VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
MATEMATINĖS INFORMATIKOS KATEDRA

Biologinių aplikacijų kūrimas su „Django" karkasu

Development of biological web applications using „Django" framework

Kursinio darbo projektas

Atliko: 3 kurso Bioinformatikos studentė
Ugnė Antanaitytė

Darbo vadovas: lekt. Irus Grinis

Vilnius 2023

TURINYS

Įvadas

Žiniatinklio programa arba dar kitaip vadinama aplikacija yra programinė įranga, kurią galima pasiekti naudojant interneto naršyklę [MAr23]. Tokių aplikacijų kūrimas ir pritaikymas apima labai daug sričių, jos vartotojai gali būti tiek įvairios organizacijos, tiek asmenys, naudojantys programą individualioms reikmėms. Biologinė aplikacija tai yra žiniatinklio programa, kurios paskirtis dažniausiai būna kokio nors algoritmo taikymas biologinių duomenų analizavimui, modifikavimui ar vertinimui.

Palyginus su darbalaukio programomis, aplikacijos yra lengviau palaikomos, sukelia mažiau suderinamumo problemų, bei yra lengviau prieinamos ir universalesnes. Nepaisant šių privalumų kūrėjams gali iškilti įvairių iššūkių programos vystymo metu. Programuotojai, norėdami paprasčiau ir efektyviau kurti aplikacijas, tam pasitelkia karkasus – įrankių rinkinius, kurie leidžia palaikyti tvarkingą ir vientisą programavimo architektūrą integruojant naują funkcionalumą. Vienas iš tokių įrankių yra „Django" - aukšto lygio „Python" žiniatinklio karkasas, leidžiantis kurti biologines aplikacijas [Fou21].

Anksčiau vystytame projekte buvo išnagrinėti „Django" karkaso veikimo principai ir juo kuriamos programos dizaino architektūra. Praktinei darbo daliai sukurta biologinė žiniatinklio programa, kuri suteikia galimybę prisijungusiems vartotojams naudotis terminalo emuliatoriumi tam, kad galėtų konstruoti „Unix" apvalkalo komandų ir E-įrankių konvejerius. Aplikacijoje aptikta saugumo spraga – didelis pažeidžiamumos komandų injekcijos atakoms, kuris atsirado dėl netinkamo vartotojo įvesties apdorojimo ir suteiktų šeimininko privilegijų vartotojams. Tokia saugumo spraga yra laikoma esminiu trūkumų bet kokioje programoje, todėl užduotis pašalinti šią problemą turėtų būti laikoma prioritetu.

Šio darbo tikslas yra detaliau panagrinėti komandų injekcijos atakas, prevencijos būdus, bei praktiškai pritaikyti žinias toliau vystant biologinę žiniatinklio aplikaciją su „Django" karkasu.

Uždaviniai

Siekiant išspręsti problemą, susijusią su komandų injekcijos atakų pažeidžiamumu, sukurtoje aplikacijoje ir toliau vystant papildomą programos funkcionalumą, išsikelti šie uždaviniai:

1. Apibrėžti, kas yra komandų injekcijos atakos ir kokie yra jų tipai.
2. Išnagrinėti literatūrą, aprašančią komandų injekcijos reikšmę ir padarinius, bei pateikti tokios atakos ir jos įtakos pavyzdžių.
3. Aptarti kaip galima aptikti programoje pažeidžiamumą komandų injekcijoms ir kokių atsakomųjų veiksmų galima imtis, kad jų išvengti.
4. Aprašyti „EMBOSS“ įrankių reikšmę ir panaudojimą bioinformatikoje.
5. Panaudoti „commix“ įrankį automatiniam komandų injekcijų pažeidžiamumo aptikimui realizuotoje biologinėje aplikacijoje.
6. Remiantis įgytomis žiniomis nagrinėjant literatūrą, pabandyti išspręsti pažeidžiamumo komandų injekcijoms atakos problemą.
7. Implementuoti galimybę biologinėje aplikacijoje vartotojams naudotis „EMBOSS“ įrankiais, bei kurti aplankalus.

1. Kodo injekcijos

Kodo injekcija yra bendras terminas apimantis atakas, kurios pasireiškia kai pažeidžiamos programos vykdo tam tikrą vartotojo pateiktą įvesties kodą. Tokio tipo atakos laikomos itin pavojingomis ir netgi 2017 metais „OWASP" – standartizuotame dokumente apie žiniatinklio programų pažeidžiamumus, buvo pateiktos, kaip atakos, sukeliančios programų pažeidimus dažniausiai (2021 metais pateiktos trečioje vietoje) [fou22]. Tokio tipo pažeidžiamumai atsiranda dėl netinkamo arba nepakankamo duomenų apdorojimo, kuomet vartotojas įterpia savavališką kodą arba komandas, kurių vykdymas yra nesuplanuotas. Tokiom atakom priklauso: komandų injekcijos, „SQL" injekcijos, kryžminis skriptas, „LDAP" ir „XPath" injekcijos. Šiame darbe detaliau nagrinėjamos komandų injekcijos atakos. Pagal „OWASP" dokumentą tokia injekcija apibrėžiama, kaip ataka, kurios tikslas savavališko kodo vykdymas šeimininko (*angl. host*) operacinėje sistemoje per pažeidžiamą programą. Tokių atakų metu, pateiktos komandos dažniausiai vykdomos su pažeidžiamos komandos privilegijomis ir yra nepilnai apdorojamos, patikrinamos. Tokios komandų injekcijos atakos yra nepriklausomos nuo operacinės sistemos, programavimo kalbos ar tinklapiui kurti skirto programavimo karkaso [Ana14].

1.1. Padariniai ir reikšmė

Komandų injekcijos atakos gali turėti ypatingai neigiamų padarinių sistemos vartotojams ar jos tiekėjams, tai galėtų būti: konfidencialių duomenų praradimas, sistemos funkciniai pažeidimai, neteisėta prieiga prie šeimininko sistemos ir t.t. Pavyzdžiui, užpuolikas neteisėtai galėtų prieiti prie jam neleistinų duomenų, slaptažodžių ar sistemos failų, suradęs būdą pasiekti šiuos duomenis, užpuolikas galėtų juos ištrinti, nutekinti arba modifikuoti jam palankiu būdu.

Vienas iš geriausiai žinomų šios injekcijos pažeidžiamumo pavyzdys ko gero 2014 metais aptikta „Shellshock" klaida [Ant14]. Tai populiarios programos „Bash" pažeidžiamumas, kuris buvo beveik visuose „Linux" sistema paremtuose kompiuteriuose ir įrenginiuose. Šis pažeidžiamumas, leidžia užpuolikui naudoti aplinkos kintamuosius, kaip funkcijas, ir taip lengvai paleisti tas funkcijas tiesiogiai šeimininko sistemoje. Šis pažeidžiamumas tuo metu galimai paveikė apie pusę interneto svetainių ir dar begalę su internetu susietų įrenginių. Dar ir šiuo metu, nepaisant žiniasklaidos įspėjimų ir valstybės saugumo departamento įspėjimų, kai kurios itin svarbios sistemos

lieka nepataisytos. Šis pavyzdys parodo, kokią didelę įtaką gali turėti tokio tipo pažeidžiamumai visuotinai naudojamose programose ir kaip svarbu taikyti prevenciją tokioms atakoms.

1.2. Rezultatais paremtos komandų injekcijos

Rezultatais paremtų komandų injekcijų metu užpuolikas gali tiesiogiai numatyti įvestos komandos per aplikaciją rezultatą. Tokio tipo injekcijos gali būti skirstomos šiuo būdu:

1. Klasikinė rezultatais paremta komandų injekcija – paprasčiausia ir dažniausiai atliekama injekcija, kurios metu užpuolikas pasinaudodamas tam tikrais operatoriais konkatenuoja tikrąjį kodą su injekcijos kodu arba pašalina tikrąjį kodą ir palieką vykdyti tik savo pridėtą.

2. Dinaminio kodo vertinimo metodas – taikomas pasinaudojant funkcija „eval“, kuri naudojama dinamiškam kodo apdorojimui kodo vykdymo metu interpretuojant įvestą skaitinę reikšmę kaip kodą.

```
compute = input('\nExpression=>')
if compute:
    print(eval(compute))
```

1.3. Nepastebimos komandų injekcijos

Vykdant nepastebimas komandų injekcijos atakas atakuojama aplikacija negrąžina jokio rezultato užpuolikui, todėl užpuolikas netiesioginiais metodais daro išvadas apie išvestį. Tokie metodai gali būti skirstomi tokiu būdu:

1. Laiku paremtas metodas – metodas, kuomet vykdant kodą įtraukiami laiko uždelsimai ir jais remiantis, sprendžiama ar kodas įvykdytas sėkmingai.

2. Failais paremtas metodas – metodas, kuris veikia tokiu pačiu principu kaip ir klasikinis, tačiau kadangi užpuolikas negali matyti kodo komandos išvesties, ji yra nukreipiama į failą. Tam, kad tokia ataka būtų vykdoma užpuolikui turi būti suteiktos rašymo teisės, jeigu jų ataką atliekantis asmuo neturi, išvestis gali būti rašoma į laikinuosius failus, o šie gali būti perskaitomi laiku paremtu komandos injekcijos metodu.

1.4. Komandų injekcijos aptikimas

Nors ir komandų injekcijos atakos yra ganėtinai paplitusios ir laikomos itin pavojingomis, jų aptikimas ir prevencija nėra taip plačiai nagrinėjama kaip SQL injekcijų ar kryžminio skripto atakų. Vienas iš siūlomų įrankių, kuris padeda automatizuoti komandų injekcijų aptikimą žiniatinklio programose, yra „commix“ (*angl. Command Injection Exploitation*) [Sta22]. Šis įrankis skirtas testavimui ir saugumo spragų paieškoms aplikacijose. „Commix“ veikia kaip įvestį nuskaitydamas svetainės ar aplikacijos URL adresą su GET ar POST užklausų parametrais, toliau įvykdžius užklausą, jos rezultatai analizuojami – tikrinama ar nėra saugumo spragų.

Šis įrankis susideda iš trijų dalių: atakos generavimo modulio, kuris generuoja komandų injekcijos atakas kiekvienam atakos tipui, pažeidžiamumo aptikimo variklio, kuris atlieka atakas su parinktais HTTP parametrais ir eksplotacijos modulio, kuris toliau išnaudoja aptiktą pažeidžiamumą ir apie tai praneša vartotojui.

1.5. Atsakomosios priemonės

Aptikus pažeidžiamumą komandų injekcijoms aplikacijoje būtina imtis atsakomųjų priemonių. Svarbiausios prevencinės priemonės šių atakų užkirtimui yra įvesties patikrinimas ir perfiltravimas, pašalinat pavojingus simbolius, bei išvesties pašalinimas, kuomet nesuteikiama vartotojui jokios informacijos apie sistemoje vykstančius procesus. Taip pat svarbu programų kūrėjams itin atsižvelgti į įvesties panaudojimą funkcijose, kurios interpretuoja skaitinę įvestį kaip kodą. Tokios funkcijos yra: eval, exec, system.

Norint tinkamai apdoroti įvestį šioms funkcijoms naudojami baltųjų ir juodųjų sąrašų (*angl. whitelisting and blacklisting*) metodai. Įtraukimas į baltąjį sąrašą yra metodas, naudojamas turiniui filtruoti pagal patvirtintą simbolių rinkinį, o juodieji sąrašai veikia atvirkščiai – visi į sąrašą įtraukti simboliai ar simbolių grupės yra užblokuoti. Įtraukimo į baltąjį sąrašą metodo privalumai yra itin stiprus saugumas ir efektyvus blokavimas pažeidžiamumą sukeliančių simbolių, tačiau tokio sąrašo implementacija kartais gali pareikalauti itin daug informacijos apie naudojamos aplikacijos kontekstą ir galimas pažeidimo spragas. Taip pat toks sąrašas atima dalį programos funkcionalumo lankstumo, todėl norint to išvengti, reikia itin gerai apgalvoti kaip tai turėtų būti implementuota. Tuo tarpu, juodasis sąrašas neatima tiek daug programos lankstumo ir yra daug paprasčiau imple-

mentuojamas, tačiau toks metodas nėra toks saugus kaip anksčiau minėtas, kadangi nėra veiksmingas, kuomet tam tikros grėsmės yra nežinomos arba neapsvarstytos [Pac22].

2. Kenkėjiškų komandų injekcijos prevencija

Python subprocesų modulis leidžia pradėti naujus procesus, prisijungti prie jų įvesties, išvesties, klaidų kanalų ir gauti jų grąžinimo kodus. Šio modulio metodai, skirti komandoms, kurios pateikiamos vartotojo kaip argumentai vykdyti, gali sukelti komandų injekcijų pažeidžiamumą. Piktavališki vartotojai gali išnaudoti tokį sistemos trūkumą tam, kad pasiektų jiems neprieinamus failus, perskaitytų jų turinį, ištrintų ar kaip nors kitaip pakenktų tinklapio programai. Ši problema itin aktuali, kuomet siekiama realizuoti komandas su specialiais „Unix" apvalkalo simboliais (konvejerių ar peradresavimo), kadangi tokiu atveju būtina iškviesti „Unix" apvalkalą tiesiogiai.

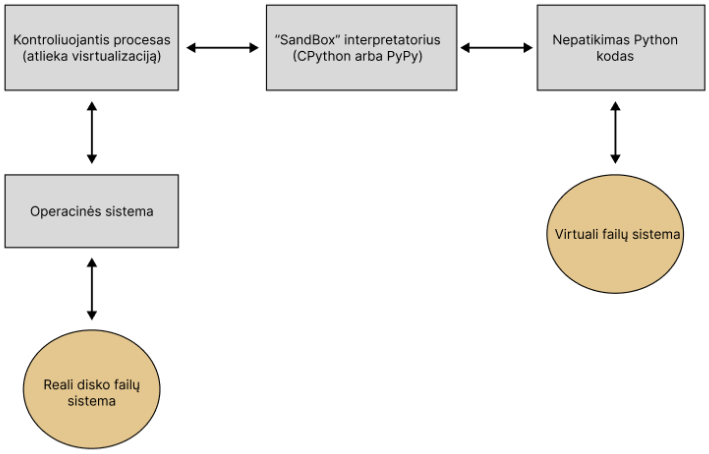
2.1. Apribotas „Python"

Python sąsaja gali būti suteikta vartotojams per žiniatinklio aplikaciją. Toks metodas gali sukelti saugumo pavojus, ypač jei suteikiamas vartotojams leidimas pridėti naujas funkcijas Python scenarijuje. Apribotas „Python" tokiose situacijose gali būti naudojamas, kaip papildoma prevencinė priemonė, tam kad būtų užtikrintas programos ir serverio saugumas. Apribotas „Python" (*angl. RestrictedPython*) veikia apibrėžiant apribotą Python kalbos poaibį, kuriame uždrausta prieiga prie tam tikrų kalbos elementų, nėra leidžiama kviesti jokios funkcijos, kuri nėra įtrauktą į leistinų metodų sąrašą (*angl. whitelist*) [FC22].

2.2. Smėlio dėžė

Tam, kad išvengti komandų injekcijų pavojaus galima naudoti smėlio dėžę (*angl. sandbox*), izoliuotą testavimo aplinką, leidžiančią vartotojams paleisti programas arba atidaryti failus nepažeidžiant sistemos, kurioje jie veikia. Dažniausiai smėlio dėžė naudojama išbandyti galimai kenkėjiškoms programinėms įrangoms, apribojant prieigą prie vartotojo duomenų ir sistemos išteklių. Viena iš tokių dėžių, kuriomis galima kontroliuoti „Python" kodo vykdymą - „PyPy" smėlio dėžė (*angl. PyPy sandboxing*), kuri veikia antrinio proceso principu: kaip antrinis procesas paleidžiamas nepatikimas „Python" kodas, tačiau jis nėra tiesiogiai atliekamas, vietoj to, šio proceso įvestis ir išvestis patenka į procesus kontroliuojančius vambzdžius. Nuskaitant vambzdžius nusprendžiama, kurios komandos leistinos, o kurios ne, arba kurios interpretuojamos kitaip, negu pateiktos

– virtualizuojamos. Naudojantis šią priemonę taip pat galima papildomai kontroliuoti naudojamą atminties kiekį, bei procesoriaus laiką [Pro22].



1 pav. „PyPy“ smėlio dėžė

2.3. Operacinės sistemos savybės

Yra įvairių „Linux“ operacinės sistemos funkcijų, kurias galima naudoti norint kontroliuoti procesų vykdymą, apriboti failus ir kitus įvesties, išvesties išteklius, prie kurių gali patekti procesas, bei apriboti procesoriaus atminties suvartojimą. Viena iš tokių funkcijų „chroot“ pakeičia vykdomo proceso ir jo dukterinių procesų šakninį katalogą. Taip sukuriamas „chroot“ kalėjimas (*angl. chrot jail*) – modifikuota aplinka, kurioje vykdoma programa negali pasiekti failų ir komandų, esančių už katalogo medžio ribų. Norint kontroliuoti procesų ir atidarytų failų kiekį, bei vartotojui prieinamą atmintį, galima naudotis „ulimit“ funkcija, kuri apriboja sistemos išteklių naudojimą. „Seccomp“ „Linux“ branduolio funkcija gali būti naudojama apriboti veiksmus konteinerio viduje, leidžiant pasiekti tik atidarytus failų deskriptorius arba nutraukiant procesus [Jac22].

2.4. Konteinerių valdymo įrankiai

Konteineriai yra programos kodo paketai kartu su priklausomybėmis, pvz., konkrečiomis programavimo kalbos vykdymo laiko versijomis ir bibliotekomis, reikalingomis programinės įrangos paslaugoms paleisti. Konteineriai gali būti naudojami paleisti programinės įrangos procesams, sudėtingoms programoms ar mažiems mikroservisams (*angl. microservises*). Vienas didžiausių konteinerių privalumų – žymiai mažesnės resursų sąnaudos, negu virtualios mašinos, kadangi kon-

teineriai nereikalauja individualios operacinės sistemos. Taip pat konteineriai gali būti naudojami: aplikacijų perkėlimui į naujas modernesnes aplinkas, sistemų patogesniam paleidimui, programų priklausomybių kontroliavimui. Konteineris „Docker“ gali tarnauti ir kaip kenkėjiškų komandų injekcijos prevencija. „Docker“ naudoja „Linux“ branduolio izoliavimo ir virtualizavimo funkcijas, kad nepriklausomi konteineriai galėtų veikti viename „Linux“ serveryje. Kiekviename „Docker“ konteineryje yra pagrindinė „Linux“ versija, tačiau pasinaudojus šiomis funkcijomis galima konteinerius sukonfigūruoti taip, kad būtų apribojami sistemos ištekliai [Net22].

2.5. Kelių serverių architektūra

Kelių serverių architektūra yra sistema, naudojama programai kurti, pagrįsta atskirų paslaugų, veikiančių kartu, pasirinkimu, siekiant užtikrinti sklandų ir labai greitą veikimą. Naudojant tokią architektūrą žiniatinklio programoms kurti, servisai suskirstomi pagal atskiras funkcijas, kurios diegiamos kaip atskiros programos. Tokios architektūros privalumai yra: lankstumas (kadangi kiekviena paslauga yra nepriklausoma, programuotojai gali kurti modulius naudodami skirtingas kalbas ar platformas), pakartotinas panaudojamumas (ši funkcija leidžia kūrėjams pakartotinai naudoti funkcinius modulius ir pritaikyti juos kitoms programoms), atsparumas pokyčiams ir saugumas (laisvai susietos programos funkcijos yra mažiau priklausomos viena nuo kitos, saugumo spragos vienoje dalyje nebūtinai pakenkia visai sistemai).Pakeitus sistemos architektūrą taip, kad žiniatinklio aplikacija veiktų viename serveryje, o kiekvieno vartotojo kodas būtų vykdomas atskiruose serveriuose, būtų sumažinta arba net pašalinta rizika, kad vartotojo kodas turės įtakos žiniatinklio paslaugos veikimui ar kitiems vartotojams. Pagal šį scenarijų žiniatinklio paslauga veikia kaip tarpininkas, nukopijuojantis vartotojo kodą į serverį, stebintis jo vykdymo būseną ir grąžinantis rezultatus vartotojams [Sim19].

3. EMBOSS įrankiai

„EMBOSS“ (*angl. European Molecular Biology Open Software Suite*) yra dažniausiai naudojamas atvirojo kodo programinės įrangos paketas, kuris gali tvarkyti neriboto ilgio sekas, bei duomenų bazes iš daugelio šaltinių [Pet12]. Šio paketo įrankiai skirti DNR ir baltymų sekų apdorojimui „UNIX“ platformoje. Įrankio populiarumą, kuris pradėjo augti 2002 m. nulėmė tai, jog jis sprendė keletą tuo metu buvusių bioinfarmatikinės programinės įrangos problemų – šis įrankis, ne taip, kaip kiti, buvo paprastai naudojamas ir įdiegiamas, nemokamas, atvirojo kodo, bei gerai sąveikavo su kitomis programomis. Toks įrankis tapo idealus mokamų įrankių pakaitalas asmenims, kurie gerai išmano ir moka naudotis „UNIX“ operacine sistema. Taip pat jis idealiai pritaikomas organizacijų serveriuose, bei naudojamas su žiniatinklio sąsajomis, tam kad suteiktų paprastesnį naudojimą mažiau patyrusiems vartotojams [Ols02].

EMBOSS yra sudarytas iš standartinio DNR ir baltymų sekų analizės programų: restrikcijos fermentų kartografavimo, transliacijos ir atvirkštinė transliacijos, porinio globalaus ir lokalaus palyginimo, atvirojo skaitymo rėmelio analizės, antrinės struktūros numatymo ir t. t. Taip pat pakete yra keletas programų, kurios paprastai nerandamos sekų analizės paketuose: CpG turtingų regionų brėžimas, DNR sekos sukimosi apskaičiavimas ir t. t. „EMBOSS“ programos automatiškai atpažįsta daugybę sekos formatų, todėl galima naudoti sekos failus, gautus iš kitų šaltinių, prieš tai jų nekonvertuojant. Taigi „EMBOSS“ puikiai veikia aplinkoje, kurioje naudojama daug skirtingų programų. Programos nenustato jokių sekos ilgio apribojimų. Visas programas galima paleisti tik iš komandinės eilutės, todėl galima kurti skriptus, norint automatizuoti sekų analizės procesus.

Taigi, „EMBOSS“ programų įrankis puikiai pritaikytas bioinformatikams, kadangi puikiai sąveikauja su skirtingomis programomis, yra puikiai pritaikomas kaip komandinės eilutės įrankis, susideda iš daugiau nei 150 programų, kurių panaudojimas yra labai plataus pobūdžio.

4. Biologinės aplikacijos kūrimas

4.1. Tikslai

Praktinei šio darbo daliai atlikti pasirinkta toliau tęsti sukurtos biologinės žiniatinklio programos kūrimą. Ši biologinė aplikacija suteikia galimybę prisijungusiems vartotojams naudotis terminalo emuliatoriumi tam, kad galėtų konstruoti „Unix“ apvalkalo komandų ir E-įrankių konvejerius. Vartotojai savo sukurtoje paskyroje gali įsikelti arba atsisiųsti failus su biologiniais duomenis, kuriuos vėliau galėtų apdoroti sistemos suteikiamų įrankių pagalba ir taip atlikti duomenų analizę arba apdorojimą. Šiame darbe siekiama atsižvelgti į saugumo problemas jau sukurtoje aplikacijoje ir pritaikyti prevencines priemonės prieš aptiktus pažeidžiamumus, taip pat pridėti naujo funkcionalumo: galimybę naudotis „EMBOSS“ įrankiais, bei kurti aplankalus.

Remiantis nagrinėta literatūra galima spręsti, jog ši aplikacija yra pažeidžiama komandų injekcijos atakoms. Kadangi ši injekcija pagal „OWASP” standartą laikoma itin pavojinga ir yra esminis programos trūkumas, būtina imtis atsakomųjų priemonių, tam kad šis pažeidžiamumas būtų pašalintas. Atsižvelgiant į „EMBOSS“ įrankių platų panaudojimą sekų analizėje ir į tai, jog juos patogu jungti į konvejerius, nuspręsta pridėti tam tikrus šio paketo įrankius: skirtus poriniam lokaliam ir lokaliam palyginimui. Dabartinėje jau realizuotoje biologinėje aplikacijoje suteikta galimybė kiekvienam vartotojui naudotis tik vienu nuosavo aplankalu. Siekiant suteikti vartotojams patogesnį ir platesnį sistemos panaudojimą, kuris imituotų menamą failų sistemą, aplikacijoje bus pridėta galimybė vartotojams kurti savo failų sistemas.

Siekiant įvertinti dabartinį programos pažeidžiamumą komandų injekcijoms pirmiausia bus atliekama komandų injekcijų atakų paieška su testavimo įrankių „COMMIX“. Atsižvelgiant į gautus testavimo rezultatus ir priemones, kurios siūlomos literatūroje, kaip atsakomosios ar prevencinės prieš šių atakų pažeidžiamumą, aplikacijoje bus siekiama sumažinti atakų galimybę ir išspręsti aptiktas saugumo problemas. Pritaikius atsakomąsias priemonės bus atliekamas pakartotinis testavimas, tam kad įvertinti ar pavyko išspręsti pažeidžiamumo problemą, bei, tam kad nustatyti pačios atsakomosios priemonės (metodo, kuriuo buvo siekiama užkirsti kelią atakoms) efektyvumą. Galiausiai bus implementuota galimybė vartotojams kurti aplankalus bei naudotis tam tikrais „EMBOSS“ įrankiais.

4.2. Testavimas

Siekiant nustatyti biologinės aplikacijos pažeidžiamumo lygį komandų injekcijos atakoms buvo atliktas testavimas su įrankiu „COMMIX“, kuris skirtas komandų injekcijų pažeidžiamumų eksplotavimui tinklapiuose ir žiniatinklio programose. Šis įrankio moduliai leidžia vartotojams aptikti silpnas saugumo vietas tam tikroje programoje. Numatytos atakos vykdomos programose, kurių pateiktas URL, HTTP antraštė ir identifikavimo slapukas. Vartotojas testuodamas, gali pateikti įvairius parametrus, kurie nurodo, kokio tipo atakų ieškoma, kokio lygio testavimas atliekamas (nuo 1 iki 3 lygio), kokia išvestis turi būti pateikta testų vykdymo metu.

Atliekant testavimą su šiuo įrankiu kaip įvestis buvo pateikta aplikacijos URL, tam tikro aktyvaus ir prisijungusio vartoto „CSRF” slapukas (nes vykdomos vartotojo pateiktos komandos, tik jam prisijungus), bei duomenys „JSON” formatu (’id’:’cmd’,’cmd’:INJECTHERE). Žymė „INJECTHERE“ nurodo, kurioje duomenų dalyje bus atlieka komandų injekcijos ataka. Nenurodžius šios žymės injekcija atliekama visuose duomenyse. Kadangi komandos identifikatorius šioje aplikacijoje nurodo kokia funkcija turėtų būti vykdomą, jame atlikti injekciją būtų netikslinga. Su šia įvestimi buvo atliktos visų keturių tipų injekcijos atakos.

Testavimo rezultatai:

1. Klasikinės rezultatais paremtos komandos injekcijos atveju pažeidžiamumas aptiktas jau po pirmojo testo. Pirmas testas – „ECHO 1 echo PVQWXX$((78+18))$(echo PVQWXX)PVQWXX“, gauta HttpResponse – „1 echo PVQWXX96PVQWXXPVQWXX“.

2. Dinaminio kodo vertinimo metodu pažeidžiamumo neaptikta.

3. Laiku paremtu nepastebimos komandos injekcijos atveju aptiktas pažeidžiamumas tik su vienu testu. Testas, su kuriuo aptiktas pažeidžiamumas – „echo 1 ;str=$(expr 7 + 2); if [9 - ne$str];then sleep 0;else sleep 1;fi“.

4. Atliekant injekcijos eksplotavimą failais paremtu metodu aptiktas pažeidžiamumas su testu – „echo 1 ||echo TCLNKG > /documents/TCLNKG.txt“

5. Atliekant injekcijos eksplotavimą laikinais failais paremtu metodu aptiktas pažeidžiamumas

su testu – „echo 1 ;str=$(cut -c1-2 /tmp/GENFTE.txt);if [51 -ne $str];then sleep 0;else sleep 1;fi“

Remiantis gautais testavimo rezultatais, galima teigti, jog sukurtoje biologinėje aplikacijoje aptiktas itin didelis pažeidžiamumas komandų injekcijos atakoms. Didžiausias aptiktas pažeidžiamumas yra klasikinėm atakoms. Testavimo metu nebuvo aptikta jokio pažeidžiamumo atakoms, kurios atliekamos dinaminio kodo vertinimo metodu.

4.3. Atsakomųjų priemonių taikymas

Remiantis išnagrinėta literatūra ir testavimo rezultatais nuspręsta, kad siekiant sumažinti pažeidžiamumą komandų injekcijoms bus bandoma izoliuoti serverio aplinką su „Docker“ konteinerių valdymo įrankiu [Inc22].Taip pat tam, kad apriboti vartotojo galimybes naudotis „Unix“ apvalkalu, bei ištrūkti iš savo apribotos aplinkas (savo failų sistemos) bus taikoma įrašymo į baltąjį sąrašą technika (*angl. whitelisting*). Po šių technikų taikymo, bus atliktas pakartotinas testavimas su „Commix“.

Šiame darbe buvo naudoti doker ir doker-compose failai tam, kad atskirtų aplikaciją nuo šeimininko operacinės sistemos ir taip pažeidžiamumo atveju, jos neįtakotų. Buvo panaudota daugiakonteirinė (*angl. multi-container*) aplikacijos sistema, kurioje atskirus konteinerius sudarė pati programa ir jos postgresql duomenų bazė.

Aplikacijoje buvo implementuoti baltieji sąrašai. Šiuo atveju tai buvo individualūs kiekvieno vartotojo failai (jiems neprieinami), kuriuose saugomi keliai iki visų direktorijų, kurias gali pasiekti tam tikras vartotojas. Vykdant bet kokią komandą pirmiausia patikrinama ar vartotojas turi prieigą prie atitinkamos direktorijos (pateiktos komandoje). Tokiu būdu vartotojas izoliuojamas tik savo aplankale.

Tam, kad apdoroti komandų įvestį, prieš jų vykdymą buvo atliktas komandų validumo patikrinimas (tikrinama, ar komanda nėra kenkėjiška). Pirmiausia visas komandų konvejeris padalinamas į atskiras komandas, tuomet tikrinama, ar pati „Unix“ komanda validi. Jei komanda leistina, tikrinamos jos opcijos. Tam tikros opcijos nėra leidžiamos, nes gali sukelti didelę žalą vartotojo failų sistemai (pvz.: rm -rf). Galiausiai tikrinama, ar vartotojo pateikti failai ir direktorijos yra jam prieinamos. Vykdant tam tikras komandas, kurių metu kuriami nauji failai ar direktorijos, tikrina-

ma ar pateikti vardai yra legalūs. Legalus pavadinimas laikomas tuomet, kai jame nėra tam tikrų simbolių: tarpų, klaustukų, šauktukų ir t.t. Jei komandos validumas patvirtinamas, ji yra įvykdoma ir jos išvestis perduodama kitai komandai, kaip įvestis. Žinoma, sekančios komandos validumas taip pat turi būti patvirtinamas.

Po atsakomųjų priemonių taikymo buvo atliktas pakartotinis testavimas su įrankiu „commix“. Testavimo metu nebuvo aptiktas pažeidžiamumas komandų injekcijoms. Testuojant vietoj šeimininko šakninio katalogo buvo pateiktas konteinerio katalogas. Šiuo atveju, net jei ir būtų aptiktas pažeidžiamumas, jis neturėtų tokios didelės neigiamos įtakos šeimininko operacinei sistemai, kadangi sistema veikia izoliuotoje aplinkoje. Taigi, galima teigti, kad baltųjų sąrašų metodas, kartu su izoliuota serverio aplinka („docker“ konteineriu) yra pakankamas įrankis norint apsisaugoti nuo komandų injekcijos atakų.

4.4. Naujo funkcionalumo implementacija

Šio darbo praktinėje dalyje taip pat siekiama biologinėje aplikacijoje pridėti naujo funkcionalumo. Tam, kad suteikti vartotojams galimybę sistemos pagalba atlikti sekų lokalius ir globalius palyginimus, buvo implementuoti „EMBOSS“ įrankiai skirti šiai paskirčiai. Šioje aplikacijoje vartotojams gali būti nepatogu naudotis įrankiais, kadangi kiekvienas vartotojas turi tik vieną aplankalą ir apribotos komandos, skirtos jiems kurti. Todėl svarbu implementuoti galimybę kurti savo failų sistemas.

Prevencinės priemonės, kurios buvo taikomos apsisaugoti nuo komandų injekcijos atakų, atvėrė galimybes platesnei įrankių įvairovei aplikacijoje. Kiekvienam vartotojui sukuriamas atskiras failas su jam pasiekiamom direktorijomis. Tokiu failu pasinaudoti galima ir siekiant suteikti vartotojams galimybę kurti aplankalus. Vartotojui kuriant direktorijas, keliai iki jų pridedami ir šiame faile. Tokiu būdu išlaikomas saugumas ir suteikiama galimybė kurti nuosavas failų sistemas.

Kviečiant komandinėje eilutėje tam tikrus „EMBOSS“ sekų palyginimo įrankius („needle“, „water“ ir t.t.), vartotojo paprašoma papildomų parametrų. Kadangi toks komandų vykdymas yra negalimas kuomet terminalas pateikiamas per tinklapį, nuspręsta „EMBOSS“ įrankius kviesti su visais būtinaisiais parametrais. Kai kurie iš jų yra numatytieji, todėl vartotojas neprivalo pateikti visų parametrų. Jei jie nėra pateikti nustatomi numatytieji ir komanda siunčiama į serverio pusę.

Taip nesutrikdoma kliento pusės ir serverio sąveika ir negrąžinama klaida.

Taigi, pridėtas papildomas funkcionalumas suteikė vartotojams platesnį sistemos panaudoją. Šių funkcijų implementacija buvo pritaikyta taip, kad butų palaikytos anksčiau minėtos priemonės, apsaugančios nuo komandų injekcijos atakų, ir aplikacijoje neatsirastų papildomų saugumo spragų.

4.5. Dar nerealizuotas funkcionalumas

Atsižvelgiant į jau implementuotą funkcionalumą aplikacijoje, perskaitytą ir išanalizuotą literatūrą, bei kūrėjo (užimančio vartotojo rolę) pastebėjimus, naudojantis aplikacija, galima spręsti apie dabartinės žiniatinklio programos trūkumus ir funkcionalumą, kuris būtų naudingas vartotojui. Apibendrinat visa tai, galima susidaryti planą, kaip dar galima patobulinti šią biologinę aplikaciją, tam kad vartotojui ji būtų naudingesnė ir patogesnė.

Tam, kad sistema būtų patogi ir paprastai naudojama vartotojui, galima atlikti nemažai patobulinimų. Kartais naudotis komandinės eilutės įrankiais nėra patogu, kadangi kai kurios komandos gali būti itin ilgos ir jų rašymas vartotojams ilgai užtruktų. Tam dažniausiai aukšto lygio terminalo emuliatoriai turi „tabo“ funkciją, su kuria vartotojas gali pratęsti savo įvestas komandas. Taip pat tokiuose terminaluose dažnai būna realizuota galimybė su rodyklėmis automatiškai įrašyti komandas iš savo komandų istorijos. Šios dvi funkcijos būtų itin naudingos kuriamoje aplikacijoje ir jų implementavimas nebūtų sudėtingas, kadangi jau pritaikyta dalis priemonių, kurios skirtos šiam funkcionalumui.

Sistemos platus ir tikslingas panaudojimas viena iš svarbiausių priežasčių vartotojui naudotis šia aplikacija. Todėl svarbu įdiegti daugiau įrankių, kurie apimtų kuo daugiau bioinformatikos sričių. Taip pat svarbu atsižvelgti ir į tai, jog ne visada terminalo emuliatorius yra patogi priemonė. Tarkim skaitant didelius failus ar juos redaguojant daug patogiau yra naudotis tekstiniu redaktoriumi. Todėl tekstinio redaktoriaus implementavimas būtų reikšmingas aplikacijos funkcionalumo pagerinimas.

Šioje aplikacijoje galima pritaikyti daugiau saugumą užtikrinančių priemonių. Reikėtų apsvarstyti pačio „docker“ konteinerio vykdytojo teisių apribojimą, vartotojo failų ir aplankalų dydžio limito nustatymą, serverio resursų apribojimą vykdant komandas.

Rezultatai

Šiame darbe buvo siekiama detaliau susipažinti su komandų injekcijos atakomis, bei prevenciniais metodais, skirtais apsisaugojimui nuo jų. Taip pat, remiantis įgytomis žiniomis ir testavimo rezultatais, buvo bandoma praktiškai pritaikyti atsakomąsias priemones savo biologinėje žiniatinklio programoje, tam kad užkirsti kelią tokioms atakoms.

Rezultatai:

1. Aprašytos komandų injekcijos ir jų tipai.
2. Pateikti komandų injekcijos padarinių pavyzdžiai, bei išanalizuota, kodėl ši ataka itin pavojinga.
3. Aptartas įrankis „commix“, kuris skirtas komandų injekcijos atakų aptikimui.
4. Išnagrinėta literatūra, aprašanti komandų injekcijos atakų prevencijos metodus bei priemones.
5. Aprašytas įrankių rinkinys „EMBOSS“, bei jo pritaikymas ir reikšmė bioinformatikoje.
6. Praktinėje darbo dalyje pateikiamas būdas, kaip atliekant testavimą su ”commix”, taikant „docker“ konteinerių valdymo įrankį ir baltųjų sąrašų metodą, galima apsisaugoti nuo komandų injekcijos atakų.

Išvados

Šio darbo metu buvo plačia išnagrinėtos komandų injekcijos atakos, bei priemonės, kurios taikomos siekiant jų išvengti. Taip pat aptarta „EMBOSS“ įrankių rinkinio reikšmė bioinformatikoje. Praktinėje darbo dalyje, pritaikyti įvairūs metodai, veikiantis kaip atsakomosios priemonės prieš komandų injekcijos atakas, bei implementuotas naujas funkcionalumas, suteikiantis vartotojui patogesnį, bei platesnį aplikacijos sistemos panaudojimą.

Remiantis darbo rezultatai, galima daryti šias išvadas (kai kuriuose punktuose pateikiama nuomonė):

1. Komandų injekcijos atakos yra itin paplitusios, kadangi galima jas atlikti naudojant daug įvairių metodų, iš kurių nemažai yra itin lengvai atliekami, o apsisaugojimas nuo jų reikalauja nemažai sistemos kūrėjų pastangų ir žinių.

2. Šios injekcijos neigiami padariniai gali būti itin dideli, nes tokios atakos metu gali būti pažeista visa vidinė serverio sistema, bei prieita prie neleistinų failų.

3. Komandų injekcijos eksploatacijos įrankis „commix“ veiksminga priemonė komandų injekcijos pažeidžiamumo aptikimui programose.

4. Programos sistemos izoliavimas nuo šeimininko operacinės sistemos, bei teisingas ir išsamus vartotojo įvesties apdorojimas, ar perfiltravimas, yra veiksmingos priemonės, norint išvengti komandų injekcijos atakų.

5. Remiantis pakartotinais testavimo rezultatais, baltųjų sąrašų įtraukimo metodas, kartu su „docker“ konteinerių valdymo įrankiu yra pakankamos atsakomosios priemonės prieš tokio tipo atakas aplikacijose.

Literatūra

[Ana14] Christos Xenakis Anastasios Stasinopoulos Christoforos Ntantogian. Commix: Detecting and exploiting command injection flaws, 2014. URL: https://www.blackhat.com/docs/eu-15/materials/eu-15-Stasinopoulos-Commix-Detecting-And-Exploiting-Command-Injection-Flaws-wp.pdf.

[Ant14] Sabastian Anthony. Shellshock: A deadly new vulnerability that could lay waste to the internet (updated), 2014. URL: https://www.extremetech.com/computing/190959-shellshock-a-deadly-new-vulnerability-that-could-lay-waste-to-the-internet.

[FC22] Zope Foundation ir Contributors. The idea behind RestrictedPython, 2022. URL: https://restrictedpython.readthedocs.io/en/latest/install/index.html.

[Fou21] Django Software Foundation. Django (Version 4.0), 2021. URL: https://www.djangoproject.com/.

[fou22] The Open Web Application Security Project foundation. Top 10 Web Application Security Risks, 2022. URL: https://owasp.org/www-project-top-ten/.

[Inc22] Docker Inc. Docker, 2022. URL: https://www.docker.com/.

[Jac22] Mike Jackson. Executing Python code submitted via a web service, 2022. URL: https://www.software.ac.uk/blog/2022-09-08-executing-python-code-submitted-web-service.

[MAr23] Matthew MArtin. Web application (Web app), 2023. URL: https://www.guru99.com/difference-web-application-website.html.

[Net22] NetApp. What are containers?, 2022. URL: https://www.netapp.com/devops-solutions/what-are-containers/.

[Ols02] Sue A. Olson. EMBOSS opens up sequence alignment, 2002. URL: https://academic.oup.com/bib/article/3/1/87/193749.

[Pac22] PacketLabs. What's the Difference Between Blacklisting, Whitelisting Greylisting?, 2022. URL: https://www.packetlabs.net/posts/blacklisting-whitelisting-greylisting/.

[Pet12] Jon Ison Peter Rice Alan Bleasby. EMBOSS, 2012. URL: https://www.ebi.ac.uk/Tools/emboss/.

[Pro22] The PyPy Project. PyPy's sandboxing features, 2022. URL: https://doc.pypy.org/en/latest/sandbox.html.

[Sim19] Sofija Simic. What are Microservices? Introduction to Microservices Architecture, 2019. URL: https://phoenixnap.com/kb/introduction-to-microservices-architecture.

[Sta22] Anastasios Stasinopoulos. Commix project, 2022. URL: https://commixproject.com/.