VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS INSTITUTAS
KOMPIUTERINIO IR DUOMENŲ MODELIAVIMO KATEDRA

Bakalauro darbas

Saugus pažeidžiamumų skaitytuvas sistemų auditavimui

Atliko:
Jonas Gavėnavičius parašas

Vadovas:
Lektorius Virgilijus Krinickij

Vilnius
2020

Turinys

Sutartinis terminų žodynas

- FTP - *File trasnfer protocol* Failų perkelimo protokolas, protokolas leidžiantis perkelti duomenis iš vienos sitemos į kitą[19].
- MITM - *Man in the middle* - Žmogaus viduryje ataka, tai kibernetinės atakos tipas, kai puolėjas perima bendravima tarp serverio ir kliento[6].
- Buffer overflow - tai viena iš potencialių rizikų, kai dėl per didelio kiekio duomenų, informacija yra perrašoma gretimuose atminties blokuose[7].
- SSH - *Secure Shell*, tai tinklo protokolas kuris leidžia vartotojui saugiai pasiekti sistemą per nesaugu tinklą [27].
- Framework - Tai karkasas, į kurį įeina kelios ar daugiau programinės įrangos ar kitų sistemų.
- NASL - *Nessus Attack Scripting Language*, tai paprastą kalbą, skirta aprašyti atskiras grėsmes ir galimas atakas.
- IDS - *Intrusion detection system*, tai sistema, skirta aktyviam saugumui, ji gali aptikti išpuoli realiu laiku[22].
- IPS - *Intrusion prevention system*, tai *IDS* papildymas, kuris gali aptikti įsilaužimą, ir ji sustabdyti[22].
- HTTP - *HyperText Transfer Protocol* tai informacijos priemimo perdavimo protokolas[9].
- Daemon - Tai programa kuri veikia fone ir nėra valdoma vartotojo, bet laukia specifinio įvykio ar salygos suveikti.
- Slaptas įejimas - tai kodo dalis kuri leidžia atakuotojuj į sistemą patekti nepastebėtam.
- Kenkėjiška programinė įranga - tai bendras pavadinimas tokiai programinei įrangai, kuri yra kenkėjiška ir patenka į sistema be vartotojo leidimo[11].
- SQL injekcija - tai vienas iš kibernetinės atakos tipų kai vartotojo įvestis internetinėje svėtainėje yra sumanipuliuojama taip, kad ji padarytu daugiau negu buvo planuota, paveikiant duomenų bazę ir joje įgyvendinant užklausą[14].
- Fišingas - būdas skirtas pavogti privačia informacija apsimetant tam tikru tiekėju.

Santrauka

Šiais laikais kai pasaulis tampa vis labiau ir labiau skaitmenizuotas, iškyla opi problema, tai yra kibernetine sauga. Vis daugiau pavyždžių kasdieniame gyvenime matome, kai įsibraunama į sistemas, kurias išnaudoja dėl finansinių ar kitų priežasčiu. Taip dėl to nukenčia tiekėjai prarasdami savo reputacija ir kapitalą, ar tų sistemų vartotojai, kai jų privati informacija būna pavogiama ir paviešinama. Niekas nėra saugus nuo tokių situacijų. Todėl valstybės, įmonės ar korporacijos vis daugiau investuoja į kibernetinę saugą. Kibernetinė sauga tampa vis dažnesnė diskusija visuomenėja, vis daugiau dėmėsio ir resursų skiriama butent kibernetinei saugai, taip stengemasi užkirsti kelią situacijoms, kurios atneštų žalą vartotojams, įmonėms ar net šalims. Kuriami įrankiai kurie padėtu apsisaugoti nuo tokių situacijų. Šie įrankiai analizuoja sistemas ir randa jų spragas, taip pat pritaikomos įvairios technologijos ar metodai kurie perspėja apie galima ar vykstančią ataką prieš sistemą, stengiamasi rasti sistemoje spragas ir užlopyti jas kol jų nerado asmenys kurie išnaudotų jas.

Summary

Secure Vulnerability Scanner for System Auditing

Nowadays, as the world becomes more and more digital, there is a pressing problem, which is cybersecurity. In more and more examples of everyday life we see when intruding into the systems exploits for financial or other reasons. This causes suppliers to lose their reputation and capital or their systems users when their private information is stolen and made public. No one is safe from such situations. As a result, governments, companies or corporations are investing more and more to cybersecurity. Cyber security is becoming an increasingly public debate, more and more the focus and resources are focused on cybersecurity, trying to prevent situations that would harm consumers, businesses or even countries. Tools are developing to help prevent such situations..These tools analyze systems and identify gaps, and adapt various technologies or methods which warn of a possible or ongoing attack on the system, attempts to find loopholes in the system and patch them until they are found by people who exploit them.

Įvadas

Šiame darbe kuriamas saugus įrankis, kuris neišduotų savo buvimo vietos ir vartotojas pasinaudojas juo galėtų gauti išsamią informaciją apie atitinkamą internetinę svetainę ir joje egzistuojančias spragas bei pažeidžiamumus ar modifikuotus failus. Taip pat nerizikuodamas užkrėsti savo sistemą skenavimo metu. **Darbo tikslas** – Sukurti saugų pažeidžiamumų skenavimo įrankį, kuris veiktu iš tam tikros slepiamos vietos, saugiai skenuotų internetinę svetainę bei jos failus ir pateiktų informaciją apie skenavimo rezultatus. Keliami **darbo uždaviniai** yra tokie:

1. Apžvelgti egzistuojančius panašius skevanimo įrankius;
2. Išanalizuoti dažniausiai pasitaikančius internetinių svetainių paže idžiamumus;
3. Išanalizuoti dažniausiai pasitaikančius internetinių svetainių skenavimo metodus;
4. Pateikti gerąsias praktikas kurios padėtų užtikrinti didesnį saugumą sistemai;
5. Sukurti įranki, kuris būtu saugiai skenuoti internetinę svetainę.

Kibernetinis saugumas yra vienas svarbiausiu aspektų šių dienų technologijose. Kiekvienas asmuo, turintis išmanujį įrenginį yra vienaip ar kitaip priklausomas nuo kibernetinio saugumo. Vienas pažeidžiamumas sistemoje gali lemti visos žmogaus asmeninės informacijos esančios toje sistemoje ar tame įrenginyje pasisavinimą. Internetinės svėtaines taip pat neatsiejamos kasdieniame gyvenime, nes jos naudojamos kaip varotojo sąsajos, internetiniams apsipirkimams, banko sąskaitų valdymui, socialiniams tinklams, verslo sprendimams. Ši problema yra ypač opi įmonėms, kurių paslaugomis naudojasi dideli kiekiai žmonių, nes dėl vieno pažeidžiamo, visa jų vartotojų privati informacija gali būti pavogta ir panaudota kitiems tikslams. Tai gali sukelti didžiulius nuostolius įmonėms ir jos gali visam laikui prarasti reputaciją, dėl kurios vartotojai rinkosi juos.

Kadangi kiekvieną sistemą kuria žmogus, ir į kiekvieną sistemą įeina žmogiškasis faktorius, dėl kurio yra tikimybė, kad ši sistema turės pažeidžiamumų, kurias gali išnaudoti puolėjas siekiantis tam tikros naudos. Dėl šios priežasties pažeidžiamumų skenavimo įrankis būtų ypač naudingas bet kokiai sistemai. Aptikus pažeidžiamumą, galima įtarti, kad tą pažeidžiamumą gali aptikti ir nuostolio siekiantys asmenys, arba jie jau jį aptiko, ir tam tikri failai sistemoje buvo įdėti arba modifikuoti modifikuoti įdedant tam tikrą kodą, kuris leistų atakuojančiui asmeniui patekti į sistemą nepastebėtam arba sutrikdyti sistemos veikimą. Dėl šių priežasčių saugus ir automatizuotas sistemų pažeidžiamumų skenavimo įrankis yra ypač aktualus. Tačiau šio įrankio kūrimą apsunkina šios priežastys:

- Įrankio kūrimas reikalauja didelio bagažo žinių;
- Daugumos sistemų, kurios turi pažeidžiamas vietas, išeities kodas nėra atviras, todėl kai kurie pažeidžiamumai negali būti aptiktos automatizuotu įrankiu. Taip pat kiekvienai technologijai, kurią naudoja sistema, reikia atlikti atskirą analizę, kas taip pat sumažina tokio įrankio efektyvumą ir ženkliai padidiną kurimo sudetingumą;
- Sistemoje gali būti tiek daug skirtingų potencialių pažeidžiamų, jog jų aptikimo automatizavimas tampa problematiškas;
- Dėl didelio skaičiaus vietų, kur pažeidžiamumai gali apsireikšti, bei dėl labai didelio galimų spagų skaičiaus, laiko kaštai ženkliai išauga.

Šias problemas siūloma spręsti bandant identifikuoti pačias opiausias vietas, kuriose dažniausiai pasitaiko pažeidžiamumai ir kiti neatitikimai, tuomet aprebti identifikuotas opiausias vietas ir naudoti skirtingas analizes bei skenavimų metodus, su kuriais yra didžiausias šansas rasti šiose vietose pažeidžiamumus. Tokiu būdu yra sumažinama tikimybė surasti visus pažeidžiamumus sistemoje, bet padidinama tikimybė aptikti daugiausiai pažeidžiamumų, taip pat sumažint laiko bei žinių kaštus.

Šio įrankio kūrimo metu, siekiant sukurti saugų pažeidžiamumų aptikimo įrankį yra siekiama pagerinti sistemos saugumą, bet neužtikrinti jo, taip pat užtikrinant ir pačio įrankio sistemos saugumą. Įrankis bus skirtas tik internetinėms svetainėms ir nebus stengiamasi jo pritaikyti ir kitoms sistemoms.

Pirmajame skyriuje yra analizuojami jau egzistuojančios sistemos ar įrankiai, kurie yra butent skirti pažeidžiamumų skenavimui tam tikrose vietose, ar visoje sistemoje. Antrajame skyriuje aptariami skirtingi metodai sistemų auditavimui, paaiškinama kam tie metodai skirti, kokiius pažeidžiamumus jie padeda aptikti. Trečiajame skyriuje yra pateikiamos gerosios praktikos, kokio tipo programinė įranga yra saugesnė, kas padeda sumažinti riziką turėti pažeidžiamumą sistemoje. Ketvirtajame skyriuje aprašomas įrankio kurimo procesas, architektura, naudotos technologijos, nepasisekę tikslai, įgyvendinti tikslai, ir pasiektas rezultatas.

1. Susijusių darbų analizė

Susijusių darbų analizėje yra analizuojami projektai, darbai, įrankiai, kurie vienaip ar kitaip skenuoja sistemas, ieško jose spragų. Analizės metu bandoma paaiškinti, kam šie darbai yra skirti, kokios yra jų stiprybės, kokio tipo pažeidžiamumus galima aptikti su šiais projektais, darbais ar įrankiais.

1.1. Nessus skaitytuvas

„Nessus“ įrankis yra tinklo pažeidžiamumų skaitytuvas, kuris naudoja bendrąją pažeidžiamumų architektūrą, kad lengvai susietų suderinamus kibernetinio saugumo įrankius. „Nessus“ naudoja *NASL*[21].

„Nessus“ turi modulinę architektūrą, susidedančią iš serverio *daemon* atliekančio nuskaitymą, ir nuotolinų kliento kuris yra valdomas administratoriaus. Administratoriai gali įtraukti *NASL* visų įtariamų pažeidžiamumų aprašus, kad sukurtų tinkintus nuskaitymus. Reikšmingos „Nessus“ galimybės:

- Suderinamumas su bet kokio dydžio kompiuteriais ir serveriais.
- Apsaugos spragų aptikimas vietiniuose ar nuotoliniuose kompiuteriuose.
- Trūkstamų sistemų ir programinės įrangos saugumo atnaujinimų aptikimas.
- Imituoti išpuoliai, skirti nustatyti pažeidžiamumą.
- Saugumo testų atlikimas uždaroje aplinkoje.
- Suplanuotas saugumo auditas.

„Nessus“ serverį šiuo metu galima naudoti su dauguma „Linux“ operacinių sistemų. Klientas yra prieinamas „Linux“ arba „Windows“ operacinėms sistemoms.

1.2. OpenVAS skaitytuvas

1.2.1. Įrankio aprašymas

„OpenVAS“ yra visa apimantis pažeidžiamumų skaitytuvas. Jo galimybės apima įvairių aukšto ir žemo lygio interneto ir pramoninių protokolų skanavimą, našumo derinimą didelės apimties nuskaitymams ir galingą vidinę programavimo kalbą, kuri leidžia įgyvendinti didelio skaičiaus pažeidžiamumų testus.[2].

1.2.2. Irankio ištakos

2006 m. Buvo sukurtos kelios „Nessus“ atviro kodo atšakos, kaip reakcija į "Nessus" įrankio komercilizavimą nebepalaikant atviro kodo. Iš šių šakų tik viena toliau rodė aktyvumą: „OpenVAS“, atviro kodo pažeidžiamumų skenavimo sistema. „OpenVAS“ buvo įregistruotas kaip „Software in the Public Interest, Inc.“ projektas, skirtas valdyti ir apsaugoti domeną „openvas.org“.

Dėl šios priežasties abu įrankiai yra panašus. Didžiausias tarp jų skirtumas yra tas, kad „Nessus“ įrankis yra komercializuotas, priešingai negu „OpenVAS“.

1.3. Nmap

„Nmap“ („Network Mapper“) yra nemokamas ir atvirojo kodo įrankis skirtas tinklo skanavimui. Daugelis sistemų ir tinklo administratorių mano, kad šis įrankis naudingas atliekant tokias užduotis kaip tinklo inventorizavimas ir pagrindinio kompiuterio ar paslaugos veikimo stebėjimas. „Nmap“ naudoja neapdorotus IP paketus, kas taip pat padeda įrankiui būti daug efektyviasniam. Įrankis buvo sukurtas greitai nuskaityti didelius tinklus, tačiau puikiai veikia su atskirais kompiuteriais ar serveriais. „Nmap“ veikia visose pagrindinėse kompiuterių operacinėse sistemose „Linux“, „Windows“ ir „Mac OS X“[18].

Irankio skanavimo galimybes:

- Tinklo skanavimas: "Nmap" gali identifikuoti tinkle visus esančius įrenginius, tokius kaip serverius, maršrutizatorius, kelvedžius, taip pat kaip jie yra sujungti;
- Operacinės sistemos aptikimas: "Nmap" gali identifikuoti, kokia operacinė sistema veikia pasirinktame įrenginyje, kiek laiko įrenginys jau yra aktyvus, programinės įrangos versijas;
- "Nmap" įrankis ne tik aptinka įrenginius tinkle, bet taip pat kokia jų paskirtis, ar tai yra internetinės sistemos serveris ar pašto serveris, ar kažkas kito, taip pat jis aptinka su tuo susijusios programinės įrangos versijas;
- Saugumo auditavimas: Taip pat "Nmap" aptinka kokias ugniasienes ar paketų filtrus pasirinktasis irenginys naudoja.

2. Pažeidiamumų ir programinių klaidų analizės metodai

Programinės įrankos ar sistemų auditavimas apima platų spektrą metodikų, analizių. Sistemų auditavimas padeda surasti spragas prieš joms patenkant į galutinį produktą, ar produkto veikimo metu. Sistemu auditavimas taip pat padeda atrasti įsilaužimo paliktus pėdsakus - kenkėjiškų programų failus, ar paliktus potencialius slaptus įejimus.

2.1. Statinė kodo analizė

Statinė analizė suteikia galimybę gauti informacijos apie galimą programos elgesį programos vykdymo metu, nevykdant programos. Statinė analizė tiria išeities kodą ir ieško įtartinų kodo segmentų kurie galėtu turėti spragą. Atlikus teisingai statinę analizę, galima aptikti akivaizdžias klaidas kurių programuotojas galėjo nepastebėti, tai sutaupo laiko bei sumažina spragų kiekį, taip pat galimai aptinkami nenumatyti scenarijai[8]. Kai kurios programavimo aplinkos (Visual Studio, IntelliJ...) atlieka pastovią statine analizę tam, kad programuotojai pamatytų potencialias klaidas prieš sistemos startą.

Statinė analizė padeda aptikti:

- Neįcituotus kintamuosius;
- Potencialias klaidas sistemos išeities kode;
- *Buffer overflow* spragas.

2.2. Dinaminė kodo analizė

Dinaminė analizė vykdoma kai programa jau yra vykdomo stadijoje. Dynaminės analizės metu bandoma igyvendinti visus įmanomus scenarijus ir išbandyti visas imanomas įvesčių variacijas suvedant jas į programos ivestį. Dinaminė analizė galima taikyti modifikuotoms programoms, virusams ir kitiems paleidžiamiems projektams [4].

Veikimo metu programa gali neatlaisvinti atminties atgal į operacinę sistemą, to pasekoje serveris kuriame programa veikia, pritruks atminties ir pradės veikti lečiau kol galiausiai sustos. Nuo to padėtų apsaugoti dinaminė analizė, atlikus ją teisingai, galima aptikti didžiają dalį spragų kurios potencialiai labiausiai įtakos sistemą. Jas ištaisius, sistema veikimo stabilumas padidėja, nenumatytų scenarijų skaičius taip pat pamažėja.

Dinaminė analizė padeda aptikti:

- Atminties nutekėjimus;
- Netikėtus scenarijus;
- Opiausias spragas;

2.3. Išorinių spragų skenavimas

Išorinis pažeidžiamumų skenavimas atliekamas iš sistemos tinklo išorės, o pagrindinis jo tikslas yra aptikti perimetro gynybos spragas, pavyzdžiui: atvirus tinklo užkardos prievadus ar specializuotą žiniatinklio programų užkardą[12]. Išorinis pažeidžiamumų skenavimas gali padėti organizacijoms išspręsti saugumo problemas, kurios įsilaužėliams galėtų suteikti prieigą prie organizaci-

jos tinklo.
Išorinis pažeidžiamumų skenavimas aptiks:

- Didžiausios tiesioginės grėsmės sistemoje;
- Programinę įrangą kuriai reikia atnaujinimų bei priežiuros;
- Atidaryti prievadus ir protokolus - įėjimo taškus į sistemos tinklą;

2.4. Vidinių pažeidžiamumų skenavimas

Vidinis pažeidžiamumo patikrinimas atliekamas iš organizacijos perimetro gynybos [3]. Jos tikslas yra aptikti pažeidžiamumus, kuriuos galėtų išnaudoti įsilaužėliai arba nepatenkinti darbuotojai, sėkmingai įsiskverbiantys į perimetro gynybą, arba turintys teisėtą prieigą prie organizacijos tinklo.

Vidinių pažeidžiamumų skenavimas aptiks:

- Sistemos komponentus kurie galimai gali sukelti gresmę;
- Pasenusi programinė įranga, kuriai reikia atnaujinimų;

2.5. Oligomorfinių virusų skenavimas

Virusų kurėjai greitai suprato, kad užšifruotą virusą antivirusinei programinei įrangai aptikti yra paprasta, kol paties iššifruotojo kodas yra pakankamai ilgas ir pakankamai unikalus. Norėdami apgauti antivirusinius produktus, jie nusprendė įgyvendinti techninį ieškojimą, jie nusprendė įdiegti mutavusių iššifruoklių kūrimo būdus[25].

2.6. Polimorfinių virusų skenavimas

Polimorfiniai virusai gali iššifruoti jų iššifratorius iki daugybės skirtingų atvejų, kurie gali pasireikšti milijonais skirtingų formų[25].

2.7. Metamorfinių virusų skenavimas

Metamorfiniai virusai neturi iššifruotojo ar nuolatinio viruso kūno, tačiau sugeba sukurti naujas kartas, kurios atrodo kitaip. jie nenaudoja duomenų srities užpildo su styginių konstantomis, tačiau turi vieną vieno kodo pagrindą, kuris duomenis kaupia kaip kodą[25].

3. Gerosios praktikos

Gerųjų praktikų pritaikymas sitemoje ar projekte gali padėti atrasti ęsamus pažeidžiamumus, bei užtikrinti mažesnį skaičių busimų pažeidžiamumų. Gerosios praktikos turi būti taikomos ne vieną kartą, o pastoviai, kiekviename darbe, tik taip galima užtikrinti maksimalų gerųjų praktikų našumą sistemos saugume.

3.1. Atviro ir uždaro kodo programinė įranga

Visame pasaulyje vis daugiau dėmesio skiriama atvirojo kodo programinei įrangai, ypač operacinei sistemai "Linux" ir įvairioms programoms kurios būtent veikia su šia operacinę sistema. Įvairios didžiosios įmonės ir vyriausybės vis labiau priema atviro kodo modelį. Dėl to yra daugybė publikacijų apie atviro kodo pranašumus ir trūkumus. Vykstančios diskusijos apima platų temų spektrą, pavyzdžiui, „Windows“ lyginimas su „Linux“, išlaidų klausimus, intelektinės nuosavybės teises, kūrimo metodus ir panašias temas. Atkreipiant dėmesį būtent į saugumo problemas susijusias su atviro ir uždaro kodo metodika, kompiuterių saugumo bendruomenėje tapo gana nusistovėjęs įsitikinimas, kad dizaino ir protokolų publikavimas prisideda prie jų pagrindu sukurtų sistemų saugumo[13]. Bet ar iš tiesų išeities kodo publikavimas prisideda sistemos saugumo daugiau negu uždaras išeities kodas? Šis klausimas sukelia daug diskusijų ir vieno aiškaus atsakymo niekada nebūna, dauguma specialistų sutinka su tokia nuomone, kad tiek uždaras kodas, tiek atviras kodas turi savų pliusų ir minusų. Todėl peršasi išvada, kad paprasto atsakymo nėra į šį klausimą, ir vienintelis sprendimas tokiai dilemai yra įsigilinti į abi šias metodikas, ir išsiaiškinti, kuo viena metodika pranašesnė už kitą, ir kur atsiranda trūkūmų.

3.1.1. Atviro kodo programinė įranga

Argumentai prieš atvirą kodą:

- Atviras kodas suteikia didelį pranašumą atakuojančiui asmeniui dėl spragų radimo. Atakuojančiam asmeniui reikia surasti vieną spragą su kuria jis galėtu sekmingai užpulti sistemą, o programuotojams reikia ištaisyti visas spragas, kurios neleistu atakuojančiam asmeniui to padaryti[5].

- Yra didelis skirtumas tarp atviro dizaino ir atviro kodo. Atviras dizainas gali atskleisti logines klaidas kurios gali pakenkti sistemos saugumui. Bet skyrus pakankamai dėmesio ir peržiurėjus kodą pakankamai gerai, šios klaidos gali būti rastos ir ištaisytos, skirtingai negu atvirame kode kur klaidas aptikti yra ženkliai sunkiau [13].

- Atakuotojai gali apsimesti programuotojais kurie nori prisidėti prie atviro kodo sistemos kurimo ir palaikymo siulydami savo pataisymus kuriuose slepiasi slapti įejimai ar kitas klaidinantis kodas kuris iš pirmo žvilgsnio atlieka savo funkciją, bet įsigilinus pasimato, kad šie pataisymai yra skirti suteikti pranašumą puolėjuj[26].

- Kodo uždarymas užkerta kelią atakuojančiui asmeniui lengvai gauti informacijos apie sistemą ir jos spragas, priešingai negu laikant kodą atvira. Laikant kodą atvira atakuojantis asmuo gali labai lengvai rasti spragas, kurios jam padėtų įsilaužti į sistemą, arba jai pakenkti[13].

- Viena iš didžiausių priežasčių kodėl atviras kodas nėra idealus pasirinkimas yra tai, kad kodo atvirumas negarantuoja, kad kodą peržiurės kvalifikuoti specialistai, kurie suteiks savo įžvalgas[26].

- Atviro kodo projektai daug dažniau nustoja būti aktyvus negu uždaro kodo projektai dėl finansinių ar kitų priežasčių. Kadangi projektai būna neaktyvus ir nebepalaikomi, rastos klaidos nebėra taisomos, o sistemą kuri naudoja tokį produktą, turi didelią saugumo spagą[26].

Argumentai už atviro kodo programinę įranga:

- Atidarant kodą visiems, yra lengviau ir greičiau identifikuoti problemas, negu uždarant kodą. Atidarius koda visiems, jį vertina ne tik kurėjai, bet ir vartotojai, bei kitos grupės žmonių, kurių deka spragos yra pamatos daug ankščiau lyginant su uždaro kodo produktais, taip pat jas pastebėti yra žymiai lengviau dėl būtent bendruomenės, išvengiama rizika, kad spraga bus nepastebėta ilgą laiką kol ją išnaudos nuostolio siekentys žmonės[20].

- Žmonės naudojantys atviro kodo programinė įrangą galės patys surasti ir sutvarkyti problemas kilusias su produktu, tuo atvėju jie gali savo pataisymus siulyti į pagrindinę produkto repozitoriją, tokie pataisymai bus patvirtinti ir atsiras pačiame produkte, tuomet kiti žmonės galės parsisiusti šiuos pakeitimus pas save, taip padidindami savo sistemos saugumą. "*Linus's law: Given enough eyeballs, all bugs are shallow*[15]"

3.1.2. Uždaro kodo programinė įranga

Argumentai prieš uždarą kodą:

- Uždaro kodo programinės įrangos kokybė dažnai nėra tokia aukšta kaip galima būtų tikėtis. Manoma, kad atviro kodo projektuose kokybės kontrolė būna beveik neegzistuojanti, ir ją užtikrinti yra gan sunku dėl didelio skaičiaus žmonių kurie nori prisidėti prie projekto, jų kodas buna tikrinamas pavirsutiniskai dėl laiko taupymo to rezultatas yra prasta kodo kokybė ir didejantis potencialas spragoms. Bet kaip galima pastebėti, paviešistas kodas kuris visada arba ilgai buvo uždaras dažnai būna ypatingai prastos kokybės ir iš to paviešinto kodo spragos būna išgaunamos labai greitai. Viena iš to priežasčių galima būtų teigti, kad parašytas kodas uždarame projekte patikrinimas ženkliai mažiau žmonių negu atviro projekto kodas, kurį gali tikrinti visi. Saugumo specialistai analizuojantys tokį kodą greitai atranda spragas ir paviešina įrankius, skirtus išnaudoti tokias spragas. Vienas pavyzdžių kai kaikurios Microsoft Windows NT 4.0 produkto kodo dalys buvo paviešintos, keliu dienų eigoje pirmieji įrankiai buvo sukurti tam, kad išnaudotų spragas esančias šiame produkte[13].

- Uždaras kodas neužtikrina, kad spragos nebus rastos, nors kodas ir yra neprieinamas, vis tobulėjantys irankiai skirti dekompiliuoti produktą tam, kad išgauti jame esanti kodą ir rasti spragas, palengvina darbui saugumo specialistams ir asmenims kurie nori pasinaudoti išgautomis spragomis. Dažni atvėjai kai uždaro kodo produkto spragos būna paviešinamos, šios spragos būna užtaisomos atnaujinimais, bet kaikurios spragos būna rastos ir nepaviešintos tam, kad niekad jų nesutvarkytų, ir atakuojantys asmenys turetu ilgesnį laiką išnaudoti šias spragas. Iš to galima teigti, kad nors ir spragos buna lečiau ir sunkiau surandamos uždarame produkte, tų spragų paviešinimas yra daug greitesnis potencialiai atviro kodo produktuose[17].

- Atviro kodo produktai pasižymi savo bendruomenė, dažnai prie atviro kodo produkto gali prisidėti visi kas tik gali. Todėl kodas būna tvarkomas daug greičiau, vartotojai patys randa problemas kode, informuoja apie tai kurėjus, dažnai net pasiulo savo sprendimus. Tokiais atvėjais kurėjams nebereikia patiems investuoti laiko sprendžiant problemą, užtenka tiesiog peržiurėti vartotojo sprendimą, ir jei viskas tinka - jį pritaikyti. Tuo nepasižymi uždaro kodo produktai, vartotojai beveik negali, arba išvis negali prisidėti prie produkto kurimo. Vartotojai taip pat negali kurėjams patarti produkto kurimo klausimais, ar padėti ištaisyti klaidas. Kurėjai turi patys aiškintis tas klaidas ir dažnai tokių problemų sprendimas užtrunka ilgai, nes tam reikia investuoti daug laiko ir resursų labai opių spragų sprendimas kartais užtrunka savaites ar net mėnesius[13].

Argumentas už uždaro kodo saugumą yra toks, kad uždaras kodas gali turėti spragų kurias galėtų išnaudoti atakuojantys asmenys, bet atsiranda daug didesnis šansas, kad jų tiesiog neišnaudos, nes apie jas nežinos[10]. Priešingai negu atviro kodo programinės įrangos spragas, kurias rasti yra neitikėtinai daug lengviau ir rasti gali bet kas ir apie tai neprivalo pranešti. Tuomet gali galvoti kaip tai išnaudoti savo reikmėms, net jeigu ir spraga yra užtaisoma greitai, nereiškia, kad tuo užtaisymu nėra padaroma kita spragą, kuria kiti asmenys vėl taip pat lengvai gali rasti ir išnaudoti. Kaip pavyzdi, galima žiurėti į Microsoft Windows NT 4.0 produkto spragas prieš paviešinimą, jos buvo daugybe metų, bet jų niekad nerado ir neišnaudojo, kai kodas buvo nutekintas, jas rado per pirmas kelias dienas[13]. Galima teigti, kad labai gerai prižiūrimas uždaras kodas turi tikrai didelį pliusą, nes jeigu ir jame yra spraga, gali būti kad jos niekas ilgai neras, o tuo tarpu patys kurėjai ją turi laiko pastebėti daug daugiau[24].

3.1.3. Apibendrinimas

Saugesnė atvirojo kodo programinė įranga ar uždaro kodo programinė įranga? Galima teigti, kad dižiausias atviro kodo pliusas yra tai kad vartotojas gali peržiurėti programos kodą prieš ją naudodamas, skirtingai negu uždaro kodo programinė įranga, kuria vartotojas turi pasitikėti aklai[8]. Atviro kodo programinė įranga tiek užpuolikams, tiek gynėjams suteikia didesnę galimybę imtis veiksmų, vienintelis klausimas, kas pirmas pastebės spraga, ir ką su ja darys[17]. Uždaro kodo programinė įranga turiu pliusų žiurint iš komercinės prasmės, bet iš saugumo pusės - sakymas, kad saugumą užtikrina kodo slepimas, galima teigti, kad jis yra labai nepasitvirtines ir turint dabartines dekompiliavimo galimybės, nebegali teigti, kad tai yra tiesa [13]. Taigi, atsižvelgiant į visus argumentus ir pavyzdžius, galima teigti, kad saugumo prasme, geresnis pasirinkimas būtų rinktis atviro kodo produktus savo sistemai.

3.2. Sistemos auditavimas

Sistemos auditavimo metodikų yra didelis skaičius ir kuo daugiau jų yra taikoma, tuo daugiau naudos jie gali atnešti sistemos saugumui. Sistemų auditavimas susideda ne vien iš skenavimo ar analizės metodų, bet ir iš paprastų kasdienių konfiguravimo ar programavimo darbų.

3.2.1. Sistemos ir jos komponentų atnaujimas

Vienas iš svarbesnių darbų kuriuos yra privaloma atlikti sistemoje norint užtikrinti sistemos saugumą - sistemos ir jos programanių komponentų atnaujimas. Dažnas atvėjis, kai senose operacinės sistemos, ar programinės įrangos versijose atrandama pažeidžiamumų, kuriais pasinaudojus

įsilaužialiai gali patekti į sistemą. Problema kyla tada, kai vartotojai neatnaujina savo sistemose ęsančių operacinių sistemų ar programinės įrangos, sistemos tampa taikiniais, o kiekvienas įsilaužėlis turi raktą tiesiai į ją. Kaip pavyzdį galima pateikti aptiktą Microsoft Windows operacinėse sistemose buvusią spragą, vadinama „BlueKeep“. Šis pažeidžiamumas leisdavo bet kokiam puolėjui patekti į sistemą kuri turi Microsoft Windows operacinės sistemos tam tikrą versiją naudojant RDP ir neturint jos prisijungimo duomenų[1]. Problema buvo sutvarkyta, pažeidžiamumas ištaisytas naujesniuose operacinės sistemos atnaujinimuose. Iškylanti problema kuri egzistuoja dabar yra ta, kad skaičiuojama, kad milijonas sistemų iki dabar yra pažeidžiamos, dėl to, kad jos tiesiog nėra atnaujinamos[23].

3.2.2. Pažeidžiamumų ir programinių klaidų analizės metodų taikymas

Pažeidžiamumų ir programinių klaidų analizės metodų taikymas padėtų rasti pažeidžiamumus internetinėse svėtainėse ar jų sistemose veikimo metu, ar svėtainės programavimo metu. Šie metodai yra aprašomi 2 skyriuje. Teisingai juos atliekant reguleriai, galima išvengti įvairių programavimo metu programos logikoje padarytų klaidų kurios potencialiai pasireikštų ne iš karto ir lauktų kaip tiksinčios laiko bombos. Statines ir dinamines kodo analizės atlieka dažnai atlieka programavimo aplinkos, kurios ispėja apie galimas būsimas problemas programavimo metu. Kitos analizės susijusios su kenkėjiškų programų radimu yra plačiai naudojamos populiariausių antivirusinių sistemų. Todėl gera praktika būtų turėti laiko patikrintą, geros reputacijos kenkėjiškų programų ieškojimo įrankį, arba dar kitaip - antivirusinę.

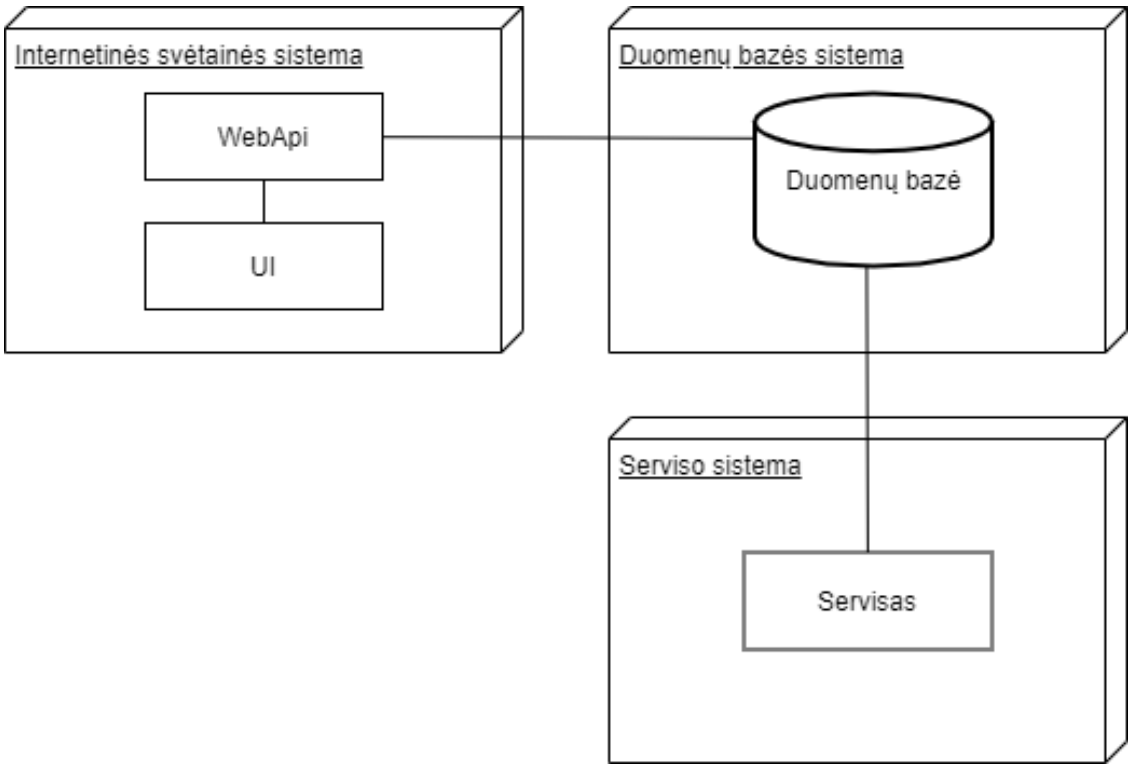
4. Sistemų auditavimo įrankis

Kurimo darbo metu buvo vadovaujamasi šiuo darbo tikslu: Sukurti saugų pažeidžiamumų skenavimo įrankį, kuris veiktu iš tam tikros slepiamos vietos, skenuotų internetinę svetainę bei jos failus, aptiktų pažeidžiamumus ar infekuotus failus, ir pateiktų klientui informaciją apie skenavimo rezultatus. Šis tikslas pasiektas tokiais veiksmais:

- Užtikrinamas saugus ryšys tarp įrankio ir skanuojamos sistemos;
- Sukurta saugi aplinka į kurią galima būtų siusti potencialiai užkrėtus failus;
- Išorinis sistemos skanavimas, bandant išgauti kuo daugiau informacijos iš sistemos, ieškant atidarytų prievadų ar beveikenčių sistemų skenuojamoje sistemoje;
- Tikrinamas pats svetainės adresas, ar yra saugu eiti į jį;
- Parsisiusti failai iš nurodyto *FTP* serverio yra tikrinami ar jų tūriniai nėra potencialiai infekuoti ir kelentys grėsmę;

4.1. Įrankio architektūra

Skenavimo įrankio architektūra yra paremta Nessus įrankio architektūra, vartotojo sąsaja tiesiogiai nebendrauja su servisu, kuris vykdo visus skenavimo procesus. Visa architektūra yra modulinė - išskirstyta per tris atskiras sistemas. Tokia architektūra buvo pasirinkta dėl sistemos saugumo ir stabilumo. Vienai sistemai sutrikus, sutrikimas visiškai neįtakoja kitų sistemų. Taip pat, jeigu įvyktų įsilaužimas į viena iš sistemų, įsilaužėliai neturėt prieigos prie viso projekto.

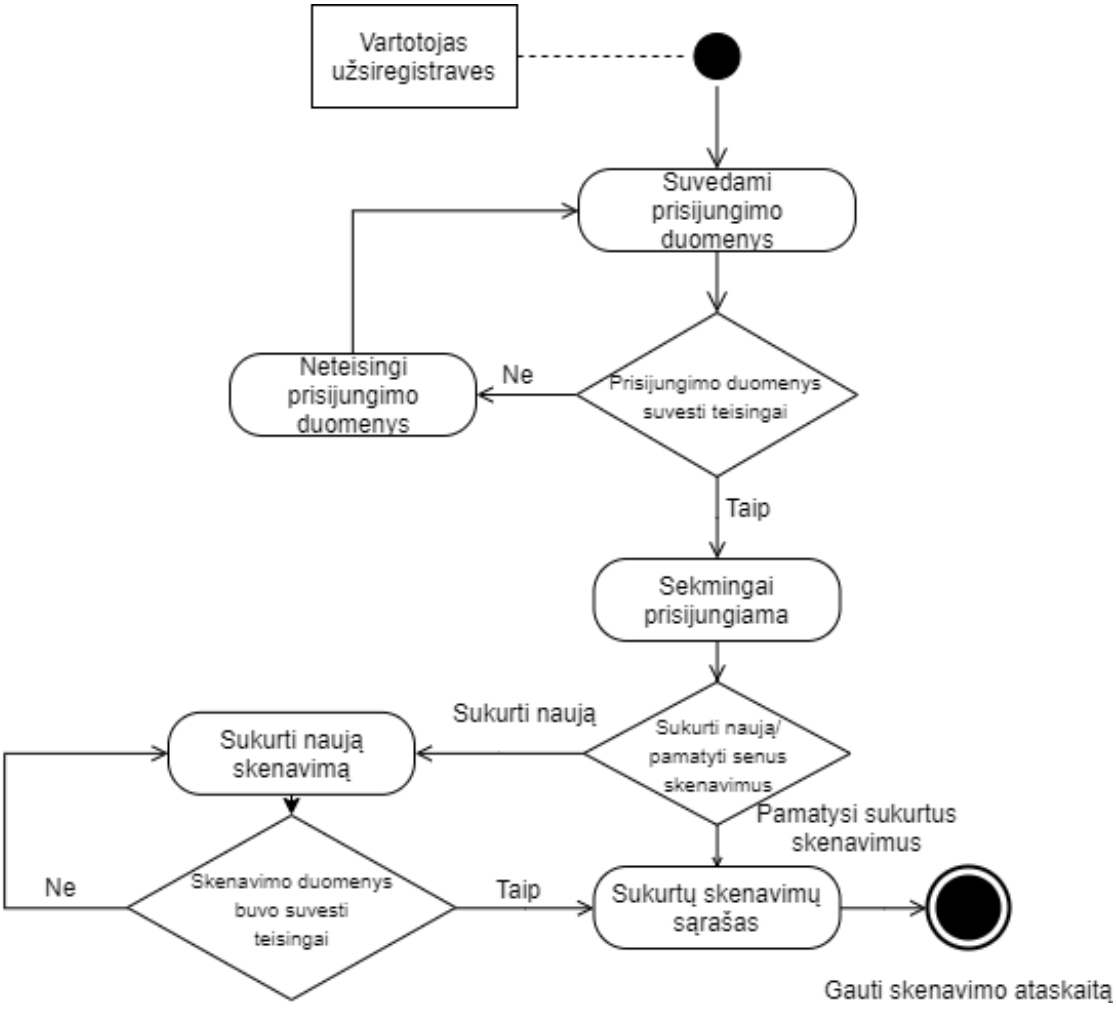


1 pav. Sistemos architektūros schema

Sistemos architektūros schemoje 1 pavyzdyje, matome tris sistemas - internetinės svetainės sistemą, duomenų bazės sistemą ir serviso sistemą. Internetinės svėtainės sistemoje veikia pati internetinė svetainė, kurioje vyksta prisijungimas prie sistemos, skenavimo užklausu kurimas, ir skenavimo atąskaitų parsisiuntimas. Duomenų bazės sistemoje veikia pati duomenų bazė, kurios paskirtis yra laikyti skenavimo užklausas ir jų rezultatus, taip pat laikyti prisijungo duomenis. Serviso sistema veikia pats servisas kuris atlieką visa skenavimo logiką ir visą bendravima su skenuojama internetine svetaine. Taip pat bendrauja ir su duomenų baze, iš jos pasiema visus duomenis reikalingus skenavimui ir į ją deda visus skenavimo rezultatus.

UI aplikacija internės svetainės sistemoje sukurta naudojant Angular. WebApi kuris atsakingas už visą bendravima su duomenų baze bei skenavimo ataskaitų formavima, sukurtas naujant ASP.NET Core. Duomenų bazė sukurta naudojant Microsoft SQL. Serviso sistemoje esanti serviso aplikacija sukurta naudojant .NET Core su kuriuo parašyta visa pararelinė skenavimų paleidimo logika ir bendravimas su duomenų baze, Bash scriptus, kurie skirti kurti konteinerius ir vykdyti skenavimus, Docker kuris skirtas kurti konteinerius ir užtikrinti visos sistemos saugumą ir stabilumą. Visos sistemos naudoja Ubuntu 16.04 operacinę sistemą.

4.2. Vartotojo sąsaja

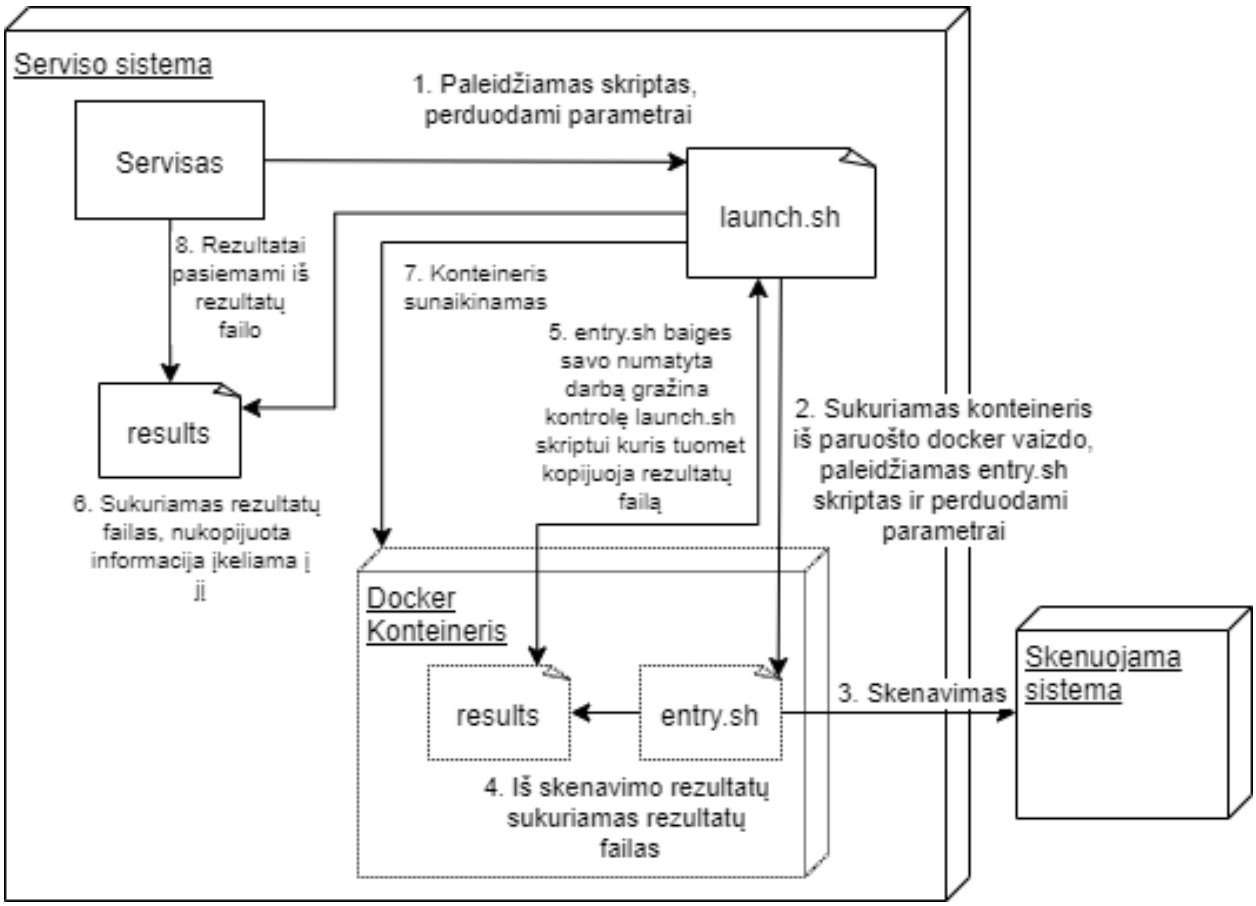


2 pav. Aktyvumo diagrama

Aktyvumo diagramoje kuri yra 2 pavyzdyje, galima matyti visus pasirinkimus kuris vartotojas turi. Vartotojo sąsaja sukurta naudojant Angular, visi atliekami veiksmai keliauja į API kuris sukurtas su ASP.NET Core. API vyksta visa internetinės svetainės logika - autentifikacijos valdymas, bei duomenų valdymas. Jungimosi metu į UI suvedami duomenys keliauja į API, kuriama prisijungimo duomenys yra verifikuojami su duomenų baze. Sekmingai prisijungus, vartotojas gali arba kurti naują skenavimo užklausą, arba pamatyti jau sukurtas. Kuriant skenavimo užklausa, duomenys taip pat yra tikrinami, patikrinus ir sekmingai užregistravus skenavimo užklausa, vartotojas gali eiti į skenavimų sąraša, kur bus pateikta jo sukurtiems skenavimas ataskaitų parsisiuntimo nuorodos, arba jis gali kurti vėl naują užklausą.

4.3. Saugios aplinkos užtikrinimas

Internetinių svėtainių skenavimo įrankyje saugi aplinka užtikrinama naudojant Docker, kuris pasižymi tuo, kad su ja yra kuriami konteineriai, kurie yra įzoliuoti nuo likusios sistemos. Pačia Docker technologiją galima lyginti su virtualiomis mašinomis kurias pavyzdžiui kuria VirtualBox įrankis. Skirtumas tarp jų vis dėl to yra didžiulis. Docker konteineriai yra kuriami operacinės sistemos lygyje skirtingai negu VirtualBox virtualios mašinos kurios kuriamos geležies lygyje, taip pat kiekviena virtuali mašina turi turėti savo atskirą operacinę sistemą, o konteineriai tiesiog naudoja tapačia opericinę sitema kurioje jie yra kuriami, todėl konteineriai reikalauja žymiai mažiau išteklių, veikia žymiai greičiau, juos galima greitai kurti ir naikinti[16]. Visa tai yra svarbu sistemai, nes kiekvienam procesui yra kuriamas atskirame konteineris, po kiekvieno proceso įgyvendinimo, konteineris yra sunaikinamas. Konteineriu greitis leidžia tai padaryti greitai ir saugiai, nėra priežasties kodėl konteinerius reiktų pernaudoti, nereikia surašinėti atskirai operacinių sistemų ir jas konfiguruoti, taip pat mažas resursu kiekis nestabdo visos sistemos bendrai, ir užtikrina, kad sistemą nesustos veikti.



3 pav. Saugios aplinkos užtikrinimo veikimas

Pilnoje saugios aplinkos įgyvendinimo schemoje kuri yra 3 pavyzdyje, matome visą procesą, kuris užtikrina sistemos ir įrankio saugumą. Kiekvieno žingsnio kuris yra įgyvendinamas paaiškinimas:

1. Iš pradžių servisas paleidžia „launch.sh“ bash skriptą, tuo pačiu skriptui perduodamas parametrus reikalingus skenavimui įgyvendinti. Servisas viso šio skripto metu laukia, kol skriptas pabaigs savo darbą.

2. Skriptas „launch.sh“ sukuria konteinerį pagal pasirinktą docker vaizdą. Kiekvienas atliekamas skenavimas turi savo specifinį vaizdą kurie yra sukuriami vieną karta, ir visa laika laikomi išsaugoti. Kurdamas konteineri skriptas taip pat į konteinerį perduoda bash komandą, kuri paleidžia „entry.sh“ skriptą ęsanti konteineryje ir paleidimo metu perduoda jam parametrus, kuriuos pats gavo iš serviso. Skriptas konteineryje atsiranda kartu su jo sukurimu, nes šis skriptas taip pat saugomas vaizde. Skriptas „launch.sh“ paleides skriptą „entry.sh“ laukia, kol jis baigs savo darbą.

3. Skriptas „entry.sh“ įgyvendina skenavimo komandas su gautais parametrais iš „launch.sh“ skripto. Jeigu skenavimo metu tenka parsisiusti failų iš skenuojamos sistemos, šis skriptas taip pat juos parsiunčia ir patalpina į konteinerį.

4. Skriptas „entry.sh“ įgyvendines skenavimo komandas, sukuria rezultatų failą, į kurį įdeda rezultatus, prireikus juos suformatuoja tam tikra tvarka prieš įdėdamas. Tuomet jis baigia savo darbą.

5. Skriptas „launch.sh“ sulaukes skripto „entry.sh“ pabaigos, kopijuoja rezultatų failą esantį konteineryje.

6. Tuomet skriptas „launch.sh“ sukuria failą pagrindinėje sistemoje ir įkelia į jį nukopijuotus duomenis.

7. Skriptas „launch.sh“ sunaikina konteineri su visais jame esančiai failais ir baigia savo darbą. Jeigu konteineryje buvo atsiusta failu iš skenuojamos sistemos, jie taip pat yra sunaikinami.

8. Servisas sulaukia skripto „launch.sh“ pabaigos ir toliau vykdo savo darbą, pasiema rezultatus iš rezultatų failo ir juos naudoja tolimesniuose procesuose.

4.4. Igyvendinami skenavimo metodai

Iš viso yra įgyvendinti keturi skenavimo metodai. Šie metodai apriebia didelį skaičių potencialių pažeidžiamumų, taip pat su šiais metodais galima aptikti dažniausiai esančius pažeidžiamumus, kurie pasitaiko sistemose. Metodai apima atvejus ne vien tik situacijas kai esantis pažeidžiamumai yra aptinkami prieš įsilaužima, bet įsilaužimo ir po isilaužimo situacijas.

4.4.1. Išorinis sistemos skenavimas

Išorinis sistemos skenavimas skirtas tam, kad aptiktų atvirus prievadus, patikrinti, kokią informacija išduota pati sistema - kokią operacinę sistemą pati sistema naudoja, kokia tos operacinės sistemos versija, kokios aplikacijos ar sistemos veikia toje sistemoje, ar prie šių sistemų galima jungtis, ar gali prisijungti anoniminei vartotojai. Skenavimo įgyvendinimo metu yra sukuriamas konteineris, iš kurio yra atliekamas skenavimas. Skenavimui pasibaigus yra surenkami rezultatai ir atiduodami servisui, konteineris yra saugiai sunaikinamas, o rezultatai patalpinami į duomenų bazę.

4.4.2. Failų skenavimas

Internetinės sistemos failų skevimas skirtas tam, kad patikrinti ar į sistemą jau nebuvo isilaužta, ir joje nėra paliktu kenkėjiškų programų. Šis skenavimas yra įgyvendinimas skenavimo užklausos metu. Sukuriamas Docker konteineris, iš kurio naudojant *FTP* prisijungiama prie sistemos. Iš *FTP* direktorijos yra parsiunčiami visi failai į konteineri, tuomet yra generuojami visų failų MD5 maišos žodžiai, kurie yra paruošiami tikrinimui ir yra patikrinami žinomų kenkėjiškų programų failų MD5 maišos žodžių duomenų bazėje. Tuomet rezultatai yra gražinami į servisą, o pats konteineris yra saugiai sunaikinamas. Rezultatai patalpinami į duomenų bazę.

4.4.3. Internetinės svėtainės prieigos skenavimas

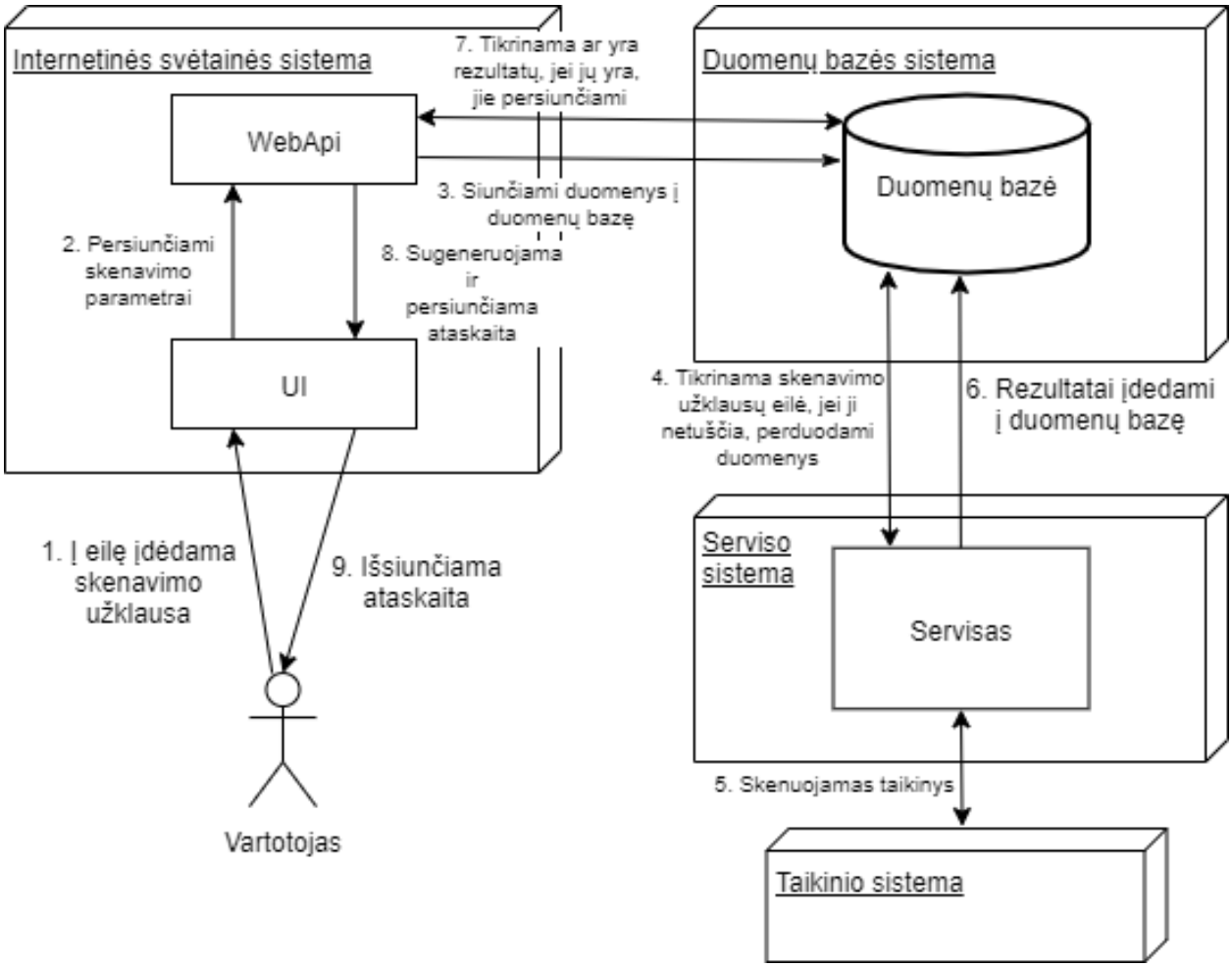
Internetinės svėtainės prieigos skenavimas yra skirtas tam kad aptiktų potencialias *MITM* atakas, fišingo svėtaines, kenkėjiškų programų svėtaines. Veikimas yra toks, kad svetainės adresas yra nusiunčiamas trečiajai šaliai, kuri svetainės adresą skenuoja su įvairiomis antivirusinėmis. Tuomet yra pateikiama detali ataskaita. Kadangi šio skenavimo metu nera kontaktuojama su pačia svetaine tiesiogiai, Docker konteineriai nėra kuriami.

4.4.4. Internetinės svėtainės enumeravimas

Internetinės svėtainės enumeravimas yra skirtas tam, kad aptiktu potencialias internetinės svėtainės spragas tokias kaip atviras direktorijas, kuriose puolėjas gali be jokių kliučių perskaityti jose ęsančių failų turinį. Taip pat aptinkama visi internetinės svetainės puslapiai, prie kurių vartotojas gali prieiti be jokios autentifikacijos, sakykime prie administratoriaus puslapio, šios problemos kyla dėl blogai sukonfiguruotų teisių ar pačio puslapio autentifikacijos. Skenavimo metu yra sukuriamas konteineris, kuriame vykdomas enumeravimas, konteineryje susigeneruoja rezultatų failas, kuris yra gražinamas servisui, o pats konteineris yra saugiai sunaikinamas. Rezultatai yra patalpinami į duomenų bazę.

4.5. Skenavimo užklausų įgyvendinimas

Skenavimo užklausų įgyvendinimas yra svarbiausias procesas šiame įrankyje. Skyriuje 4.5 bus paaiškinama visas skenavimo užklausos įgyvendinimas nuo jos sukurimo iki skenavimo užklausos ataskaitos parsisiuntimo.



4 pav. Įrankio veikimo schema

Įrankio veikimo schemoje kuri yra 4 pavyzdyje galima matyti daug procesu kurie veikia paeiliui arba pararelai. Kiekvieno proceso paaiškinamas paeiliui:

1. Prisijunges vartotojas kurdamas skenavimo užklausą suvedada atitinkamus parametrus tokius kaip: Internetinės svėtainės adresą, FTP atrasą, FTP prisijungimo duomenis;

2. Duomenys yra siunčiami į WebApi, kuriame vyksta visa duomenų verifikavimo logika, tikrinama ar duomenis yra geri, ar adresai yra teisingai suvesti;

3. Verifikuoti duomenys yra siunčiami į duomenų baze, ir yra išsaugomi;

4. Servise veikia laikmatis, kuris praejus tam tikrai laikui vis tikra ar duomenų bazėje nėra skenavimo užklausų, kurios dar nebuvo įgyvendintos. Jei tokių užklausų yra, duomenys yra paiemami;

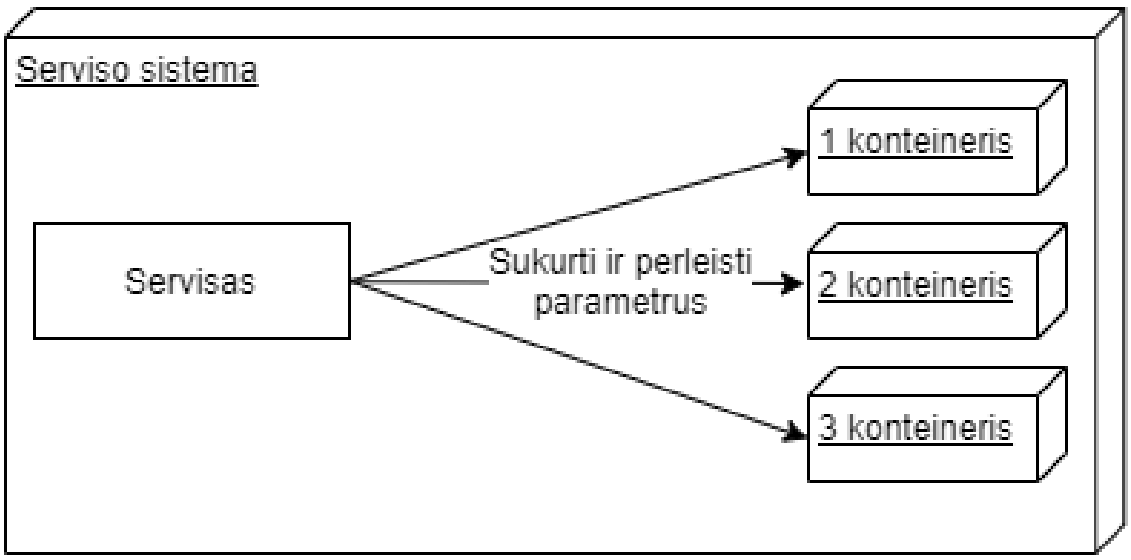
5. Pradedama vykdyti skenavimo logika kiekvienai užklaisai paeiliui. Užklausos vykdomos paeiliui, o ne iš karto visos tam, kad užtikrinti sistemos stabilumą ir minimalų resursų naudojimą. Jeigu vienu būtu įgyvendinamos šimtai užklausų, rizikuojama, kad sistema pritruks resursų visoms operacijoms įgyvendinti. Nors ir kiekviena užklausa įgyvendinama paeiliui, bet kiekvienai kiekvienai užklaisai skenavimo operacijos yra vykdomomos parareliai. Šiuo atveju nėra rizikuojama, kad sistema pritruks resursų, todėl, kad vykdomas fiksuotas skaičius operacijų;

6. Baigusis operacijoms rezultatai yra formatuojami ir įdedami į duomenų bazę.

7. Tuo tarpu WebApi vykdo užklausas į duomenų bazę ar dar nėra skenavimo užklausos duomenų kiekvieną karta, kai vartotojas perkrauna puslapį, kuriame yra skenavimo užklausų sąrašas, gavus atsaka, kad duomenų yra, vartotojuj leidžiama atsisiūsti ataskaitą;

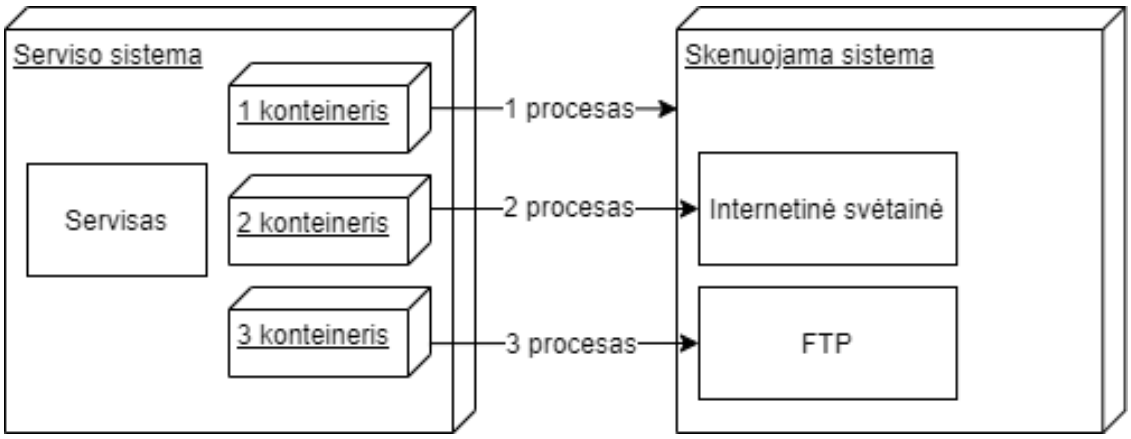
8. Vartotojas duoda užklausą gauti skenavimo užklausos ataskaitą. Tuo metu pasiemami duomenys iš duomenų bazės ir yra generuojama ataskaitą HTML formatu. Sugeneruojamas parsisiuntimo adresas.

9. Vartotojuj leidžiama parsisiusti ataskaitą.



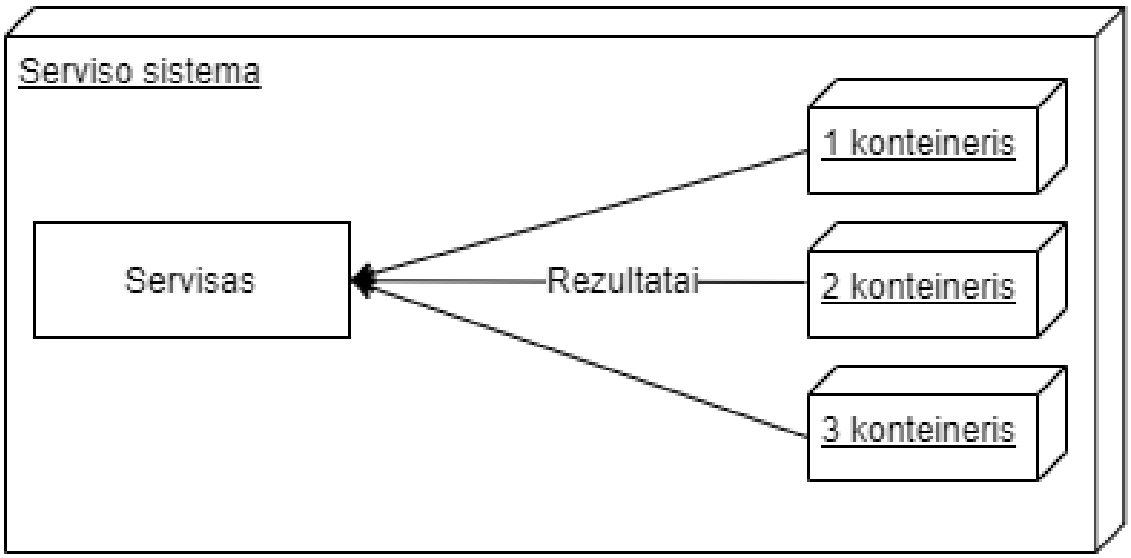
5 pav. Konteinerių sukurimas įgyvendinant skenavimo užklausas

Serviso sistemoj įgyvendinant skenavimo užklausas yra kuriami trys docker konteineriai kaip pavaizduota 5 pavizdyje. Apie patį konteinerių kurimo procesa buvo rašyta 4.3 skyriuje. Kiekvieno konteinerio procesas reikalauja skirtingų parametrų, nes atlieka skirtingus darbus. Jie gaunami 4 pavyzdžio pirmajame punkte.



6 pav. Proceso Veikimas

Proceso veikimas pavaizduotas 6 pavizdyje, ir jame matome, kad visi konteineriai atlieka skirtingus procesus paraleliškai, pirmasis konteineris atlieka skenuojamos sistemos išorinę analizę apie kurią daugiau buvo rašyta 4.4.1 skyriuje, šiam konteineriui kaip parametras yra paduodamas sistemos IP adresas. Antrasis konteineris atlieka skenuojamos internetinės svėtainės enumeraciją, kuri aprašyta 4.4.4, šiam konteineriui kaip parametras yra paduodamas internetinės svėtainės adresas. Trečiasis konteineris jungiasi į prie serverio naudodamas FTP protokolą ir atlieka failų skenavimą, kuris aprašytas 4.4.2 skyriuje, šiam konteineriui per parametrus yra paduodi FTP prisijungimo duomenys, bei FTP adresas. Taip pat yra įgyvendinamas ir internetinės prieigos skenavimas kuris aprašytas 4.4.3 skyriuje, tačiau šiam skenavimui nereikalinga prieiga skenuojamos sistemos, todėl konteineris jam nėra kuriamas.



7 pav. Rezultatų gražinimas

Įgyvendinus visus skenavimo procesus, rezultatai yra gražinami pačiam servisui, o patys konteineriai susinaikina. Taip pat internetinės svėtainės prieigos skenavimo rezultatai yra pasiemami, ir perduodami servisui, servisas vėliau juos formatuoja ir talpina į duomenų bazę. Pats duomenų perdavimas ir konteineriu sunaikinimas yra aprašytas 4.3 skyriuje.

4.6. Įrankio Skenavimo metodų platforma

Įrankis yra puikiai pritaikytas naudoti bet kokioje sistemoje, taip pat jį ypač patogu pritaikyti bet kokia scenarijuje. Šis įrankis yra plačiai naudojamas kibernetinio saugumo bendruomeneje ir taip pat yra ypač efektyvus atliekant tinklo skanavimus ar analizę. Atsisžvelgus į šiuos faktus, šis įrankis tampa butinybė bet kokioje spragų skanavimo sistemoje dėl savo didelio potencialo ir bendruomenės pasitikėjimo.

1 algoritmas. Įrankio platformos pseudo kodas

$ScanRequests \leftarrow Database$
if $ScanRequests > 0$ **then**
  for all $ScanRequests$ **do**
    $ThreadPool \leftarrow Scan1(ScanRequest)$
    $ThreadPool \leftarrow Scan2(ScanRequest)$
    $ThreadPool \leftarrow Scan3(ScanRequest)$
    $ThreadPool \leftarrow Scan4(ScanRequest)$
    $ThreadPool.WaitForAll$
  end for
  $Database \leftarrow ThreadPool.Results$
end if

Įrankio kurimo metu buvo sukurta platforma, kuri yra integruota į patį įranki. Platforma yra skirta lengvai pridėti naujus skenavimo būdus ir funkcijas. Iš šios platformos yra startuojami visi jau esantis skenavimo metodai. Naujų skenavimo metodu pridėjimas vyksta programiškai, bet pati platforma parašyta taip, kad norint pridėti kažka naujo, nereikia programuoti visko per naujo. Šios platformos kodą galime matyti 1 algoritme. Iš pradžių pasiemame visas skenavimo užklausas kurios dar nebuvo vykdytos iš duomenų bazės, tuomet iteruojame per kiekviena iš jų ir pararelіai paleisdžiame visus skenavimo metodus. Kai visi skenavimo metodai yra pasileide, laukiame, kol visi jie pasibaigs. Visiems skenavimams pasibaigus, resultatus patalpiname į duomenų bazę.

4.7. Aptiktini pažeidžiamumai

Šiuo metu skenavimo įrankis gali aptikti šiuo pažeidžiamumus:

- Aptiktinos yra *MITM* atakos tikrinant internetinės svetainės atsakus;
- Aptinkama ar puslapis yra nesaugus naudoti ir turi kenkėjiškų programų;
- Aptiktinos direktorijos, kurios sąrašo pavidalu gražina savo turinį, taip atsitinka dėl neteisingai sudėtų teisių sistemoje, dėl šio pažeidžiamumo puolėjas atrades šia direktoriją gali be jokių kliučių peržiurėti joje esančius failus;
- Aptiktini puslapiai, prie kurių neprisijunges vartotojas neturėtu galimybės prieiti. Kaip pavyzdį galima būtų pateikti prisijungimo vietas prie administratoriaus sąsajos;
- Aptiktinkamos kenkėjiškos programos ir infekuoti failai;
- Aptinkamos sistemos, kurios veikia skenuojamoje sistemoje;
- Aptinkami atidaryti prievadai.

4.8. Įgyvendinti uždaviniai

Įgyvendinti šie apsibrėžti uždaviniai:

- Sukurta svėtainė, kuri leidžia vartotojuj kurti skenavimo užklausas, ir po jų sukurimo, leidžia atsisiusti suformatuotus rezultatus;
- Sukurta platforma ir paruoštukai, kurie leidžia lengvai pridėti naujus skenavimo metodus į įrankį;
- Sukurta saugi aplinka, kuri leidžia vartotojuj saugiai skenuoti sistemas ir jų failus, nesibaiminant infekuoti pačio įrankio sistemos;
- Aptinkami pažeidžiamumų ir pati sistema paruošta naudojimui;
- Sugeneruojama ir pateikiama ataskaika kurią lengva suprasti vartotojuj.

4.9. Trūkumai

Įrankio kurimo procesas yra ypac sudetingas dėl didelio skaičiaus skirtingų technologijų, su kuriomis yra kuriamos internetinės svėtainės, taip pat kiekviena internetinė svetainė skiriasi nuo kitų savo sistemos komponentais, architektūra, naudojamomis technologijomis.

Kurimo metu buvo planuota panaudoti trečios šalies įranki SqlMap. Šio įrankio paskirtis yra ieškoti ar internetinė svetainė yra pažeidžiama SQL injekcijos atakoms. Tokio tipo skenavimui reikėtų sukurti internetinės svėtaines puslapių kodo funkciją, kuri svetainės puslapyje iėško dinaminių nuorodų. Šios nuorodos priema parametrus savo nuorodose. Tuomet SqlMap įrankis bando į šiuos parametrus įdeti manipuliuotą tekstą kuris įvykdytų užklausą duomenų bazeje. Šio įrankio funkcionalumo įgyvendinimas šioje skenavimo sistemoje tapo perdaug problematiškas ir laiko užimantis dėl pačios funkcijos kuri ieškotų tokių įveščių internetinėje svėtainėje. Dėl šios priežasties SqlMap funkcionalumo tekta atsisakyti.

Kurimo metu taip pat buvo užsibrežta įgyvendinti statinė analizę. Šis tikslas buvo įgyvendintas nepilnai, patys failai yra tikrinami žinomų kenkėjiškų programų duomenų bazėje, bet internetinės svėtainės kodas nėra tikrinamas. Šio tikslo tekta atsisakyti ir perkelti statinės analizės technikų įgyvendinimus į ateities darbus. Šio atsisakymo priežastis yra ta, kad kiekvienai programavimo kalbai reikalingas atskiras statinės analizės įgyvendinimas, kieviena kalba yra kitokia ir jos funkcionalumas ir sintaksė yra kitokia, dėl šios priežasties tokio funkcionalumo kurimo kaštai stipriai didėja.

Išvados ir rekomendacijos

Rezultatai Sukurtas saugus pažeidžiamumų skaitytuvas sistemų auditavimui naudojantis naujausiomis technologijos. Pažeidžiamumų skaitytuvas yra modulinis, turi tris modulius: Internetinę svetainę, duomenų bazę ir servisą, kuris atlieka visus skenavimus. Visos trys dalys geba veikti atskirai, taip pridedant papildoma saugumo ir stabilumo sluoksnį

Pats skaitytuvas įgyvendina keturis skirtingus skenavimus skirtingiems pažeidžiamumams aptikti. Skaitytuvas geba aptikti išorinius pažeidžiamumus skenuodamas sistomos išorė, taip bandydamas aptikti atidarytus prievadus, sistemos operacinę sistema bei jos versiją, veikiančias kitas sistemas ju tipus ir versijas pagrindinėje sistemoje, tokias kaip: duomenų bazę, internetines svetaines. Taip pat yra bandoma aptikti kokiais protokolais galima prisijungti prie duomenų bazės, ir ištestuoti, ar galima prie jų jungtis anonimiškai. Skaitytuvas taip pat geba jungtis prie sistemos per FTP, prisijungus parsisiusti visus failus ęsančius joje ir tikrinti ar jie egzistuoja žinomų kenkėjiškų programų duomenų bazėje. Skaitytuvas gali ir tikrinti internetinę svetainę, kuri veikia pasirinktoje sistemoje, tikrinama ar ši svėtainė turi direktorijų kurios pateikia savo turinio sąraša, tokiose direktorijose failų turinį gali skaityti bet kas, taip randant neteisingai sudeliotas teises sistemoje. Taip pat ieškoma ir puslapių sąrašo, kuri neautentifikuotas vartotojas gali pasiekti, taip bandoma surasti puslapius, prie kurių vartotojas gali prieiti, taip bandoma surasti, ar yra prieigos taškų, kurie neturėtu būti prieinami kiekvienam. Taip pat yra tikrinama ir pati prieiga prie internetinės svėtainės, bandoma rasti ar nėra vykdoma MITM ataka ir ar internetinėje svetainėje neveikia kenkėjiškos programos.

Skaitytuvo saugumas yra užtikrinamas naudojant docker konteinerius. Parašyti specialius paruoštukai kiekvienam skenavimui kurie leidžia sukurti specifini konteineri kiekvienam skenavimui. Naudojant konteinerių technologija yra pasiekiamas saugumas, kuris apsaugo skaitytuvo sistemą nuo potencialių grėsmių kurios gali slėptis internetinėje svetainėje. Kiekvieno skenavimo užklausa vykdoma paraleliai paleižiant skenavimus, kas užtikrina didesnį greitį. Skenavimo rezultatai keliauja į duomenų bazę iš kurios vėliau yra paemami formuoti ataskaitą. Ataskaita yra specializuota kiekvienam testui.

Internetinė svetainė veikia atskirai nuo visos likusios skaitytuvo struktūros ir yra skirta tik bendrauti su vartotoju. Svėtainėje yra autentifikacija, kuri reikalinga kuriant naujas skenavimo užklausas. Svetainėje galima kurti naujas skenavimo užklausas, peržiurėti visas kitas, ir parsisiusti jų visų ataskaitas.

Išvados Kuo daugiau mūsų gyvenimai priklauso nuo skaitmeninių technologijų, tuo labiau visi yra priklausomi nuo kibernetinės saugos. Vienas iš geriausių būdų užtikrinti sistemos saugumą, yra jos auditavimas. Nuolatos atsiranda naujų grėsmių ir naujų pažeidžiamumų, kuriai naudodamiesi įsilaužėliai gali padaryti didžiulius nuostolius. Todėl ir pats pažeidžiamumų skaitytuvas turi būti nuolatos tobulinamas, tam kad pasivytų naujas grėsmes ir technologijas. Tokie skaitytuvai yra neatsiejami nuo sistemos saugumo auditavimo, dėl savo patogumo ir laiko taupymo. Pažeidžiamumų skaitytuvai kaip niekados ankščiau yra aktualus ir turintys didžiulę naudą. Norint pasiekti maksimalų sistemos sauguma, sistema tenka audituoti dažnai ir be automatizuotų skaitytuvu, toks darbas taptu ilgas, brangus ir reikalaujantis didelių laiko kaštų.

Pažeidžiamumų skaitytuvas yra labai aktualus, bet jo kurimo procesas yra ilgas ir reikalaujantis labai didelio bagažo žinių. Taip pat jis turi būti nuolat tobulinamas, atnaujinamas. Kuo daugiau skenavimo metodų yra įgyvendinama, tuo daugiau potencialių pažeidimų jis gali aptikti. Bet problema tampa tai, kad kuo daugiau skenavimo metodų yra įgyvendinama, tuo daugiau

atnaujinimų ir priežiuros jis reikalauja tam, kad tie skenavimo metodai nepasentu ir netaptu nebeaktualus. Taip pat, tam, kad galima būtų pridėti naujų skenavimo metodu, reikia nuolatos sekti kibernetinės saugos naujienas ir ieškoti naujų ir geresnių būtų kaip automatizuoti tokius skenavimus. Pats automatizavimas yra sunkus, nes daugelis potencialių pažeidžiamumų yra dinamiainiai ir jų radima automatizuoti kartais tampa neįmanoma.

Rekomendacijos

- Prieš kuriant isitikinti, ar jus turite pakankamą bagaža žinių skirtų kibernetiniai saugai;
- Kurimas reikalauja didelio kiekio laiko, todėl tai daryti geriausia komandoje, kuri turėtų kibernetinio saugumo kompetencijos;
- Įrankio palaikymas reikalauja domėjimosi naujausiomis kibernetinės saugos aktualijomis, tad daug skaityti su tuo susijusio turinio;

Ateities darbų planas

Ateities darbų plano gairės:

- Įgyvendinti statinę analizę, parsisiuntus internetinės svetainės kodą, jį analizuoti priklausomai nuo kokia kalba jis parašytas;
- Įgyvendinti svėtainės dinaminę analizę, parsisiuntus internetinę svetainę, automatizuoti jos paleidimą ir paleidus įgyvendinti dinaminę analizę.
- Vidinių pažeidžiamumų skenavimą, prisijungti prie sistemos naudojans SSH, paleisti scriptus, kurie randa sistemos spragas.

Literatūros šaltiniai

[1] CVE-2019-0708. Available from MITRE, CVE-ID CVE-2019-0708., 2019.

[2] Muharrem Aksu, Enes Altuncu, and Kemal Bicakci. A first look at the usability of openvas vulnerability scanner. 02 2019.

[3] Bjorn Egil Asbjornslett. Assess the vulnerability of your production system. *Production Planning & Control*, 10(3):219–229, 1999.

[4] Ulrich Bayer, Andreas Moser, Christopher Kruegel, and Engin Kirda. Dynamic analysis of malicious code. *Journal in Computer Virology*, 2(1):67–77, 2006.

[5] Kenneth Brown. Opening the Open Source Debate A White Paper June 2002. 2002.

[6] Franco Callegati, Walter Cerroni, and Marco Ramilli. Man-in-the-middle attack to the https protocol. *IEEE Security & Privacy*, 7(1):78–81, 2009.

[7] Crispan Cowan, Calton Pu, Dave Maier, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, Qian Zhang, and Heather Hinton. Stackguard: Automatic adaptive detection and prevention of bu er-overflow attacks. In *USENIX Security Symposium*, volume 98, pages 63–78. San Antonio, TX, 1998.

[8] Crispin Cowan. Software Security for Open-Source Systems. *IEEE Security and Privacy*, 1(1):38–45, jan 2003.

[9] Roy Fielding, Jim Gettys, Je rey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–http/1.1, 1999.

[10] Richard Ford. Open vs. closed: which source is more secure? *Queue*, 5(1):32–38, 2007.

[11] Roshan Grewal. A hybrid approach of malware detection in android. 2017.

[12] Ron Gula. Passive vulnerability detection. *Network Security Wizards*, 9:7, 1999.

[13] Jaap-Henk Hoepman and Bart Jacobs. Increased security through open source. *arXiv preprint arXiv:0801.3924*, 2008.

[14] Paul R McWhirter, Kashif Kifayat, Qi Shi, and Bob Askwith. Sql injection attack classification through the feature extraction of sql query strings using a gap-weighted string subsequence kernel. *Journal of information security and applications*, 40:199–216, 2018.

[15] Andrew Meneely and Laurie Williams. Secure Open Source Collaboration: An Empirical Study of Linus' Law. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 453–462, New York, NY, USA, 2009. ACM.

[16] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[17] Birendra Mishra, Ashutosh Prasad, and Srinivasan Raghunathan. Quality and profits under open source versus closed source. *ICIS 2002 Proceedings*, page 32, 2002.

[18] Angela Orebaugh and Becky Pinkard. *Nmap in the Enterprise: Your Guide to Network Scanning*. Syngress Publishing, 2008.

[19] Jon Postel and Joyce Reynolds. File transfer protocol. 1985.

[20] Eric S Raymond and Tim O'Reilly. *The Cathedral and the Bazaar*. O'Reilly & Associates, Inc., USA, 1st edition, 1999.

[21] Russ Rogers. *Nessus network auditing*. Elsevier, 2011.

[22] Sapiah Sakri. Intrusion detection and prevention. 2004.

[23] Carla Sayan, Salim Hariri, and George L Ball. Semantic knowledge architecture for cyber security. In *Proceedings of the International Conference on Security and Management (SAM)*, pages 69–76. The Steering Committee of The World Congress in Computer Science, Computer …, 2019.

[24] Guido Schryen and Rouven Kadura. Open source vs. closed source software: towards measuring security. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 2016–2023. ACM, 2009.

[25] Peter Szor. *The Art of Computer Virus Research and Defense*. Addison-Wesley Professional, 2005.

[26] B Witten, C Landwehr, and M Caloyannides. Does open source improve system security? *IEEE Software*, 18(5):57–61, 2001.

[27] Tatu Ylonen and Chris Lonvick. The secure shell (ssh) protocol architecture. 2006.