists affects.CPE$. Kiekvienas toks objektas turi skaitinę reikšmę CVSS, kuri parodo kiek ši spraga yra pavojinga.

5. METODAS

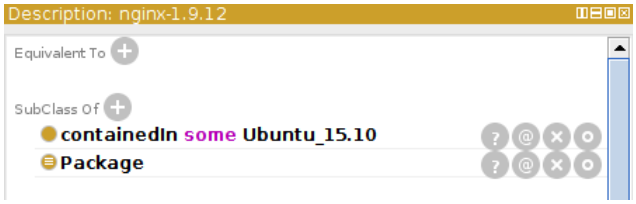
Praeitame skyriuje aprašyta bendra ontologijos sandara. Šios ontologijos tikslas yra nustatyti ar konkreti sistema yra saugi, todėl ontologiją reikia užpildyti ir duomenimis apie sistemos konfigūraciją, ir duomenimis apie egzistuojančias spragas. Šių duomenų išgavimas yra aprašyti praeituose darbo skyriuose. Darbo metu sukurtoje ontologijoje tai daroma rankiniu būdu, tačiau anksčiau darbe minėtais būdais visus šiuos duomenis galima gauti ir programiškai. Tada norint patikrinti sistemos saugumą, užteks visus šiuos duomenis suvesti į ontologiją Metodo pavyzdžiams buvo naudojama Protege 5 beta 24 programa. Ontologijos hierarchija yra pavaizduota ketvirtame paveiksle.



4 pav. Klasių hierarchija

Ontologijos užpildymą aprašo toks metodas:

1. Kiekvienos nagrinėjamos sistemos distribucija yra įvedama kaip klasė, kuri paveldi iš klasės Distribution.
2. Kiekvienam sistemoje įdiegtam paketui yra įvedama po klasę, kuri paveldi iš klasės Package.

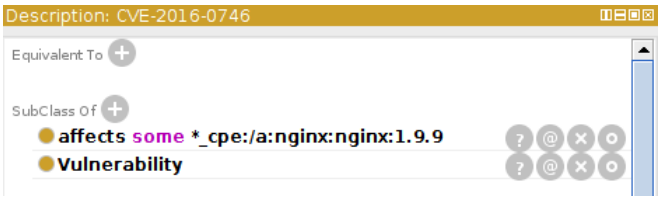


5 pav. Paketo klasės aprašymas

6 pav. Distribucijos klasės aprašymas

3. Kiekviena paketo klasė yra sujungiama su distribucijos klase ryšiu containedIn, o distribucija su ta pačia klase ryšiu – consistsOf. Šių ryšių pavyzdžiai pavaizduoti atitinkamai penktame ir šeštame paveiksluose.
4. Kiekvienam spragos objektui iš NVD duomenų bazės yra sukuriama po klasę, kuri

paveldi iš Vulnerability.



7 pav. Spragos klasės aprašymas

8 pav. CPE klasės aprašymas

5. Kiekvienai spragos klasei sukuriama bent po vieną CPE klasę, kuri yra surišama su spragos klase ryšiu isAffectedBy, o spraga su šia klase ryšiu – affects (atitinkamai septintas ir aštuntas paveikslai).

Sistemos saugumo patikrinimą aprašo toks metodas:

1. Užkraunama ontologija.
2. Patikrinama ar ontologija yra korektiška (*angl. consistent*), jei randama loginių klaidų, tai išmetamas pranešimas ir baigiamas darbas, kitu atveju – tęsiama.
3. Samprotautojas išrenka visas klases paveldinčias iš klasės SafePackage išskyrus owl:Nothing klasę. Šios klasės atspindi pažeidžiamus paketus.
4. Kiekvienai tokiai klasei randama su ja surišta CPE klasė.
5. Kiekvienai tokiai klasei randama spragos klasė. Vartotojui yra išvedamas sąrašas spragų su nuorodomis į NVD tinklapyje esantį spragos aprašymą.

6. EKSPRESYVUMAS IR SAMPROTAVIMO GREITIS

Darbe sukurta ontologija yra sukurta deskriptyvios logikos kalba OWL, o ši kalba dėl savo ekspresyvumo yra sudėtinga skaičiavimų prasme. Pirma kalbos versija yra SHOIN ekspresyvumo ir jos sudėtingumas yra NEXPTIME-complete[HPH03]. O OWL 2 yra SHOIQ ekspresyvumo, todėl jos sudėtingumas yra dar didesnis – 2NEXPTIME-complete[GHM+08].

Yra įvairių rūšių deskriptyvios logikos (*angl. description logics*). Visos jos yra vertinamos pagal savo ekspresyvumą, kuris priklauso nuo panaudotų aksiomų. Darbo metu sukurtos ontologijos ekspresyvumas yra ALCIQ, nes ontologiją sudaro tokios bazinės logikos:

- Bazinė deskriptyvios logikos kalba – AL, nes ontologijoje yra panaudotas elementarus neigimas ir paprasti apribojimai.
- C reiškia sudėtingų sampratų neigimą. Sudėtingos sampratos yra tokios, kurios susideda iš kelių kitų sampratų. Mūsų ontologijoje SafePackage aprašo aksioma „not VulnerablePackage“, o VulnerablePackage aprašo kelios aksiomos.
- I reiškia atvirkštines savybes. Pavyzdžiui sukurtoje ontologijoje distribucija turi ryšį su paketu consistsOf, o paketas turi atvirkštinį ryšį su distribucija – containedIn.
- Q reiškia apriboto kardinalumo savybes. Pavyzdžiui mūsų ontologijoje SafePackage konkreti klasė gali būti sujungta ne daugiau nei su nuliu CPE konkrečių objektų, kitaip sakant ji negali būti sujungta su nei vienu.

Šie pavyzdžiai įrodo, kad ontologija yra ALCIQ ekspresyvumo.

Tokio ekspresyvumo ontologijos patikrinimas (*angl. consistency check*) yra PSPACE-complete sudėtingumo[TB01], o samprotavimas – NEXPTIME-complete sudėtingumo klasės[Tob00].

Išskiriama, kad ontologijos veikimo greitį įtakoja egzistavimo operatorių naudojantys apribojimai, ontologijos dydis, nepriklausomų kelių kiekis ontologijos hierarchijoje, vardinių klasių charakteristikos ir hierarchijos panašumas į medį[KLK12].

Darbe sukurta ontologija yra sudaryta vien iš T-Box logikos, nes samprotautojai ją greičiau apdoroja nei A-Box logiką[BHJ+08].

Šiame darbe samprotautojo pagrindinė užduotis yra klasifikuoti ontologiją, todėl reikia patikrinti kaip klasifikacijos laikas priklauso nuo ontologijos dydžio. Atlikti bandymai rodo, kad didelės apimties ontologijų klasifikacija užtrunka ilgiau, tačiau net ontologijų turinčių kelias dešimtis tūkstančių klasių galima suklasifikuoti ne ilgiau nei per keletą minučių[GHT06]. Iš to galima spręsti, kad net ir labai didelių ontologijos klasifikavimas neužtruks labai ilgai.

Buvo ištestuotos ontologijos, kurios bandyme buvo klasifikuojamos ilgiausiai. Tai genų ontologijos projektas (The Gene Ontology) ir JAV nacionalinio vėžio instituto žodyno ontologija. Bandymas buvo atliktas su kompiuteriu turinčiu i7-5700HQ procesorių, 16GB darbinės atminties, Java aplinkos parametrai buvo „-Xmx5000M -Xms2000M“. Jo rezultatai yra antroje lentelėje. Ontologija buvo užkrauta naudojant Protege 5 beta 24 ir Hermit 1.3.8.413 samprotautoją.

2 lentelė. Ontologijų testavimo rezultatai

Ontologija	**The Gene Ontology**	**National Cancer Institure Thesaurus**
Klasių skaičius	44474	118167
Aksiomų skaičius	1562234	1585592
Ekspresyvumas	ALER+	SH
Samprotautojo darbo laikas (milisekundėmis)	352449	97632

Kitų atlikti bandymai su dar didesnėmis ontologijomis rodo, kad didėjant ontologijai, jos samprotavimo greitis populiariais samprotautojais skiriasi tiesiškai ar mažai skiriasi nuo tiesiško kitimo[WLL+06].

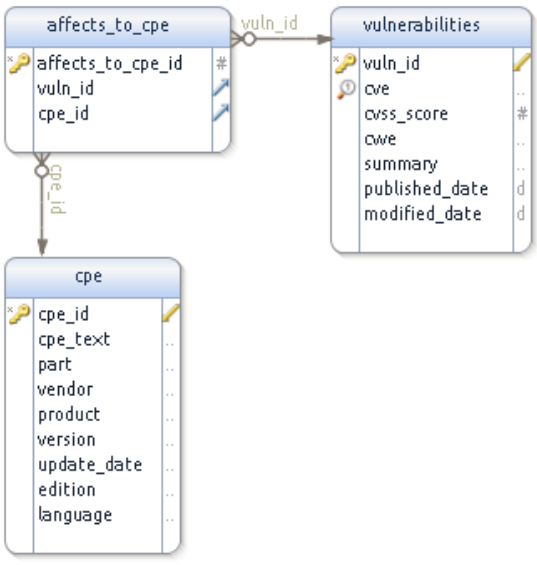
Iš aukščiau pateiktų duomenų galime spręsti, kad praplėtus darbe sukurtą ontologiją realiais duomenimis, ji veiks pakankamai greitai.

7. SAUGUMO PATIKRINIMO PROTOTIPAS

Toliau pagal ankstesniame skyriuje sudarytą metodą bus sukurtas prototipas, kuris pagal visus duomenis apie saugumo spragas iš NVD duomenų bazės ir visus duomenis apie įdiegtus paketus iš paketų valdyklės galės pasakyti ar konkreti sistema yra saugi.

NVD suteikia prieigą prie visų spragų įrašų kiekvienam norinčiam, todėl bet kuris prototipo naudotojas turės teisę naudotis naujausiais pateiktais duomenimis. Kartu su daugiau nei 70000 CVE spragų įrašų, saugomos nuorodos į daugiau nei 100000 CPE programinės įrangos versijų (vienos programos viena versija yra saugoma kaip vienas įrašas). Kadangi vis randama naujų spragų, kuriamas prototipas prieš pradėdamas spragos apdorojimą patikrina ar nėra įrašų apie tokią spragą. Tuo atveju, kai programa yra leidžiama pirmą kartą, negalima praleisti šio patikrinimo, nes duomenys apie vieną spragą gali būtų keliuose šaltiniuose. Kitaip sakant vienas paleidimas gali turėti kelis duomenis apie tą pačią spragą. Pvz.: jei spraga buvo rasta anksčiau, bet vėliau buvo gauta nauja informacija apie ją.

Programa ima visus XML spragų įrašų failus iš nurodytos direktorijos. Vėlesnėje programos versijoje gali būti įgyvendintas automatinis XML failų parsiuntimas iš NVD tinklapio.



9 pav. SQL duomenų bazės struktūra

Kiekvieną kartą norint patikrinti ar sistema yra saugi, nepatogu siųsti visus failus ir nuskaityti visą juose esančią informaciją apie spragas, nes tai pastebimai sulėtintų veikimą. Dėl šios priežasties buvo nuspręsta informaciją apie spragas nuskaityti į lokalią SQL duomenų bazę.

Kadangi įrankį naudos skirtingi naudotojai, jie gali turėti įdiegtas skirtingas SQL duomenų bazes, todėl jų patogumui, įrankio priklausomybių valdymui (*angl. dependency management*) yra naudojamas Maven[ASF16b]. Jis leidžia paprastai pakeisti JDBC bibliotekos naudojamą duomenų bazės tipą. Įprastai programoje yra naudojama atviro kodo PostgreSQL[PGD+16] duomenų bazė. Kadangi tikėtina, kad vartotojas tikrins iškart ne vienos, o kelių sistemų saugumą, tai prototipas yra kuriamas taip, kad

prototipas tikrinantis saugumą, galėtų naudoti nebūtinai lokaliai esančią spragų duomenų bazę. Tai leidžia turėti spragų duomenų bazę viename kompiuteryje ir greitai patikrinti kelių sistemų, veikiančių kituose tinklu sujungtuose kompiuteriuose, saugumą. Tai patogu, nes dėl duomenų bazės dydžio, jos importavimas nėra greitas.

Programoje naudojama duomenų bazė yra sudaryta iš trijų lentelių. Jos struktūrą galima pamatyti devintame paveiksle esančioje ERD schemoje:

1. vulnerabilities – lentelėje saugomi duomenis apie spragą ir įvairūs duomenys, kurie yra susieti tik su spraga
    1. vuld_id – automatiškai generuojamas pirminis raktas
    2. cve – spragos identifikacinis kodas NVD duomenų bazėje, taip pat naudojamas ir unikalumo užtikrinimui
    3. cvss_score – spragos poveikis pagal CVSS
    4. cwe – spragos tipas pagal CWE, jei yra atitinkantis
    5. summary – spragos aprašymas anglų kalba
    6. published_date – pirmo įvedimo į duomenų bazę data
    7. modified_date – paskutinio atnaujinimo data
2. cpe – lentelėje saugomas URI stiliaus įrašas ir šio įrašo komponentai
    1. cpe_id – automatiškai generuojamas pirminis raktas
    2. cpe_text – visas URI stiliaus įrašo tekstas, naudojamas taip pat ir unikalumo užtikrinimui
    3. part – spragos platforma (a – programinė, h – aparatinė, o – operacinės sistemos)
    4. vendor – produkto tiekėjas
    5. product – produkto pavadinimas iš CPE duomenų bazės
    6. version – versijos numeris, kaip jį nurodo pats produktas
    7. update_date – produkto atnaujinimo pavadinimas iš CPE duomenų bazės
    8. edition – produkto versija arba aparatinės įrangos architektūra
    9. language – produkto kalba, jei spraga yra tik vienoje kalboje
3. affects_to_cpe – lentelė apjungianti cpe ir vulnerabilities lentelių įrašus santykiu N:N, kiekvienas spragos įrašas gali būti išnaudojamas su daugiau nei vienu programinės įrangos vienetu ir kiekvienas programinės įrangos vienetas gali būti susijęs su daugiau nei viena spraga.
    1. affects_to_cpe_id – automatiškai generuojamas pirminis raktas
    2. vuln_id – išorinis raktas į vulnerabilities lentelės įrašus
    3. cpe_id – išorinis raktas į cpe lentelės įrašus

Cpe įrašo duomenis buvo nuspręsta išskirstyti į atskirus laukus, kad supaprastinti reikiamų duomenų paiešką ir išvedimą. Šių įrašų NVD duomenų bazėje yra daug (virš 100 tūkstančių), bet tokiam kiekiui išsaugoti nereikia labai didelės disko talpos, o paieška cpe lentelėje dėl to pagreitėja.

Kadangi kai kurios spragos gali būti susijusios su keliais CPE įrašais ir atvirkščiai vienas CPE įrašas gali būti susijęs su keliomis spragomis, šie duomenys yra laikomi atskirose lentelėse ir juos jungia atskira lentelė. Taip pat tai sumažina duomenų bazėje laikomų duomenų dydį, neleidžia jiems kartotis, išlaiko trečiąją normalinę formą.

Spragos duomenyse atskirai yra saugoma įvedimo data, kad galėtume sužinoti kada spraga atsirado, o atnaujinimo data yra reikalinga, kad teisingai atpažintume pasikeitimus duomenų bazėje. Po kurio laiko informacija apie įrašą gali būti atnaujinta, tokiu atveju reikia patikrinti ar informacija apie spragą yra naujesnė nei dabartinė įvesta į mūsų duomenų bazę ir tik tokiu atveju imtis duomenų bazės atnaujinimo. Atnaujinant duomenis yra atnaujinamos ir visos susijusios lentelės.

Ateityje prototipas bus atnaujintas taip, kad duomenis paimtų iš XML spragų failų oficialiame NVD tinklapyje. Naujoje SQL duomenų bazės lentelėje bus laikomas automatinis ID laukas, XML failo su duomenimis pavadinimas ir jo atnaujinimo data, kurią galima rasti NVD puslapyje prie kiekvieno failo eilutės. Pagal šiuos duomenis mūsų paleistas prototipas patikrins ar puslapyje nebuvo įdėtas naujesnis failas. Jei naujesnis failas buvo įdėtas, tai prototipas parsiųs šį failą ir imsis skaityti failą ir atnaujinti duomenų bazėje esančius duomenis. Priešingu atveju, jei puslapyje nėra naujesnio failo, tai prototipas tiesiog šį failą praleis. Kadangi dauguma spragų, kartu ir tos, kurios buvo aptiktos seniau nei dabartiniai metai, yra atnaujinamos, ši funkcija supaprastintų ir kartu pagreitintų prototipo turimų duomenų atnaujinimo laiką.

8. SAUGUMO PATIKRINIMAS

Sudarius duomenų bazę, pagal surinktus duomenis galima įvertinti distribucijos saugumą. Saugumas yra tikrinamas imant kiekvieną įdiegtą per paketų valdyklę paketą, paimant jo pavadinimą ir versijos numerį be specifinio distribucijos paketo versijos numerio (*angl. build number*) ir ieškant atitikmenų CPE lentelėje. Programinė įranga įdiegta ne per paketų valdyklę nėra aptinkama.

Kadangi versijos numerio formatas skiriasi priklausomai nuo distribucijos, tai programa turi žinoti kokia yra sistemos distribucija. Tai įvyksta analogiškai, kaip ir sudarant distribucijos įdiegtų paketų sąrašą. Paketo versijos numeris gali būti neatvaizduojamas iš viso, paskutinis skaičius atskirtas įvairiais skirtukais ir t.t.

Visada turi būti rastas tik vienas atitinkantis CPE įrašas. Jei rasta daugiau, tai įvyko kažkokia nenumatyta klaida. Kadangi vienas programinės įrangos elementas gali turėti daugiau nei vieną spragą, norint sutaupyti disko vietą naudojamą duomenų bazės ir išlaikyti duomenų normalinę formą, keli NVD spragų įrašai gali rodyti į vieną tą patį CPE įrašą. Jei įrašų nebuvo rasta visai, tai arba šis paketas distribucijoje yra kitaip pavadintas arba jis yra saugus.

Radus vienintelį atitinkantį įrašą, yra išvedamas vienas arba daugiau atitinkančių spragų įrašų. Jei joks įrašas nebuvo rastas, duomenų bazės formavimo arba atnaujinimo metu įvyko klaida. Pvz.: buvo atnaujinta spragos informacija, kad su kažkokia tam tikra versija spraga neįvyksta, buvo atnaujinta vulnerabilities ir cpe_to_affects lentelės, bet ne cpe. Iš šios lentelės reikia ištrinti visus įrašus, į kuriuos nėra nuorodos iš cpe_to_affects lentelės. Rasto įrašo CVSS reikšmė yra pridedama prie bendros šios sistemos CVSS reikšmės.

Patikrinus visus įrašus, prototipas pateikia šią informaciją:

- įdiegtų paketų skaičius;
- rastų paketų su spragomis skaičius;
- rastų spragų skaičius;
- vidutinė CVSS vertė.

Pagal šią informaciją, sistemos naudotojas turi galimybę įvertinti bendrą sistemos saugumą. Jei rasta daug spragų arba vidutinė CVSS reikšmė yra didelė, tai vartotojui yra rekomenduojama atnaujinti sistemą. Jei sistemoje yra mažai spragų, tačiau jos turi aukštą CVSS reikšmę, tai sistemos vartotojui yra rekomenduojama atnaujinti bent tuos paketus.

Toliau galima prototipą atnaujinti taip, kad radus spragų, prototipas pats pasiūlytų atnaujinti visą sistemą arba bent konkrečius paketus. Šiam tikslui prototipas taip pat galėtų

pasitelkti integruotos paketų valdyklės funkcijas. Kadangi šiuo atveju jau keičiama bendra sistemos būsena, tai paketų valdykle reiktų kviesti tam turinčio pakankamas privilegijas sistemos vartotojo prisijungimu. Kadangi kitoms funkcijoms nereikia aukštesnių nei paprasto vartotojo privilegijų, tai prieš tai, tikrinant sistemos saugumą, prototipas yra leidžiamas paprasto vartotojo teisėmis. Norint gauti aukštesnes privilegijas, prototipas galėtų pasinaudoti universaliu Linux įrankiu „sudo“[Mil16]. Atsargesnis veikimo variantas, būtų atnaujinti ne visą sistemą, o atskirus jos paketus. Abiems variantams gali tekti pakeisti įrankio naudojamą paketų tvarkymo įrankį. Pvz. Debian šeimos distribucijose prototipas paketų aptikimui naudoja labai bendrą žemo lygio įrankį „dpkg“. Toks sprendimas buvo padarytas su tikslu, kad prototipas tikrai veiktų visose Debian šeimos distribucijose, tačiau jis neleidžia paprastai atnaujinti paketų, nes jis nepalaiko standartinių repozitorijų, o paketus gali diegti tik tiesiogiai iš failų. Šiuo ir panašiais atvejais reikės naudoti aukštesnio lygio paketų tvarkymo įrankį (pvz.: „apt-get“). Analogiška problema yra ir su RPM paketus naudojančiomis distribucijomis. RPM komanda negali parsiųsti naujausių paketų versijų, todėl patogiausia būtų pasinaudoti aukštesnio lygio paketų tvarkymo programomis. Tuo atveju, kai yra naudojama Arch Linux distribucija arba ja paremtos, ir toliau galima naudoti pacman paketų valdyklę, nes ji yra pagrindinė distribucijos paketų valdyklė ir todėl sugeba patikrinti ir parsiųsti naujas paketų versijas.

9. PROTOTIPO VEIKIMO TESTAVIMAS

Darbo metu sukurtą prototipą reikėjo ištestuoti sąlygomis kiek įmanoma artimomis realioms. Testavimui buvo parinktos populiarios distribucijos naudojančios skirtingas paketų valdykles.

Prieš pradedant prototipo testavimą, buvo atsiųsti visi duomenys, kurie testavimo metu buvo NVD duomenų bazėje, apie saugumo spragas ir įkelti į bendrą SQL duomenų bazę.

Rezultate buvo gauta duomenų bazė, kurią sudarė tokie įrašai:

1. 74506 saugumo spragų įrašai;
2. 179771 CPE įrašai apie programinės įrangos elementus;
3. 1919596 įrašai surišantys spragas su CPE įrašais.

Testavimui buvo parinktos testavimo metu naujausios versijos distribucijų, kurios buvo aptartos ankstesniuose darbo skyriuose. Šios distribucijos apima visas aptartas paketų valdyklių šeimas. Kaip jau minėta anksčiau, pakanka turėti vieną patikrinimą žemiausio lygio paketų programai, kuri sutampa visose išvestinėse distribucijose. Arch Linux nėra nurodyta konkreti versija, nes tai pastoviai atnaujinama distribucija (*angl. rolling release*). Įdiegtų distribucijų sąrašas su jų versijomis:

- Debian jessie 8.2;
- Ubuntu Wily Werewolf 15.10;
- CentOS 7.2.1511;
- Mageia 5;
- openSUSE Leap 42.1;
- Fedora 23;
- Arch Linux atnaujinta.

Įdiegtų paketų sąrašas:

- Apache HTTP server[ASF16a];
- PHP language[PG16];
- Tomcat[ASF16c];
- OpenSSH[Ope16];
- MariaDB[Mar16];
- MongoDB[Mon16].

Prototipo modulių testavimo metu iškilo poreikis turėti minimalią izoliuotą sistemą, kurioje galima būtų išbandyti prototipą. Ši virtuali sistema turėjo atitikti šiuos poreikius:

- galimybė įsidiegti naujausias populiarių distribucijų versijas;
- automatizuotų komandinės eilutės komandų paleidimo funkcija;
- galimybė išsaugoti virtualios sistemos atvaizdą tam tikrais žingsniais ir paleisti sistemą nuo to žingsnio;
- galimybė virtualioje sistemoje naudoti bendrus failus iš realios sistemos;
- galimybė virtualioje sistemoje pasiekti realios sistemos tinklą.

Visus šiuos kriterijus atitiko Docker[Doc16] konteinerių platforma. Joje kiekvienai atskirai distribucijai buvo sukurtas atskiras Docker failas, kuriame buvo nurodytos reikalingos komandos, kurios parsiųsdavo reikiamą distribucijos atvaizdą, įdiegdavo numatytus paketus, prijungdavo numatytą realios sistemos direktoriją, kurioje buvo prototipo failai, ir paleisdavo prototipą. Ši platforma pagreitino pradinį prototipo testavimą, bet netiko galutiniam testavimui.

Galutiniam testavimui reikėjo visiškai izoliuotos sistemos, kurioje būtų sudiegti visi realiai reikalingi paketai, o Docker, dėl sistemos resursų ir bendro veikimo greičio, įdiegia tik minimaliai reikalingus sistemos veikimui paketus, o kitus naudoja iš realios sistemos. Tokiu būdu pvz. sistemos branduolys (*angl. kernel*) yra naudojamas iš realios sistemos. Vadinasi, negalėtume ištestuoti tokių paketų saugumo, todėl buvo pasirinkta įprastas virtualizacijos įrankis – VirtualBox[Ora16a]. Jame buvo sudiegtos visos reikalingos distribucijų versijos ir reikalingi paketai.

Galutinio testavimo metu gauti rezultatai yra trečioje lentelėje.

3 Lentelė.Testavimo rezultatai

Distribucija	Viso paketų	Rasta paketų su spragomis	Viso rasta spragų	Vidutinė CVSS vertė
Debian	1784	12	36	6.2055555978
Ubuntu	1915	9	20	6.9450000285
CentOS	1375	51	164	5.4030488457
Mageia	901	13	36	6.6305555736
openSUSE	1354	10	23	5.9166667328
Fedora	1499	17	30	5.4133333963
Arch	702	8	18	6.9166667328

Kaip matome iš lentelės, standartinėje Fedora instaliacijoje buvo įdiegta daugiausiai paketų, o Arch – mažiausiai. Vadinasi, jei sistema turi užimti mažiausiai vietos diske, geriausiai tinka ši Arch Linux distribucija.

Daugiausia paketų su spragomis ir daugiausia spragų rasta CentOS distribucijoje, todėl galima teigti, kad stabili šios distribucijos versija yra mažiau saugi už kitas distribucijas.

Mažiausiai paketų su spragomis ir mažiausiai spragų yra rasta Arch distribucijoje, tačiau šioje distribucijoje yra didelė vidutinė CVSS reikšmė, tai reiškia, kad spragos šioje distribucijoje yra santykinai pavojingos.

Kita distribucija, kurioje yra panašus mažas spragų kiekis yra – Ubuntu. Šioje distribucijoje yra vienu daugiau paketų su spragomis ir dviem bendrai spragomis daugiau. Taip pat ir CVSS vidutinė spragų vertė yra didesnė.

Svarbiausias testavimo rezultatas yra teisingas prototipo veikimas. Prototipo testavimas parodė:

- prototipas teisingai nuskaito visus NVD duomenų bazėje esančius įrašus;
- sukuria kiekvienam įrašui atitinkantį įrašą lokalioje SQL duomenų bazėje;
- randa visus populiariausiose Linux distribucijose įdiegtus paketus;
- kiekvienam paketui prototipas randa atitinkamą CPE įrašą;
- kiekvienam tinkančiam CPE įrašui prototipas randa atitinkantį vieną spragos įrašą ir jo CVSS vertę;
- prototipas paskaičiuoja vidutinę visų rastų spragų CVSS vertę.

Iš testavimo rezultatų galime teigti, kad darbo metu sukurtas prototipas atitinka jam iškeltus reikalavimus.

IŠVADOS IR REZULTATAI

Šiame darbe buvo rasta strategija universaliam ir patikimam žinomų saugumo spragų radimui programinėje įrangoje ir buvo sukurtas metodas, aprašantis kaip galima patikrinti Linux sistemos saugumą pagal NVD saugumo spragų duomenų bazę. Pagal metodą buvo sukurtas jį įgyvendinantis prototipas.

Darbo rezultatai:

1. Sukurta saugumo spragų srities ontologija.
2. Sukurtas metodas kaip nustatyti sistemos saugumą pasinaudojant ontologija.
3. Sukurtas ir ištestuotas prototipas sistemos saugumo nustatymui.

Darbo išvados:

1. Ekspresyvi ontologijų kalba OWL DL gerai tinka žinių aprašymui ir leidžia samprotauti pasinaudojant deskriptyvia logika.
2. Pagal NVD laisvai pateikiamus duomenis galima suformuoti duomenų bazę turinčią visus reikalingus duomenis apie saugumo spragas.
3. Iš atliktų eksperimentų galima teigti, kad praplėtus darbe sukurtą ontologiją realiais duomenimis, ji veiks pakankamai greitai.

ŠALTINIAI

[AL16] Arch Linux. [žiūrėta 2016-01-10]. Prieiga per internetą: <https://www.archlinux.org/>.

[ALR+04] A.Avizienis, J.-C.Laprie, B.Randell, C.Landwehr. Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing, IEEE, 1(1), 2004, pp.11-33. [žiūrėta 2015-01-19]. Prieiga per internetą: <http://dl.acm.org/citation.cfm?id=1026488.1026492>.

[ASF16a] Apache Software Foundation. Apache. [žiūrėta 2016-01-10]. Prieiga per internetą: <https://httpd.apache.org/>.

[ASF16b] Apache Software Foundation. Maven. [žiūrėta 2016-01-10]. Prieiga per internetą: <https://maven.apache.org/>.

[ASF16c] Apache Software Foundation. Tomcat. [žiūrėta 2016-01-10]. Prieiga per internetą: <http://tomcat.apache.org/>.

[BHJ+08] J.Bock, P.Haase, Q.Ji, R.Volz. Benchmarking OWL reasoners. Advancing Reasoning on the Web: Scalability and Commonsense, 2008. [žiūrėta 2016-01-10]. Prieiga per internetą: <http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-350/>.

[BHS04] F. Baader, I.Horrocks, U.Sattler. Handbook on Ontologies. Springer, 2004, p.21-43. [žiūrėta 2015-06-15]. Prieiga per internetą: <http://www.springer.com/us/book/9783540709992>.

[Bry86] R.E.Bryant. Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers, IEEE, C-35(8) 1986 , pp.677–691. [žiūrėta 2015-06-15]. Prieiga per internetą: <http://dl.acm.org/citation.cfm?id=6432.6433>.

[BS85] R.J.Brachman, J.G.Schmolze. An Overview of the KL-ONE Knowledge Representation System. Cognitive Science, 9(2), 1985, pp.171–216. [žiūrėta 2015-06-10]. Prieiga per internetą: <http://doi.wiley.com/10.1207/s15516709cog0902_1>.

[Can16] Canonical Ltd. Ubuntu. [žiūrėta 2016-01-10]. Prieiga per internetą: <http://www.ubuntu.com/>.

[CAP+15] Common Attack Pattern Enumeration and Classification (CAPEC) . [žiūrėta 2015-06-14]. Prieiga per internetą: <https://capec.mitre.org/>.

[CP16] CentOS Project. Centos. [žiūrėta 2016-01-10]. Prieiga per internetą: <https://www.centos.org/>.

[CVS+15] Common Vulnerability Scoring System (CVSS-SIG). [žiūrėta 2015-06-14]. Prieiga

per internetą: <https://www.first.org/cvss>.

[CWE15] Common Weakness Enumeration. [žiūrėta 2015-06-14]. Prieiga per internetą: <https://cwe.mitre.org/>.

[Deb16] Debian. [žiūrėta 2016-01-10]. Prieiga per internetą: <https://www.debian.org/index.lt.html>.

[Doc16] Docker Inc. Docker. [žiūrėta 2016-01-10]. Prieiga per internetą: <https://www.docker.com/>.

[ED16] Exploit Database. Exploit Database. [žiūrėta 2016-01-10]. Prieiga per internetą: <https://www.exploit-db.com/>.

[Fen11] D.Fensel. Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce. Springer, 2001. [žiūrėta 2015-06-14]. Prieiga per internetą: <http://www.springer.com/us/book/9783540003021>.

[Gal11] L.Gallon. Vulnerability discrimination using CVSS framework. 2011 4th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2011 - Proceedings. IEEE, 2011. [žiūrėta 2015-05-17]. Prieiga per internetą: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5720656>.

[Gar10] R.Garcia. A Semantic Web approach to Digital Rights Management. Communication. VDM Verlag, Saarbrücken, Germany, 2010. [žiūrėta 2015-05-17]. Prieiga per internetą: <http://rhizomik.net/html/~roberto/thesis/>.

[GH16] GitHub, Inc. GitHub. [žiūrėta 2016-01-10]. Prieiga per internetą: <https://github.com/>.

[GHM+08] B.C.Grau, I.Horrocks, B.Motik, B.Parsia, P.Patel-Schneider, U.Sattler. OWL 2: The next step for OWL. Web Semantics: Science, Services and Agents on the World Wide Web, Elsevier Science Publishers B. V. Amsterdam, 6(4), 2008, pp.309-22. [žiūrėta 2015-05-17]. Prieiga per internetą: <http://www.websemanticsjournal.org/index.php/ps/article/view/156>.

[GHT06] T.Gardiner, I.Horrocks, D.Tsarkov. Automated Benchmarking of Description Logic Reasoners. Proceedings of the 2006 International Workshop on Description Logics, CEUR Workshop Proceedings 189, Windermere, 2006.

[HPH03] I.Horrocks, P.Patel-Schneider, F.Van Harmelen. From SHIQ and RDF to OWL: The Making of a Web Ontology Language. Journal of Web Semantics, Elsevier, 1(1), 2003, pp.7-26. [žiūrėta 2015-05-17]. Prieiga per internetą: <http://www.cs.vu.nl/~frankh/abstracts/JWS03.html>.

[JWX+08] A.Ju, A.Wang, M.Xia, F.Zhang. Metrics for Information Security

Vulnerabilities. Journal of Applied Globable Research, 1(1), 2008, pp.48-58. [žiūrėta 2015-05-17]. Prieiga per internetą: <http://mason.gmu.edu/~fzhang4/paper/wang-iic07.pdf>.

[KLK12] Y.B.Kang, Y.F.Li, S.Krishnaswamy. Predicting Reasoning Performance Using Ontology Metrics. ISWC'12 Proceedings of the 11th international conference on The Semantic Web, 1(1), 2012, pp.198-214. [žiūrėta 2015-05-17]. Prieiga per internetą: <http://link.springer.com/chapter/10.1007%2F978-3-642-35176-1_13#page-1>.

[Knu15] D.E.Knuth. Lectures. Stanford Center for Professional Development, Stanford. [žiūrėta 2015-06-15]. Prieiga per internetą: <http://scpd.stanford.edu/free-stuff/engineering-archives/donald-e-knuth-lectures>.

[Mag16] Mageia. [žiūrėta 2016-01-10]. Prieiga per internetą: <https://www.mageia.org/en/>.

[Mar16] MariaDB Foundation. MariaDB. [žiūrėta 2016-01-10]. Prieiga per internetą: <https://mariadb.org/>.

[Mil16] T.C.Miller. sudo. [žiūrėta 2016-01-10]. Prieiga per internetą: <https://www.sudo.ws/>.

[MIT15] The MITRE Corporation. [žiūrėta 2015-06-14]. Prieiga per internetą: <http://www.mitre.org/>.

[Mon16] MongoDB Inc. MongoDB. [žiūrėta 2016-01-10]. Prieiga per internetą: <https://www.mongodb.org/>.

[NB03] D.Nardi, R.J.Brachman. An introduction to description logics. Cambridge University Press, New York, 2003. [žiūrėta 2015-06-14]. Prieiga per internetą: <http://dl.acm.org/citation.cfm?id=885746.885748>.

[NG84] J.D.Novak, D.B.Gowin. Learning How to Learn. Cambridge University Press, New York, 1984. [žiūrėta 2015-06-15]. Prieiga per internetą: <http://www.cambridge.org/us/academic/subjects/psychology/developmental-psychology/learning-how-learn>.

[NRC91] National Research Council. Computers at risk: safe computing in the information age. National Academy Press, Washington, 1991. [žiūrėta 2015-06-16]. Prieiga per internetą: <http://dl.acm.org/citation.cfm?id=102690>.

[NVD15] NVD. [žiūrėta 2015-06-15]. Prieiga per internetą: <https://nvd.nist.gov/>.

[OPE15] openSUSE.org. [žiūrėta 2015-06-15]. Prieiga per internetą: <https://www.opensuse.org/lt/>.

[Ope16] OpenBSD. OpenSSH. [žiūrėta 2016-01-10]. Prieiga per internetą: <http://www.openssh.com/>.

[Ora16a] Oracle Corporation. VirtualBox. [žiūrėta 2016-01-10]. Prieiga per internetą: <https://www.virtualbox.org/>.

[OSV15] OSVDB: Open Sourced Vulnerability Database. [žiūrėta 2015-06-14]. Prieiga per internetą: <http://osvdb.org/>.

[PCI15] Official Source of PCI DSS Data Security Standards Documents and Payment Card Compliance Guidelines. [žiūrėta 2015-06-14]. Prieiga per internetą: <https://www.pcisecuritystandards.org/security_standards/>.

[PG16] PHP Group. PHP. [žiūrėta 2016-01-10]. Prieiga per internetą: <https://www.php.net/>.

[PGD+16] PostgreSQL Global Development Group. PostgreSQL. [žiūrėta 2016-01-10]. Prieiga per internetą: <http://www.postgresql.org/>.

[RBS15] Risk Based Security. [žiūrėta 2015-06-14]. Prieiga per internetą: <https://www.riskbasedsecurity.com/>.

[RH15] Red Hat. [žiūrėta 2015-06-15]. Prieiga per internetą: <http://www.redhat.com/en>.

[RH16a] Red Hat Inc. Fedora. [žiūrėta 2016-01-10]. Prieiga per internetą: <https://fedoraproject.org/>.

[RH16b] Red Hat Inc. Red Hat Enterprise Linux. [žiūrėta 2016-01-10]. Prieiga per internetą: <http://www.redhat.com/en>.

[SBF98] R.Studer, V. R.Benjamins, D.Fensel. Knowledge engineering: Principles and methods. Data & Knowledge Engineering, Elsevier Science Publishers B. V. Amsterdam, 25(1-2), 1998, pp.161–197. [žiūrėta 2015-06-15]. Prieiga per internetą: <http://dl.acm.org/citation.cfm?id=290547.290553>.

[Sus16] SUSE Linux Enterprise. [žiūrėta 2016-01-10]. Prieiga per internetą: <https://www.suse.com/>.

[TB01] S.Tobies, F.Baader. Complexity results and practical algorithms for logics in knowledge representation. RWTH Achen, Achen, 2001. [žiūrėta 2015-05-17]. Prieiga per internetą: <http://publications.rwth-aachen.de/record/52234>.

[Tob00] S.Tobies. The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. Journal of Artificial Intelligence Research, AI Access Foundation, 12(1), 2000, pp.199-217. [žiūrėta 2015-05-17]. Prieiga per internetą: <http://dl.acm.org/citation.cfm?id=1622255>.

[TW07] R.Telang, S.Wattal. An Empirical Analysis of the Impact of Software Vulnerability

Announcements on Firm Stock Price. IEEE Transactions on Software Engineering, IEEE Press Piscataway, 33(8), 2007, pp.544-557. [žiūrėta 2015-05-17]. Prieiga per internetą: <http://dl.acm.org/citation.cfm?id=1437850>.

[UI15] Unsigned Integer Limited. DistroWatch.com. [žiūrėta 2015-06-14]. Prieiga per internetą: <http://distrowatch.com/>.

[WLL+06] T.Weithöner, T.Liebig, M.Luther, S.Böhm. What's Wrong with OWL Benchmarks? Proceedings of the Second International Workshop on Scalable Semantic Web Knowledge Base Systems, 2006, pp.101-114. [žiūrėta 2015-05-17]. Prieiga per internetą: <https://www.uni-ulm.de/fileadmin/.../weithoener-et-al-ssws06.pdf>.

[WMI15] Windows Management Instrumentation. [žiūrėta 2015-06-14]. Prieiga per internetą: <https://msdn.microsoft.com/en-us/library/aa394582(v=vs.85).aspx>.

[XF16] IBM Security. X-FORCE. [žiūrėta 2016-01-10]. Prieiga per internetą: <http://www-03.ibm.com/security/xforce/>.

[XFE16] IBM Security. X-FORCE Exchange. [žiūrėta 2016-01-10]. Prieiga per internetą: <https://exchange.xforce.ibmcloud.com/>.

[ZC09] M.A.Zhivich, R.K.Cunningham. The Real Cost of Software Errors. IEEE Security & Privacy, IEEE, 7(2), 2009, pp.87-90. [žiūrėta 2015-05-17]. Prieiga per internetą: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4812166>.

SĄVOKOS IR SANTRUMPOS

CPE – Common Platform Enumeration, būdas unikaliai identifikuoti programinės įrangos vienetą su jo versijos numeriu.

CVE – Common Vulnerabilities and Exposures, NVD organizacijos saugumo spragų duomenų bazė.

CVSS – Common Vulnerability Scoring System, sistema leidžianti įvertinti spragos veiksmingumą skaitine verte.

ERD – Entity Relationship Diagram, DB objektų diagrama.

NIST – National Institute of Standards and Technology, JAV vyriausybės technologijų ir standartų institutas.

NVD – National Vulnerability Database, JAV vyriausybės IT saugumo repozitorija.

OSVDB – Open Sourced Vulnerability Database, atviro kodo saugumo spragų duomenų bazė.