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 perkėlimo protokolas, protokolas leidžiantis perkelti duomenis iš vienos sitemos į kitą[19].
- MITM - *Man in the middle* - „Žmogaus viduryje“ ataka – kibernetinės atakos tipas, kai puolėjas perima bendravimą tarp serverio ir kliento[6].
- Buffer overflow – viena iš potencialių rizikų, kai dėl per didelio kiekio duomenų informacija yra perrašoma gretimuose atminties blokuose[7].
- SSH - *Secure Shell* – tinklo protokolas, kuris leidžia vartotojui saugiai pasiekti sistemą per nesaugų tinklą [27].
- Framework - karkasas, į kurį įeina kelios (ar daugiau) programinės įrangos.
- NASL - *Nessus Attack Scripting Language* – paprasta kalba, kuri skirta aprašyti atskiroms grėsmėms ir galimoms atakoms.
- IDS - *Intrusion detection system* – sistema, skirta aktyviam saugumui, ji gali aptikti išpuolį realiu laiku[22].
- IPS - *Intrusion prevention system – IDS* papildymas, kuris gali aptikti įsilaužimą ir jį sustabdyti[22].
- HTTP - *HyperText Transfer Protocol* – informacijos priėmimo ir perdavimo protokolas[9].
- Daemon – programa, kuri veikia fone ir nėra valdoma vartotojo, bet laukia specifinio įvykio ar salygos suveikti.
- Slaptas įėjimas – kodo dalis, kuri leidžia atakuotojui į sistemą patekti nepastebėtam.
- Kenkėjiška programinė įranga – bendras pavadinimas tokiai programinei įrangai, kuri yra kenkėjiška ir patenka į sistemą be vartotojo leidimo[11].
- SQL injekcija – vienas iš kibernetinės atakos tipų, kai vartotojo įvestis internetinėje svetainėje yra sumanipuliuojama taip, kad ji padarytų daugiau nei buvo planuota, paveiktų duomenų bazę ir joje įgyvendintų užklausą[14].
- Fišingas – būdas skirtas pavogti privačią informaciją apsimetant tam tikru tiekėju.

Santrauka

Šiais laikais, kai pasaulis tampa vis labiau ir labiau skaitmenizuotas, iškyla opi problema – kibernetinė sauga. Todėl valstybės, įmonės ir korporacijos vis daugiau investuoja į kibernetinę saugą. Stengiamasi užkirsti kelią situacijoms, kurios atneštų žalą vartotojams, įmonėms ar net šalims. Kuriami įrankiai, kurie padeda apsisaugoti nuo tokių situacijų, analizuoja sistemas ir randa jų spragas.

Darbo tikslas – sukurti saugų pažeidžiamumų skenavimo įrankį, kuris saugiai skenuotų internetinę svetainę, jos sistemą bei jos failus ir pateiktų informaciją apie skenavimo rezultatus.

Darbe pristatotamas saugus pažeidžiamumų skaitytuvas, sistemų ir internetinių svetainių auditavimui. Taip pat pristatoma susijusių darbų analizė, kuri atskleidžia gerasias kitų skaitytuvų praktikas ir jų pritaikymą šiame skaitytuve. Be to, pristatomi pažeidžiamumų ir programinių klaidų analizės metodai, jų analizė ir panaudojimas skaitytuve. Taip pat pristatomos ir gerosios praktikos, kurių laikantis galima potencialiai sumažinti pažeidžiamumų skaičių sistemoje. Skaitytuvas įgyvendintas naudojant C# programavimo kalbą, .NET Core bei Angular karkasus, Docker konteinerių technologiją, Microsoft SQL duomenų bazę. Pažeidžiamumų skaitytuvas gali aptikti išorines sistemos spragas, kenkėjiškas programas internetinės svetainės failuose. Skaitytuvas geba nustatyti, ar internetinė svetainė saugi lankymuisi, jos atviras direktorijas, puslapius, prie kurių vartotojai turėtų neturėti teisės prieiti. Skaitytuvo saugumas yra užtikrinamas konteinerių technologija, kuri užtikrina, kad kenkėjiškos programos nepateks į skaitytuvo sistemą skenuojant internetinės svetainės failus, ar atliekant kitas skenavimo operacijas.

Summary

Secure Vulnerability Scanner for System Auditing

Nowadays, as the world becomes more and more digitized, cyber security is a major issue. As a result, states, businesses and corporations are increasingly investing in cybersecurity. Efforts are being made to prevent situations that could harm consumers, businesses or even countries. Tools are developed to help prevent such situations, analyze systems and identify gaps.

The purpose of this work is to create a secure vulnerability scanning tool that can safely scan the Internet site, its system and its files, and provide information about the scan results.

This paper introduces a secure vulnerability scanner for auditing systems and websites.. It also presents an analysis of related work that reveals best practices of other scanners and their implementation in this scanner. In addition, the methods of vulnerability and bug analysis, their analysis and implementation in the scanner are presented. The scanner was developed using C# programming language, .NET Core and Angular frameworks, Docker container technology, Microsoft SQL database. A vulnerability scanner can detect external system vulnerabilities and malicious software in web site files. It is also capable of detecting whether a website is safe to visit, its open directories or pages that users should not have access to. Scanner security is ensured by container technology, which ensures that malware does not enter the scanner system when scanning website files or performing other scanning operations.

Įvadas

Šiame darbe sukurtas saugus įrankis, kuris neišduoda savo buvimo vietos ir vartotojas pasinaudojęs juo gali gauti išsamią informaciją apie atitinkamą internetinę svetainę, joje egzistuojančius pažeidžiamumus ar modifikuotus failus, nerizikuodamas užkrėsti įrankio sistemos skenavimo metu. **Darbo tikslas** – sukurti saugų pažeidžiamumų skenavimo įrankį, kuris veiktų 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 pažeidžiamumus;
3. Išanalizuoti dažniausiai pasitaikančius internetinių svetainių skenavimo metodus;
4. Pateikti gerasias praktikas, kurios padėtų užtikrinti didesnį saugumą sistemai;
5. Sukurti įrankį, kuris saugiai skenuotų internetinę svetainę.

Kibernetinis saugumas yra vienas svarbiausių aspektų šių dienų technologijose. Kiekvienas asmuo, turintis išmanųjį įrenginį yra vienaip ar kitaip priklausomas nuo kibernetinio saugumo. Vienas pažeidžiamumas sistemoje gali lemti visos žmogaus asmeninės informacijos, esančios toje sistemoje ar įrenginyje, pasisavinimą. Internetinės svetainės taip pat neatsiejamos nuo kasdienio gyvenimo, 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. Dėl vieno pažeidžiamo, visa įmonės vartotojų (klientų) privati informacija gali būti pavogta ir panaudota kitiems tikslams. Tai gali sukelti didžiulius nuostolius įmonėms, jos gali visam laikui prarasti reputaciją, dėl kurios vartotojai pradėjo naudotis jų paslaugomis.

Kadangi kiekvieną sistemą kuria žmogus ir kiekvienoje sistemoje egzistuoja žmogiškasis faktorius, išlieka tikimybė, kad ši sistema turės pažeidžiamumų, kuriais gali pasinaudoti puolėjas. Dėl šios priežasties pažeidžiamumų skenavimo įrankis būtų ypač naudingas bet kokiai sistemai. Aptikus pažeidžiamumą, galima įtarti, kad tą patį pažeidžiamumą gali aptikti ir nuostolio įmonei ar pelno sau siekiantys asmenys. Taip pat galima įtarti, kad pažeidžiamumas jau buvo aptiktas ir tam tikri failai sistemoje buvo įdėti arba modifikuoti įdedant į juos tam tikrą kodą, kuris leistų atakuojančiam asmeniui patekti į sistemą nepastebėtam, sutrikdytų 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 aptikti automatizuotu įrankiu.
- Taip pat kiekvienai technologijai, kurią naudoja sistema, reikia atlikti atskirą analizę, kuri sumažina tokio įrankio efektyvumą ir ženkliai padidina kūrimo sudėtingumą;
- Sistemoje gali būti tiek daug skirtingų potencialių pažeidžiamumų, 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. Taip pat bandant aprėpti identifikuotas opiausias vietas ir naudojant skirtingus analizės bei skenavimo metodus, kurie turi didžiausius šansus rasti šiose vietose pažeidžiamumus. Tokiu būdu yra sumažinama tikimybė surasti visus pažeidžiamumus sistemoje, bet padidinama tikimybė aptikti daugiausiai pažeidžiamumų, taip pat sumažinant laiko bei žinių kaštus.

Šio įrankio kūrimo metu, siekiant sukurti saugų pažeidžiamumų aptikimo įrankį, buvo siekiama pagerinti sistemos saugumą, bet neužtikrinti jo, užtikrinant kuriamo įrankio sistemos saugumą. Įrankis yra skirtas tik internetinėms svetainėms, tačiau nėra stengiamasi jo pritaikyti kitoms sistemoms.

Pirmajame skyriuje analizuojami jau egzistuojantys pažeidžiamumų skenavimo įrankiai, kurie skenuoja pažeidžiamumus tam tikrose vietose, ar visoje sistemoje. Antrajame skyriuje aptariami skirtingi sistemų auditavimo metodai, paaiškinama, kam jie skirti, kokius pažeidžiamumus jie padeda aptikti. Trečiajame skyriuje pateikiamos gerosios praktikos, kokio tipo programinė įranga yra saugesnė, kokie metodai padeda sumažinti riziką atsirasti pažeidžiamumui sistemoje. Ketvirtajame skyriuje aprašomas įrankio kūrimo procesas, architektūra, naudotos technologijos, nepasisekę bei 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, įrankiais.

1.1. Nessus skaitytuvas

„Nessus“ įrankis yra tinklo pažeidžiamumų skaitytuvas, kurio architektūra leidžia lengvai susietų suderinamus kibernetinio saugumo įrankius[21]. „Nessus“ naudoja *NASL*.

„Nessus“ turi modulinę architektūrą, susidedančią iš serverio *daemon* atliekančio nuskaitymą ir nuotolinio 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, kurie 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 įvairaus (aukšto ir žemo) lygio interneto ir pramoninių protokolų skenavimą, 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 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 panašūs. Didžiausias jų skirtumas yra tas, kad „Nessus“ įrankis yra komercializuotas, o „OpenVAS“ – ne.

1.3. Nmap

„Nmap“ („Network Mapper“) yra nemokamas ir atvirojo kodo įrankis, kuris yra skirtas tinklo skenavimui. 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, kurie 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].

Įrankio skenavimo galimybės:

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

2. Pažeidžiamumų ir programinių klaidų analizės metodai

Programinės įrangos ar sistemų auditavimas apima platų spektrą metodikų, analizių. Sistemų auditavimas padeda surasti spragas prieš joms patenkant į galutinį produktą. Sistemų auditavimas padeda atrasti įsilaužimo paliktus pėdsakus – kenkėjiškų programų failus, paliktus potencialius slaptus įėjimus.

2.1. Statinė kodo analizė

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

Statinė analizė padeda aptikti:

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

2.2. Dinaminė kodo analizė

Dinaminė analizė vykdoma kai programa jau yra vykdymo stadijoje. Dinaminės analizės metu bandoma įgyvendinti visus įmanomus scenarijus ir išbandyti visas įmanomas įvesčių variacijas suvedant jas į programos įvestį. Dinaminę analizę galima taikyti modifikuotoms programoms, virusams ir kitiems paleidžiamiems projektams [4].

Veikimo metu programa gali neatlaisvinti atminties atgal į operacinę sistemą, dėl šios priežasties serveris, kuriame programa veikia, pritruks atminties ir pradės veikti lėčiau, kol galiausiai sustos. Nuo to padėtų apsisaugoti dinaminė analizė. Atlikus ją teisingai, galima aptikti didžiąją dalį spragų, kurios potencialiai labiausiai daro įtaką sistemai. Ištaisius spragas sistemos veikimo stabilumas padidėja, nenumatytų scenarijų skaičius sumažėja.

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:

- Didžiausią tiesioginę grėsmę sistemoje;
- Programinę įrangą, kuriai reikia atnaujinimų bei priežiūros;
- Atidarytus prievadus ir protokolus – įėjimo taškus į sistemos tinklą.

2.4. Vidinių pažeidžiamumų skenavimas

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

Vidinių pažeidžiamumų skenavimas aptiks:

- Sistemos komponentus, kurie galimai gali sukelti grėsmę;
- Pasenusią programinę įrangą, kuriai reikia atnaujinimų;

2.5. Oligomorfinių virusų skenavimas

Virusų kūrėjai 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 – įdiegti mutavusių iššifruoklių kurimo 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 kodo pagrindą, kuris duomenis kaupia kaip kodą[25].

3. Gerosios praktikos

Gerųjų praktikų pritaikymas sis temoje ar projekte gali padėti atrasti esamus pažeidžiamumus bei užtikrinti mažesnį skaičių būsimų pažeidžiamumų. Gerosios praktikos turi būti taikomos ne vieną kartą, o nuolatos ir 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 veikia su šia operacine sistema. Įvairios didžiosios įmonės ir vyriausybės vis labiau pripažįsta 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, kurimo metodus ir panašias temas. Atkreipiant dėmesį į saugumo problemas susijusias su atviro ir uždaro kodo metodika, kompiuterių saugumu. 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 prie sistemos saugumo daugiau nei uždaras išeities kodas? Šis klausimas sukelia daug diskusijų ir vieno aiškaus atsakymo niekada nebūna, dauguma specialistų sutinka su tokia nuomone, kad tiek uždaras, tiek atviras kodas turi savų pliusų ir minusų. Todėl galima teigti, kad paprasto atsakymo į šį klausimą nėra. Vienintelis sprendimas tokiai dilemai yra įsigilinti į abi šias metodikas ir išsiaiškinti, kuo viena metodika pranašesnė už kitą.

3.1.1. Atviro kodo programinė įranga

Argumentai prieš atvirą kodą:

- Atviras kodas suteikia didelį pranašumą atakuojančiam asmeniui dėl spragų radimo. Atakuojančiam asmeniui reikia surasti vieną spragą, su kuria jis galėtų sėkmingai užpulti sistemą, o programuotojams reikia ištaisyti visas spragas, kurios neleistų 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 peržiūrėjus kodą atidžiai, šios klaidos gali būti rastos ir ištaisytos, skirtingai nei 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 įėjimai ar kitas klaidinantis kodas, kuris iš pirmo žvilgsnio atlieka savo funkciją, bet įsigilinus išaiškėja, kad šie pataisymai yra skirti suteikti pranašumą puolėjui[26].

- Kodo uždarymas užkerta kelią atakuojančiam asmeniui lengvai gauti informaciją apie sistemą ir jos spragas, priešingai nei laikant kodą atvirą. Laikant kodą atvirą 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 peržiūrės kvalifikuoti specialistai, kurie suteiks savo įžvalgas[26].

- Atviro kodo projektai daug dažniau nustoja būti aktyvus nei 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 spragą[26].

Argumentai už atviro kodo programinę įrangą:

- Atidarant kodą visiems, yra lengviau identifikuoti problemas nei uždarant kodą. Atidarius kodą visiems, jį vertina ne tik kūrėjai, bet ir vartotojai bei kitos grupės žmonių, kurių dėka spragos yra pamatomos daug anksčiau lyginant su uždaro kodo produktais. Klaidas pastebėti yra žymiai lengviau dėl bendruomenės, išvengiama rizika, kad spraga bus nepastebėta ilgą laiką, kol ją išnaudos nuostolio siekiantys žmonės[20].

- Žmonės, naudojantys atviro kodo programinę įrangą, galės patys surasti ir sutvarkyti problemas kilusias su produktu. Tuomet jie gali savo pataisymus siūlyti į pagrindinę produkto repozitoriją, tokie pataisymai bus patvirtinti ir atsiras pačiame produkte, tuomet kiti žmonės galės parsisiųsti šiuos pakeitimus, 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 būna tikrinamas paviršutiniškai dėl laiko taupymo. Dėl šios priežasties kodo kokybė yra prasta ir didėja tikimybė atsirasti spragoms. Tačiau galima pastebėti, jog paviešintas kodas, kuris visada (arba ilgai) buvo uždaras dažnai būna ypatingai prastos kokybės ir iš to paviešinto kodo spragos būna išgaunamos labai greitai. Galima būtų teigti, jog viena iš priežasčių, kad parašytas kodas uždarame projekte patikrinimas ženkliai mažiau kartų nei atviro projekto kodas. Saugumo specialistai, analizuojantys tokį kodą, greitai atranda spragas ir paviešina įrankius, skirtus išnaudoti tokias spragas. Vienas pavyzdžių: 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 įrankiai skirti dekompiliuoti produktą tam, kad išgautų jame esantį kodą ir rastų spragas, palengvina darbą saugumo specialistams ir asmenims, kurie nori pasinaudoti išgautomis spragomis. Dažnais atvejais uždaro kodo produkto spragos būna paviešinamos, šios spragos būna užtaisomos atnaujinimais, bet kaikurios spragos būna rastos ir nepaviešintos tam, kad niekada jų nesutvarkytų ir atakuojantys asmenys galėtų ilgesnį laiką išnaudoti šias spragas savo tikslams. Iš to galima teigti, kad nors ir spragos būna lėčiau ir sunkiau surandamos uždarame produkte, tų spragų paviešinimas yra daug greitesnis nei atviro kodo produktuose[17].

- Atviro kodo produktai pasižymi savo bendruomene, dažnai prie atviro kodo produkto gali prisidėti visi. Todėl kodas būna tvarkomas daug greičiau, vartotojai patys randa problemas kode, informuoja apie tai kūrėjus, dažnai net pasiūlo savo sprendimus. Tokiais atvejais kūrėjams nebereikia patiems investuoti laiko sprendžiant problemą, užtenka tiesiog peržiūrėti 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 kūrėjams patarti produkto kurimo klausimais, ar padėti ištaisyti klaidas. Kūrėjai turi patys aiškintis tas klaidas ir dažnai tokių problemų sprendimas užtrunka ilgai, nes tam reikia 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 nei atviro kodo programinės įrangos spragas, kurias rasti yra neįtikėtinai lengviau, kurias rasti gali bet kas ir apie tai neprivalo pranešti. Net jeigu spraga yra ištaisoma greitai, nereiškia, kad tuo ištaisymu nėra padaroma kita spragą, kurią kiti asmenys vėl taip pat lengvai gali rasti ir išnaudoti. Kaip pavyzdį galima pateikti Microsoft Windows NT 4.0 produkto spragas. Prieš paviešinimą jos daugybę metų egzistavo, bet jų niekada 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 kūrėjai turi daugiau laiko ją pastebėti.[24].

3.1.3. Apibendrinimas

Kuri programinė įranga yra saugesnė: atvirojo ar uždarojo kodo? Galima teigti, kad didžiausias atviro kodo pliusas yra tai, kad vartotojas gali peržiūrėti programos kodą prieš ją naudodamas, skirtingai nei uždaro kodo programinė įranga, kuria vartotojas turi pasitikėti aklai[8]. Atviro kodo programinė įranga tiek užpuolikams, tiek gynėjams suteikia didesnę galimybę imtis veiksmų, vieninteliai klausimai: kas pirmas pastebės spragą, ką su ja darys?[17]. Uždaro kodo programinė įranga turiu pliusų žiūrint iš komercinės pusės, bet iš saugumo pusės sakymas, kad saugumą užtikrina kodo slėpimas, yra labai nepasitvirtinęs. Turint dabartines dekompiliavimo galimybes nebegalima 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 daug ir kuo daugiau jų taikoma, tuo daugiau naudos jos 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 programinių komponentų atnaujimas. Dažnas atvejis, kai senose operacinės sistemos, programinės įrangos versijose atrandama pažeidžiamumų, kuriais pasinaudojus įsilaužėliai 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 jas. Kaip pavyzdį galima pateikti aptiktą Microsoft Windows operacinėse sistemose buvusią spragą – „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, jog 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 svetainėse ar sistemose jų veikimo metu, svetainės 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 analizes dažnai atlieka programavimo aplinkos, kurios įspėja 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.

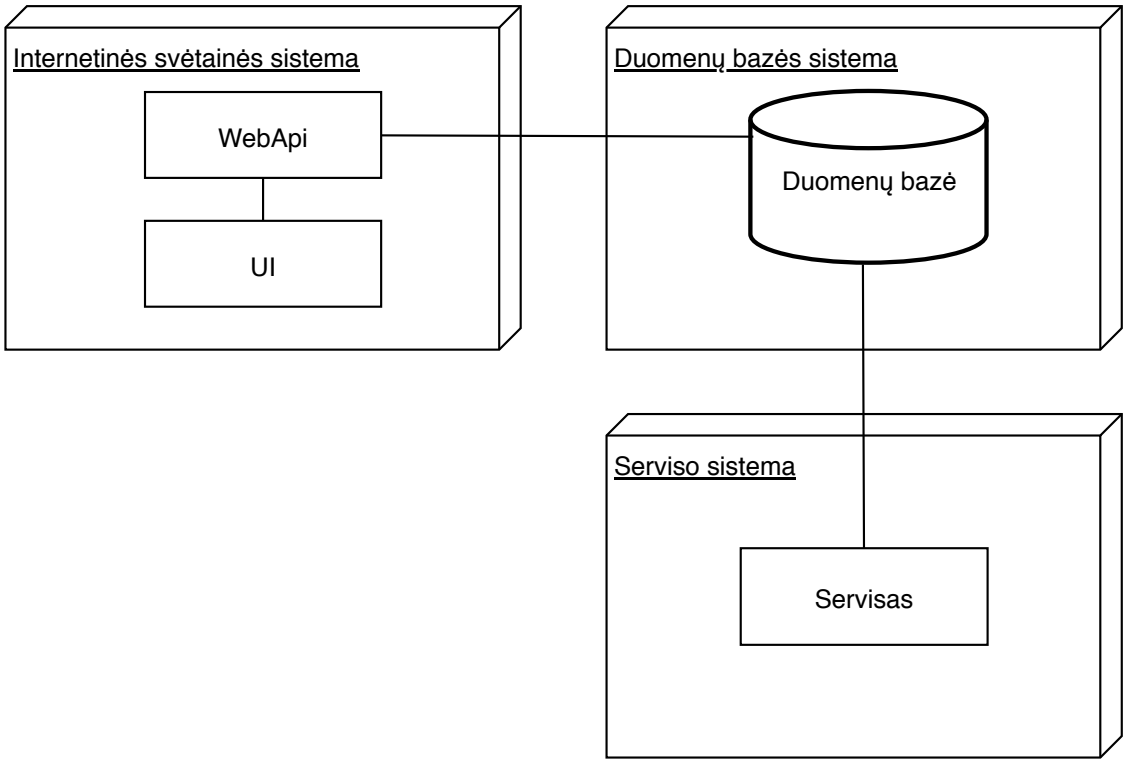
4. Sistemų auditavimo įrankis

Kurimo darbo metu buvo vadovaujamasi šiuo darbo tikslu: sukurti saugų pažeidžiamumų skenavimo įrankį, kuris veiktų 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:

- Sukurta saugi aplinka, į kurią galima būtų siųsti potencialiai užkrėtus failus;
- Išorinis sistemos skenavimas bandant išgauti kuo daugiau informacijos iš sistemos, ieškant atidarytų prievadų ar neveikenčių sistemų skenuojamoje sistemoje;
- Tikrinamas pats svetainės adresas, ar yra saugu eiti į jį;
- Parsisiųsti failai iš nurodyto *FTP* serverio yra tikrinami. Tikrinama ar jų turiniai nėra potencialiai infekuoti ir keliantys 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 nedaro įtakos kitai sistemai. Taip pat, jeigu įvyktų įsilaužimas į viena iš sistemų, įsilaužėliai neturėtų prieigos prie viso projekto.



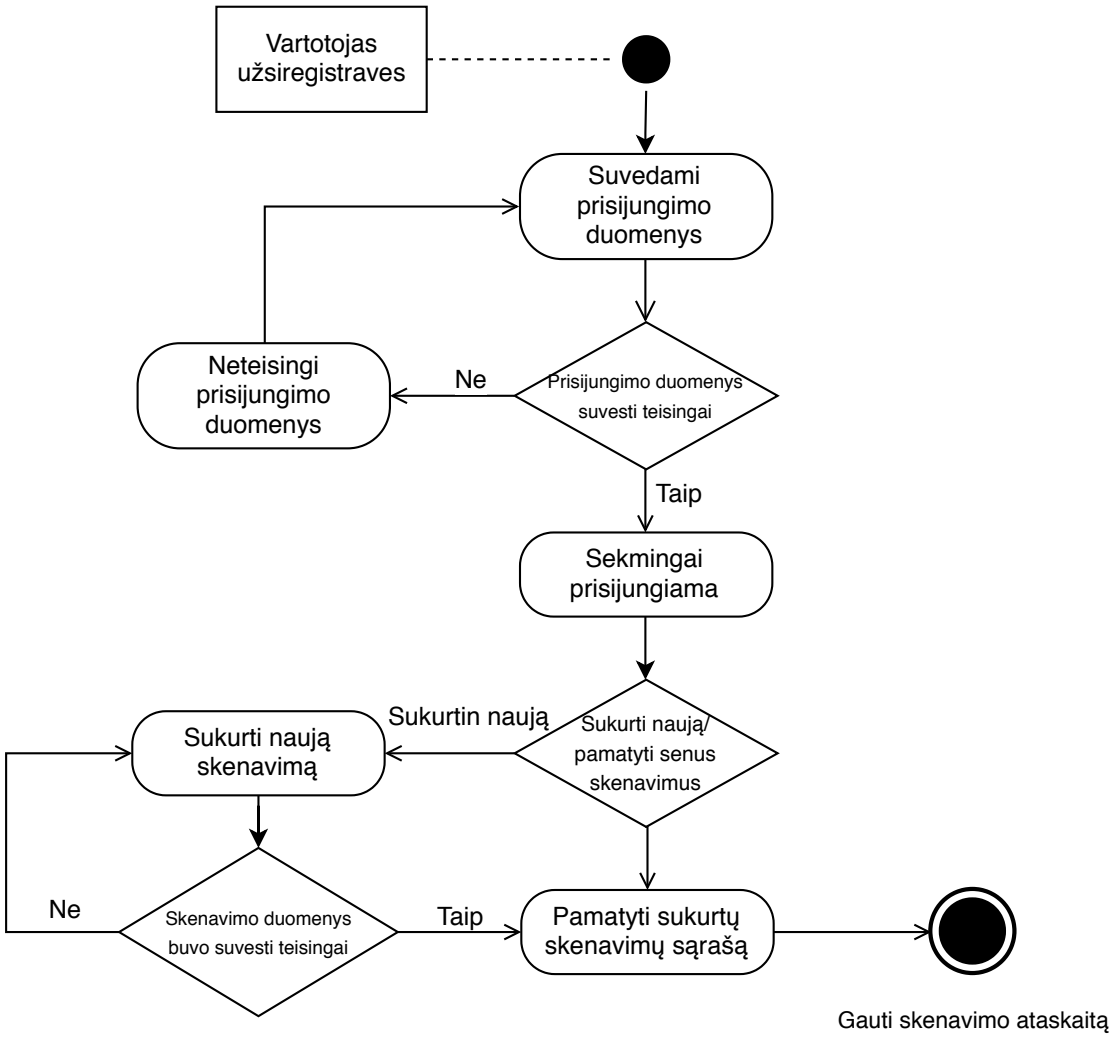
1 pav. Sistemos architektūros schema

Sistemos architektūros schemos 1 pavyzdyje matome tris sistemas – internetinės svetainės sistemą, duomenų bazės sistemą ir serviso sistemą. Internetinės svetainės sistemoje veikia pati inter-

netinė svetainė, kurioje vyksta prisijungimas prie sistemos, skenavimo užklausų kurimas ir skenavimo ataskaitų parsisiuntimas. Duomenų bazės sistemoje veikia pati duomenų bazė, kurios paskirtis yra laikyti skenavimo užklausas ir jų rezultatus, taip pat laikyti prisijungo duomenis. Serviso sistemoje veikia pats servisas, kuris atlieka visa skenavimo logiką ir visą bendravimą su skenuojama internetine svetaine. Taip pat bendrauja ir su duomenų baze, iš jos pasiima visus duomenis reikalingus skenavimui ir į ją deda visus skenavimo rezultatus.

UI aplikacija internės svetainės sistemoje sukurta naudojant Angular. WebApi, kuris atsakingas už visą bendravimą su duomenų baze bei skenavimo ataskaitų formavimą, 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



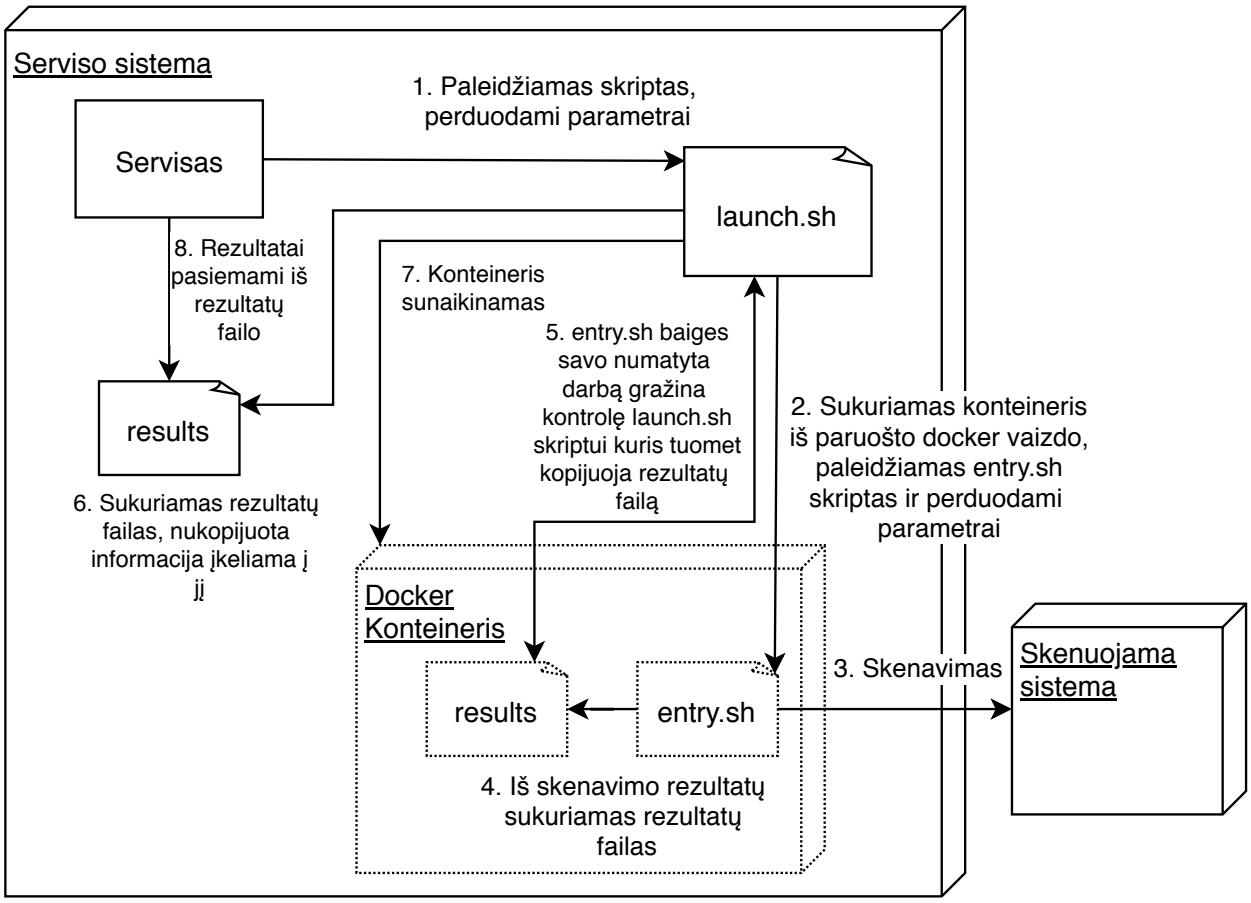
2 pav. Aktyvumo diagrama

Aktyvumo diagramoje, kuri yra 2 pavyzdyje, galima matyti visus pasirinkimus, kuriuos 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, kuriame prisijungimo duomenys yra verifikuojami su duomenų baze. Sėkmingai prisijungus, vartotojas gali arba kurti naują skenavimo užklausą, arba pamatyti jau sukurtas. Kuriant skenavimo užklausą, duomenys taip pat yra tikrinami, patikrinus ir sėkmingai užregistravus skenavimo užklausą, vartotojas gali eiti į skenavimų sąrašą, kur bus pateiktas jo sukurtas skenavimas, ataskaitų parsisiuntimo nuorodos, arba jis gali kurti naują užklausą.

4.3. Saugios aplinkos užtikrinimas

Internetinių svetainių skenavimo įrankyje saugi aplinka užtikrinama naudojant Docker, kuris pasižymi tuo, kad su ja yra kuriami konteineriai, kurie yra įzoliuoti nuo likusios sistemos. Pačią Docker technologiją galima lyginti su virtualiomis mašinomis, kurias pavyzdžiui kuria VirtualBox įrankis. Skirtumas tarp jų 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 tą pačią opericinę sitemą, 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 atskirs konteineris, po kiekvieno proceso įgyvendinimo konteineris yra sunaikinamas. Konteineriu greitis leidžia tai padaryti greitai ir saugiai, nėra priežasties, kodėl konteinerius reikėtų pernaudoti, nereikia atskirai surašinėti operacinių sistemų ir jų konfiguruoti, taip pat mažas resursų kiekis nestabdo visos sistemos bendrai ir užtikrina, kad sistema 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, kuris yra sukuriamas vieną kartą, ir visą laiką laikomas išsaugotas. Kurdamas konteinerį skriptas taip pat į konteinerį perduoda bash komandą, kuri paleidžia „entry.sh“ skriptą esanti 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 parsisiųsti 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 konteinerį su visais jame esančiai failais ir baigia savo darbą. Jeigu konteineryje buvo atsiųsta failų iš skenuojamos sistemos, jie taip pat yra sunaikinami.

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

4.4. Įgyvendinami skenavimo metodai

Iš viso yra įgyvendinti keturi skenavimo metodai. Šie metodai aprėpia 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 esantys pažeidžiamumai yra aptinkami prieš įsilaužimą, bet apima ir įsilaužimo ir po įsilaužimo situacijas.

4.4.1. Išorinis sistemos skenavimas

Išorinis sistemos skenavimas skirtas tam, kad aptiktų atvirius prievadus, patikrintų, kokią informaciją 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 anoniminiai 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 patikrinti ar į sistemą jau nebuvo įsilaužta, ir joje nėra paliktų 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 konteinerį, 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 svetainės prieigos skenavimas

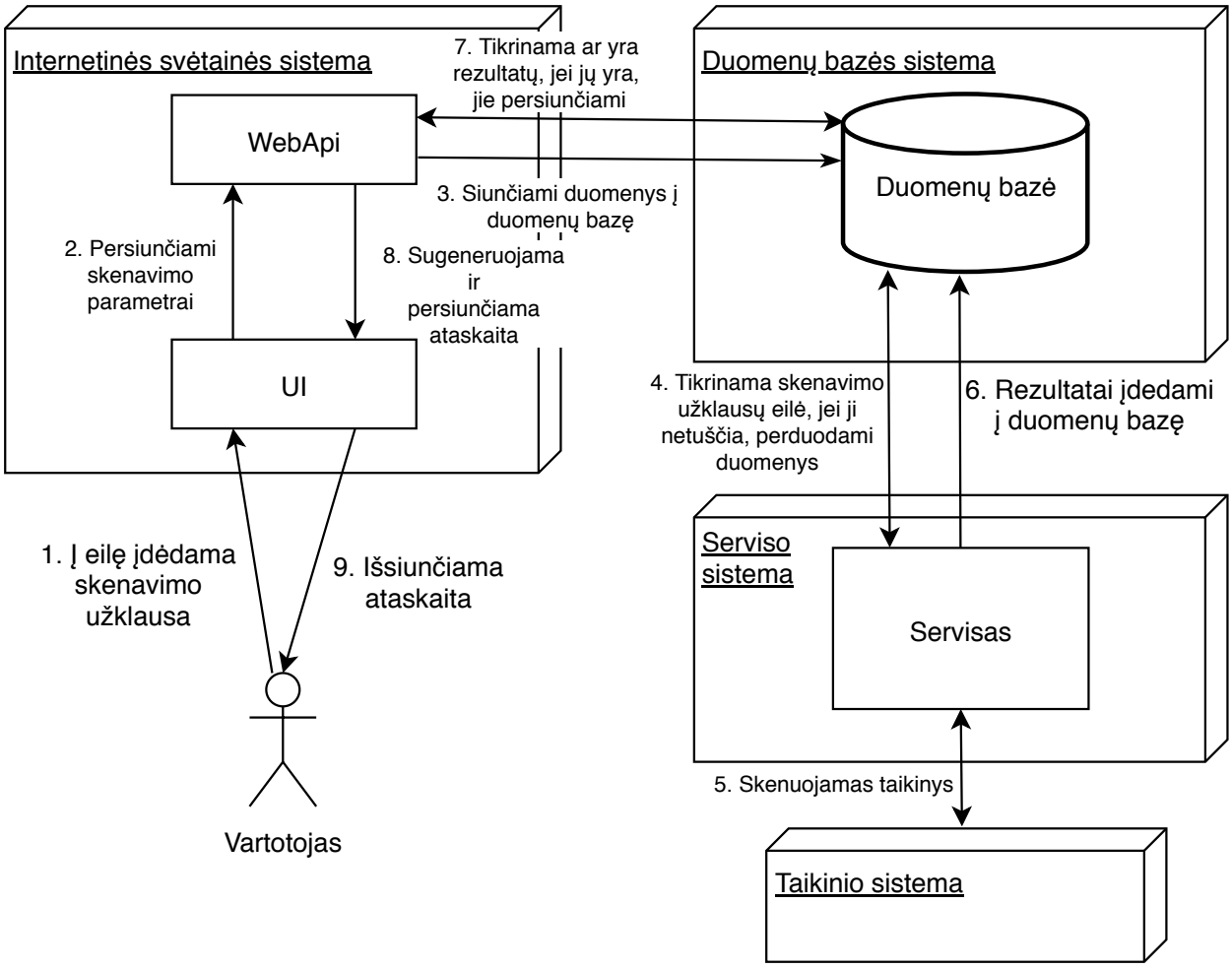
Internetinės svetainės prieigos skenavimas yra skirtas tam, kad aptiktų potencialias *MITM* atakas, fišingo svetaines, kenkėjiškų programų svetaines. Veikimas yra toks, kad svetainės adresas yra nusiunčiamas trečiajai šaliai, kuri svetainės adresą skenuoja su įvairiomis antivirusinėmis. Tuomet yra pateikiama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

4.4.4. Internetinės svetainės enumeravimas

Internetinės svetainės enumeravimas yra skirtas tam, kad aptiktų potencialias internetinės svetainės spragas, tokias kaip atviras direktorijas, kuriose puolėjas gali be jokių kliūčių 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 procesų, kurie veikia paeiliui arba paraleliai. Kiekvieno proceso paaiškinamas paeiliui:

1. Prisijungęs 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 bazę, ir yra išsaugomi;

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

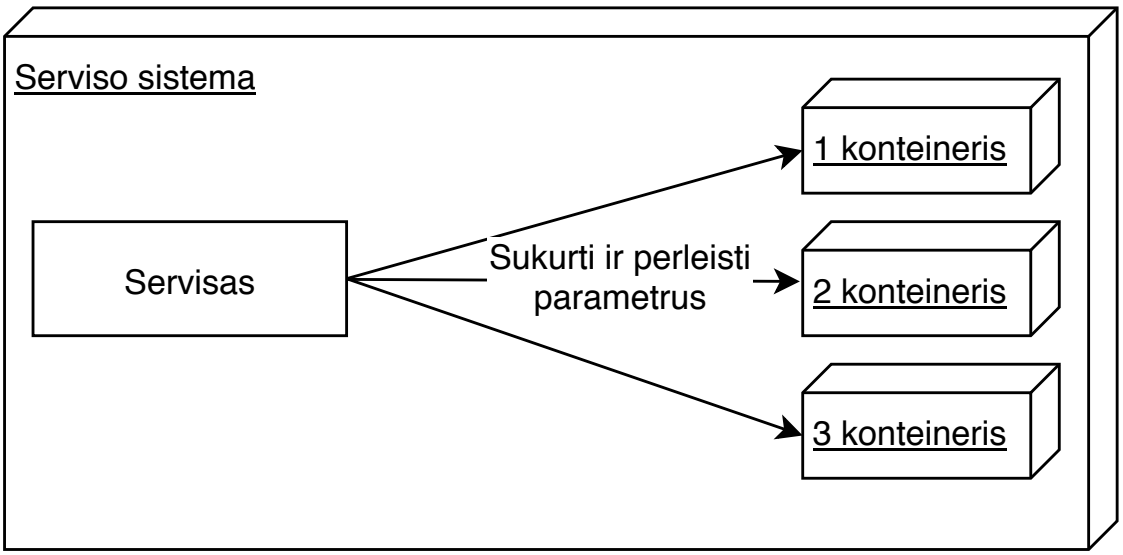
5. Pradedama vykdyti skenavimo logika kiekvienai užklausai paeiliui. Užklausos vykdomos paeiliui, o ne iš karto visos tam, kad būtų užtikrintas sistemos stabilumas ir minimalus resursų naudojimas. Jeigu vienu metu būtų įgyvendinamos šimtai užklausų, rizikuojama, kad sistema pritrūks resursų visoms operacijoms įgyvendinti. Nors ir kiekviena užklausa įgyvendinama paeiliui, bet kiekvienai užklaisai skenavimo operacijos yra vykdomomos paraleliai. Šiuo atveju nėra rizikuojama, kad sistema pritrūks resursų, nes yra 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 kartą, kai vartotojas perkrauna puslapį, kuriame yra skenavimo užklausų sąrašas, gavus atsaką, kad duomenų yra, vartotojui leidžiama atsisiųsti ataskaitą;

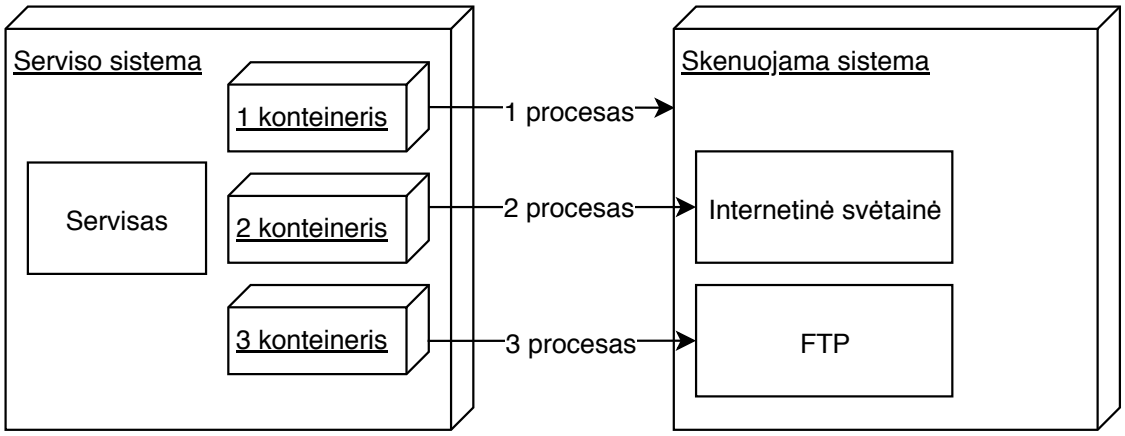
8. Vartotojas pateikia užklausą gauti skenavimo užklausos ataskaitą. Tuo metu paimami duomenys iš duomenų bazės ir yra generuojama ataskaita HTML formatu. Sugeneruojamas parsisiuntimo adresas.

9. Vartotojui leidžiama parsisiųsti ataskaitą.



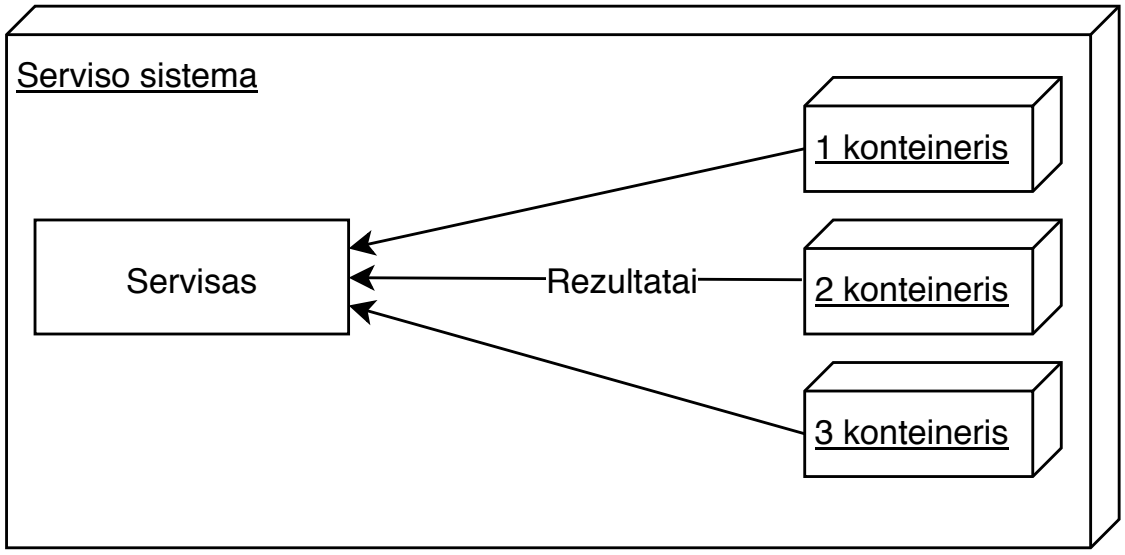
5 pav. Konteinerių sukurimas įgyvendinant skenavimo užklausas

Serviso sistemoje įgyvendinant skenavimo užklausas yra kuriami trys docker konteineriai kaip pavaizduota 5 pavyzdyje. Apie patį konteinerių kurimo procesą 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 pavyzdyje ir jame matome, kad visi konteineriai atlieka skirtingus procesus paraleliai, 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 svetainės enumeraciją, kuri aprašyta 4.4.4, šiam konteineriui kaip parametras yra paduodamas internetinės svetainės 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 skenuojamos sistemos prieiga, 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 svetainės prieigos skenavimo rezultatai yra pasiimami ir perduodami servisui, servisas vėliau juos formatuoja ir talpina į duomenų bazę. Pats duomenų perdavimas ir konteinerių 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 kokiame scenarijuje. Šis įrankis yra plačiai naudojamas kibernetinio saugumo bendruomeneje ir taip pat yra ypač efektyvus atliekant tinklo skenavimus ar analizę. Atsisžvelgus į šiuos faktus, šis įrankis tampa butinybė bet kokioje spragų skenavimo 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**
    $ThreadPool \leftarrow Scan1(ScanRequest)$
    $ThreadPool \leftarrow Scan2(ScanRequest)$
    $ThreadPool \leftarrow Scan3(ScanRequest)$
    $ThreadPool \leftarrow Scan4(ScanRequest)$
    $ThreadPool.WaitForAll$
    $Database \leftarrow ThreadPool.Results$
  end for
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 iš naujo. Šios platformos kodą galime matyti 1 algoritme. Iš pradžių pasiimame visas skenavimo užklausas, kurios dar nebuvo vykdytos iš duomenų bazės, tuomet iteruojame per kiekviena iš jų ir pararelіai paleidžiame visus skenavimo metodus. Kai visi skenavimo metodai yra paleisti, laukiame, kol visi jie pasibaigs. Visiems skenavimamo metodams 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 atradęs šią direktoriją gali be jokių kliūčių peržiūrėti joje esančius failus;
- Aptiktini puslapiai, prie kurių neprisijungęs vartotojas neturėtų 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 svetainė, kuri leidžia vartotojui kurti skenavimo užklausas ir po jų sukurimo leidžia atsisiųsti suformatuotus rezultatus;
- Sukurta platforma ir paruoštukai, kurie leidžia lengvai pridėti naujus skenavimo metodus į įrankį;
- Sukurta saugi aplinka, kuri leidžia vartotojui saugiai skenuoti sistemas ir jų failus, nesibaiminant infekuoti pačio įrankio sistemos;
- Aptinkami pažeidžiamumai ir pati sistema paruošta naudojimui;
- Sugeneruojama ir pateikiama ataskaika, kurią lengva suprasti vartotojui.

4.9. Trūkumai

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

Kurimo metu buvo planuota panaudoti trečios šalies įrankį SqlMap. Šio įrankio paskirtis yra ieškoti ar internetinė svetainė yra pažeidžiama SQL injekcijos atakoms. Tokio tipo skenavimui reikėtų sukurti internetinės svetaines puslapių kodo funkciją, kuri svetainės puslapyje ieško dinaminių nuorodų. Šios nuorodos priema parametrus savo nuorodose. Tuomet SqlMap įrankis bando į šiuos parametrus įdėti 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 svetainėje. Dėl šios priežasties SqlMap funkcionalumo atsisakyta.

Kurimo metu taip pat buvo užsibrėžta įgyvendinti statinę analizę. Šis tikslas buvo įgyvendintas nepilnai, patys failai buvo tikrinami žinomų kenkėjiškų programų duomenų bazėje, bet internetinės svetainės kodas nebuvo tikrinamas. Šio tikslo buvo atsisakyta. Šio atsisakymo priežastis yra ta, kad kiekvienai programavimo kalbai reikalingas atskiras statinės analizės įgyvendinimas, kieviena kalba yra kitokia, jos funkcionalumas ir sintaksė pat pat, dėl šios priežasties tokio funkcionalumo kurimo kaštai stipriai didėja.

4.10. Darbo rezultatų apibendrinimas

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 papildomą 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, jų 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 parsisiųsti visus failus ęsančius joje ir tikrinti ar jie egzistuoja žinomų kenkėjiškų programų duomenų bazėje. Skaitytuvas gali tikrinti internetinę svetainę, kuri veikia pasirinktoje sistemoje, tikrinama ar ši svetainė turi direktorijų, kurios pateikia savo turinio sąrašą, tokiose direktorijose failų turinį gali skaityti bet kas, taip randant neteisingai sudeliotas teises sistemoje. Taip pat ieškoma ir puslapių sąrašo, kurį neautentifikuotas vartotojas gali pasiekti, taip bandoma surasti puslapius, prie kurių vartotojas gali prieiti, bandoma surasti, ar yra prieigos taškų, kurie neturėtu būti prieinami kiekvienam. Taip pat yra tikrinama ir pati prieiga prie internetinės svetainės, bandoma rasti ar nėra vykdoma MITM ataka ir ar internetinėje svetainėje neveikia kenkėjiškos programos.

Skaitytuvo saugumas yra užtikrinamas naudojant docker konteinerius. Parašyti specialiūs paruoštukai kiekvienam skenavimui, kurie leidžia sukurti specifinį konteinerį. Naudojant konteinerių technologiją yra pasiekiamas saugumas, kuris apsaugo skaitytuvo sistemą nuo potencialių grėsmių, kurios gali slėptis internetinėje svetainėje. Kiekvieno skenavimo užklausa vykdoma paraleliai paleižiant skenavimus, o tai 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. Svetainėje yra autentifikacija, kuri reikalinga kuriant naujas skenavimo užklausas. Svetainėje galima kurti naujas skenavimo užklausas, peržiurėti visas kitas, ir parsisiųsti jų visų ataskaitas.

Išvados ir rekomendacijos

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, kuriais 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 aktualūs ir turintys didžiulę naudą. Norint pasiekti maksimalų sistemos saugumą, sistemą tenka audituoti dažnai ir be automatizuotų skaitytuvų, toks darbas taptų 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 iškyla ta, kad kuo daugiau skenavimo metodų yra įgyvendinama, tuo daugiau atnaujinimų ir priežiuros pats skaitytuvas reikalauja tam, kad tie skenavimo metodai nepasentų ir netaptų nebeaktualūs. Taip pat tam, kad galima būtų pridėti naujų skenavimo metodų, reikia nuolatos sekti kibernetinės saugos naujienas ir ieškoti naujų, geresnių būdų kaip automatizuoti tokius skenavimus. Pats automatizavimas yra sunkus, nes daugelis potencialių pažeidžiamumų yra dinaminiai ir jų radimą automatizuoti kartais tampa neįmanoma.

Rekomendacijos

- Vietoje .NET Core ateityje naudoti Python, nes Python kodo rašymas yra paprastesnis, lyginant su .NET Core. Kadangi servisas yra salyginai mažo dydžio programa, kurios pagrindinis darbas yra bash komandų vykdymas ir duomenų manipuliavimas, Python programavimo kalba padėtų įgyvendinti šias užduotis lengvesniu ir paprastesniu būdu;

- Skenavimo įrankio internetinės svetainės sistemą skelti į dvi sistemas – vienoje sistemoje yra visa vartotojo sąsaja, kuri bendrauja tarp WebApi ir kliento, kita sistema skirta vien tik WebApi, kuris bendrauja su vartotojo sąsaja ir duomenų baze. Taip yra užtikrinamas papildomas saugumo ir stabilumo sluoksnis pačiam įrankiui. Puolėjas, turi laužtis net į kelias sistemas vien tam, kad pasiektų duomenų bazę. Vartotojo sąsajoje nelaikoma jokia veikimo logika, o vien tik kiekvienam vartotojui prieinama informacija;

- Vietoje Microsoft SQL naudoti PostgreSQL, nes PostgreSQL yra daug lengviau sukonfiguruoti sistemoje, taip pats bendravimas tarp kodo ir sistemos tampa lengvesnis, jei kartu naudojama ir Python. Naudojant .NET core bendravimo su duomenų baze įgyvendinimas tampa lengvesnis naudojant Microsoft SQL. Dar vienas didelis pliusas naudojant PostgreSQL yra tai, kad jį galima instaliuoti ir sukonfiguruoti daugumoje operacinių sistemų, priešingai nei Microsoft SQL, kuris santykinai neilgai palaiko Linux operacines sistemas ir tik ribotą jų skaičių;

Ateities darbų planas

Ateities darbų plano gairės:

- Įgyvendinti statinę analizę, kuri parsisiunčia internetinės svetainės kodą, jį analizuoti priklausomai nuo to, kokia kalba jis parašytas;
- Įgyvendinti svetainės dinaminę analizę, kuri parsisiuntus internetinę svetainę ją paleiždia lokaliai ir įgyvendina dinaminę analizę;
- Įgyvendinti vidinių pažeidžiamumų skenavimą, kuris prisijungia prie sistemos naudojant SSH ir paleidžia skriptus, kurie randa sistemos spragas;
- Įgyvendinti skenavimą, kuris aptiktų svetainėje spragas, kurios leistų įgyvendinti SQL injekcijos atakas.

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.