Kauno technologijos universitetas

Informatikos fakultetas

Kompiuterinės sistemos vientisumo užtikrinimo modelio sudarymas ir tyrimas

Baigiamasis magistro projektas

Laurynas Jonušas

Projekto autorius

Prof. dr. Algimantas Venčkauskas

Vadovas

Kaunas, 2024

Kauno technologijos universitetas

Informatikos fakultetas

Kompiuterinės sistemos vientisumo užtikrinimo modelio sudarymas ir tyrimas

Baigiamasis magistro projektas

Informacijos ir informacinių technologijų sauga (6211BX008)

Laurynas Jonušas

Projekto autorius

Prof. dr. Algimantas Venčkauskas

Vadovas

doc. Rasa Brūzgienė

Recenzentė

Kaunas, 2024

Kauno technologijos universitetas

Informatikos fakultetas

Laurynas Jonušas

Kompiuterinės sistemos vientisumo užtikrinimo modelio sudarymas ir tyrimas

Akademinio sąžiningumo deklaracija

2024 m. gegužės 24 d.

Kaunas

Patvirtinu, kad:

1. baigiamąjį projektą parengiau savarankiškai ir sąžiningai, nepažeisdama(s) kitų asmenų autoriaus ar kitų teisių, laikydamasi(s) Lietuvos Respublikos autorių teisių ir gretutinių teisių įstatymo nuostatų, Kauno technologijos universiteto (toliau – Universitetas) intelektinės nuosavybės valdymo ir perdavimo nuostatų bei Universiteto akademinės etikos kodekse nustatytų etikos reikalavimų;

2. baigiamajame projekte visi pateikti duomenys ir tyrimų rezultatai yra teisingi ir gauti teisėtai, nei viena šio projekto dalis nėra plagijuota nuo jokių spausdintinių ar elektroninių šaltinių, visos baigiamojo projekto tekste pateiktos citatos ir nuorodos yra nurodytos literatūros sąraše;

3. įstatymų nenumatytų piniginių sumų už baigiamąjį projektą ar jo dalis niekam nesu mokėjęs (-usi);

4. suprantu, kad išaiškėjus nesąžiningumo ar kitų asmenų teisių pažeidimo faktui, man bus taikomos akademinės nuobaudos pagal Universitete galiojančią tvarką ir būsiu pašalinta(s) iš Universiteto, o baigiamasis projektas gali būti pateiktas Akademinės etikos ir procedūrų kontrolieriaus tarnybai nagrinėjant galimą akademinės etikos pažeidimą.

Laurynas Jonušas

Patvirtinta elektroniniu būdu

Kauno technologijos universitetas

Informatikos fakultetas

Kompiuterinės sistemos vientisumo užtikrinimo modelio sudarymas ir tyrimas

Projekto tema	Kompiuterinės sistemos vientisumo užtikrinimo modelio sudarymas ir tyrimas

Reikalavimai ir sąlygos

1. Sukurti sistemos vientisumo nuolatinio tikrinimo mechanizmus.
2. Duomenų vientisumo palaikymą, sistemos prieinamumo užtikrinimą ir apsaugą nuo neleistinų pakeitimų.
3. Įdiegti realiuoju laiku veikiančias stebėjimo priemones, kurios įspėja administratorius apie įtartinus veiksmus, galinčius pakenkti vientisumui.
4. Nustatyti aiškius vientisumo užtikrinimo modelio veiksmingumo vertinimo rodiklius.

Vadovas / Vadovė	Prof. dr. Algimantas Venčkauskas	2024-05-24
	(vadovo pareigos, vardas, pavardė, parašas)	(data)

Laurynas Jonušas. Kompiuterinės sistemos vientisumo užtikrinimo modelio sudarymas ir tyrimas. Magistro studijų baigiamasis projektas vadovas prof. dr. Algimantas Venčkauskas; Kauno technologijos universitetas, Informatikos fakultetas.

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

Reikšminiai žodžiai: vientisumas, automatizacija, konfigūracija

Kaunas, 2024. 66 p.

Santrauka

Šio baigiamojo magistro projekto tikslas yra sudaryti ir ištirti kompiuterinės sistemos vientisumo užtikrinimo modelį. Šiuo projektu siekiama automatizuoti programinės įrangos kūrimo procesą ir užtikrinti kompiuterinės sistemos statinių procesų ir failų vientisumą per visą programinės įrangos kūrimo ciklą. Bus pasinaudota įvairiais automatizavimo įrankiais ir jų principais. Šie įrankiai bus naudojami kūrimo, testavimo ir diegimo procesams automatizuoti. Sudaryta centralizuota failų saugykla padės užtikrinti, kad tinkamos programinės įrangos versijos būtų pristatytos į tinkamą aplinką ir būtų išlaikomas failų vientisumas. Turėjimas centralizuotą sistemą per kurią būtų atliekami infrastruktūros pakeitimai ypač palengvina programavimo komandų bendradarbiavimą, nes suteikia bendrą konfigūracijos failų vaizdą ir galimybę valdyti prieigą ir leidimus.

Apibendrinant, šio projekto tikslas būtų leisti infrastruktūros administratoriams pastebėti administruojamų aplinkų vientisumo pasikeitimus ir leisti šias aplinkas automatizuotai konfigūruoti. Šie abu darbai gali būti atliekami rankiniu būdu, tačiau tai lėtas, varginantis ir nelankstus sprendimas, didelėse sistemose neretai sunkiai realizuojamas. Pritaikius sudaryto modelio sprendimą, bus galima sukurti pakartojamą, vientisą infrastruktūrą įvairiose operacinėse sistemose. Sprendimas skirs pagrinde dėmesį į kuo daugiau procesų automatizavimą, įrankio išplečiamumą ir individualų prisitaikymą prie norimų sprendimų.

Laurynas Jonušas. Development and Research of Model for Computer System Integrity Assurance. Master's Final Degree Project supervisor prof. dr. Algimantas Venčkauskas; Informatics Faculty, Kaunas University of Technology.

Study field and area (study field group): Informatics Engineering, Computing.
Keywords: integrity, automation, configuration

Kaunas, 2024. 66 p.

Summary

The aim of this master's thesis project is to develop and test a model for ensuring the integrity of a computer system. The aim of this project is to automate the software development process and to ensure the integrity of the static processes and files of a computer system throughout the software development cycle. Various automation tools and their principles will be used. These tools will be used to automate the development, testing and deployment processes. A centralized file repository will help to ensure that the right versions of the software are delivered to the right environment and that file integrity is maintained. Having a centralized system through which changes to the infrastructure can be made particularly facilitates collaboration between development teams by providing a single view of configuration files and the ability to manage access and permissions.
In summary, the aim of this project would be to allow infrastructure administrators to detect changes in the integrity of managed environments and to allow automated configuration of these environments. Both tasks can be done manually, but this is a slow, tedious and inflexible solution that is often difficult to implement in large systems. The application of the model solution will allow the creation of a repeatable, seamless infrastructure across different operating systems. The solution will focus on automating as many processes as possible, extensibility of the tool and customization to the desired solutions.

Turinys

Lentelių sąrašas ... **8**
Paveikslų sąrašas ... **9**
Santrumpų ir terminų sąrašas ... **10**
Įvadas ... **11**
1. Kompiuterinės sistemos vientisumo užtikrinimo metodų analizė ... **13**
1.1. Kompiuterinės sistemos konfigūracijos valdymo problema ... 13
1.2. Kompiuterinės sistemos programinės įrangos konfigūracijos valdymo metodai ... 14
1.2.1. Scenarijų rašymas ... 15
1.2.2. Traukimo modelis ... 15
1.2.3. Stūmimo modelis ... 15
1.3. Kompiuterinės sistemos programinės įrangos konfigūracijos valdymo produktai ... 16
1.3.1. *Ansible* ... 16
1.3.2. *Puppet* ... 17
1.3.3. *Terraform* ... 19
1.4. Kompiuterinės sistemos vientisumo konfigūracijos valdymo metodai ... 20
1.4.1. Failų struktūros patvirtinimas ... 20
1.4.2. Failo maišos arba kontrolinės sumos patvirtinimas ... 20
1.4.3. Failo parašo patvirtinimas ... 21
1.4.4. Failo laiko žymų patvirtinimas ... 21
1.4.5. Failo metaduomenų patvirtinimas ... 22
1.5. Kompiuterinės sistemos vientisumo konfigūracijos valdymo produktai ... 22
1.5.1. *OSSEC* ... 22
1.5.2. *Auditd* ... 24
1.5.3. *Beats* ... 24
1.6. Analizės išvados ... 25
2. Kompiuterinės sistemos reikalavimų specifikacija ir modelis ... **27**
2.1. Siūlomas modelis ... 27
2.2. Kompiuterinės sistemos vientisumo užtikrinimo modelio koncepcija ... 27
2.3. Išvados ... 49
3. Kompiuterinės sistemos realizacija ir tyrimas ... **50**
3.1. Kompiuterinės sistemos vientisumo modelio aprašymas ... 50
3.2. Kompiuterinės sistemos vientisumo modelio naudojama įranga ... 52
3.3. Kompiuterinės sistemos vientisumo modelio testavimas, apkrovos ir greitaveikos tyrimas ... 54
3.4. Palyginimas su esamomis sistemomis ... 62
3.5. Tyrimo apibendrinimas ... 63
Išvados ... **64**
Literatūros sąrašas ... **65**

Lentelių sąrašas

1 lentelė Procesų audito taisyklės 34
2 lentelė Failų audito taisyklės 36
3 lentelė Neįtraukimo taisyklės 38
4 lentelė Programinė įranga 52
5 lentelė Python bibliotekos 52
6 lentelė Fizinė ir virtuali aparatinė įranga 53
7 lentelė Konfigūracinis serveris 53
8 lentelė Konfigūruojama įranga 53
9 lentelė Būsenos serveris 53
10 lentelė Programų konfigūracijos tikrinimo failai 60
11 lentelė Visos sistemos tikrinimo failai 60
12 lentelė Sistemų funkcionalumo palyginimas 62

Paveikslų sąrašas

1 **pav.** Kompiuterinė sistema ... 13
2 pav. Konfigūracijos valdymas ... 14
3 pav. Katalogo sukūrimo scenarijus ... 15
4 pav. *Ansible* architektūros dizainas ... 16
5 pav. *Ansible* knygelė ... 17
6 pav. *Puppet* architektūros dizainas ... 18
7 pav. *HCL* sintaksė ... 19
8 pav. Kontrolinės sumos gavimas ... 21
9 pav. *OSSEC* integracija su trečiųjų šalių įrankiais ... 22
10 pav. OSSEC architektūra ... 23
11 pav. *auditd* architektūra ... 24
12 pav. Bendrinis vientisumo užtikrinimo modelis kompiuterinei sistemai ... 28
13 pav. Sistemos architektūra ... 29
14 pav. Audito failų atvaizdavimo žiniatinklio programa ... 30
15 pav. Sistemos konfigūravimas ... 31
16 pav. Būsenos ataskaitų palyginimas ... 33
17 pav. Kopijų saugojimo aukšto lygio architektūra ... 33
18 pav. Pasyvios būsenos naujinimo programos ... 39
19 pav. Parengti fizinių įrenginių būsenos ataskaita veiklos diagrama ... 41
20 pav. Parengti naudotojų būsenos ataskaita veiklos diagrama ... 43
21 pav. Parengti tinklo būsenos ataskaitą veiklos diagrama ... 45
22 pav. Parengti kernelio modulių būsenos ataskaita veiklos diagrama ... 46
23 pav. Parengti failų sistemos būsenos ataskaita veiklos diagrama ... 48
24 pav. Diegimo diagrama ... 49
25 pav. Tipinė būsenos programa ... 51
26 pav. Suformuoti tinklo įvykiai iš audito failų ... 54
27 pav. Virtualių tinklo įrenginių pokyčiai ... 54
28 pav. Tinklo audito įrašas ... 55
29 pav. Suformuoti modulių įvykiai iš audito failų ... 55
30 pav. Modulių audito įrašas ... 56
31 pav. Suformuoti naudotojų ir grupių įvykiai iš audito failų ... 56
32 pav. Naudotojų ir grupių audito įrašas ... 56
33 pav. Suformuotas aparatinės įrangos pranešimas ... 56
34 pav. Suformuotas failų pakitimo pranešimas ... 57
35 pav. Sistemos konfigūravimas ir vientisumo atnaujinimas ... 57
36 pav. Sistemos konfigūravimas ir atstatymas pagal atstatymo failus ... 58
37 pav. Paketų būsenos pakitimas ... 58
38 pav. *Ansible* moduliai ... 59
39 pav. Failų vientisumo skenavimo CPU apkrova ... 61
40 pav. Failų vientisumo skenavimo laiko trukmė ... 61
41 pav. *Auditd* naujų tinklo ryšių sudarymo apkrova ... 62

Santrumpų ir terminų sąrašas

KVS - Konfigūracijos valdymo sistema

COBIT - Informacijos ir susijusių technologijų valdymo tikslai (angl. *Control Objectives for Information and Related Technology*)

ISACA - Informacinių sistemų audito ir kontrolės asociacija (angl. *Association for Information Systems Audit and Control*)

YAML - Duomenų serializavimo kalba. (angl. *Yet another markup language)*

JSON – Bylos formatas (angl. *JavaScript Object Notation*)

API – Aplikacijų programavimo sąsaja (angl. *Application Programming Interface*)

IaC – Infrastruktūra kaip kodas (angl. *Infrastructure as Code*)

HCL – Infrastruktūros kaip kodas programavimo kalba (angl. *HashiCorp Configuration Language*)

SIEM – Saugumo įvykių valdymo sistema (angl. *Security information and event management*)

OSI – Ryšių protokolo modelis (angl. *Open Systems Interconnection Reference Model*)

Įvadas

Darbo problematika ir aktualumas

Šiandieniniam verslui reikia konfigūracijos valdymo, nes tai leidžia efektyviai valdyti ir sekti savo IT sistemų ir infrastruktūros pokyčius. Konfigūracijos valdymo būdu turi būti sukurta ir nuolat atnaujinama organizacijos IT infrastruktūra, įskaitant jos techninę, programinę įrangą ir dokumentaciją. Įdiegusios tokį valdymo būdą, organizacijos gali garantuoti, kad jų sistemos yra nuosekliai sukonfigūruotos, apsaugotos ir atitinka pramonės reglamentų taisykles.

Galimybė išlaikyti IT sistemų ir infrastruktūros kontrolę yra vienas iš pagrindinių konfigūracijos valdymo privalumų. Organizacijos gali greitai rasti ir išspręsti problemas, taip pat anuliuoti pakeitimus iškilus problemai, stebėdamos visus sistemos pakeitimus. Tai gali padėti išvengti brangių prastovų ir sumažinti bet kokių veiklos sutrikimų padarinius. Be to, tai leidžia įmonėms būti aktyvesnėms, planuoti būsimus pokyčius ir visapusiškai suprasti IT infrastruktūros būklę šiuo metu.

Konfigūracijos valdymas ypač gali padėti įmonėms laikytis įvairių įstatymų ir pramonės standartų. Įmonės gali užtikrinti, kad jos laikosi reguliavimo institucijų nustatytų gairių. Kaip pavyzdys ISO 9001. Šis standartas apibrėžia reikalavimus kokybės vadybos sistemai, įskaitant dokumentaciją, įvairias atsakomybes ir vidaus auditą [1]. Organizacijos, kurios diegia ir atitinka ISO 9001 reikalavimus, gali būti sertifikuotos kaip atitinkančios standartą akredituotos sertifikavimo įstaigos. Šis sertifikatas gali suteikti organizacijoms konkurencinį pranašumą, nes jis parodo klientams ir kitoms suinteresuotosioms šalims, kad organizacija yra įsipareigojusi laikytis aukštų kokybės standartų [2]. Kitas panašus standartas galėtų būti *COBIT*. Jis skirtas įmonės IT administravimui ir valdymui. Jame pateikiamas IT valdymo ir valdymo taisyklių ir geriausios praktikos rinkinys, kurį sukūrė ir paskelbė ISACA. Kad įmonės galėtų apsaugoti neskelbtinus duomenis ir išvengti baudų, šių įstatymų ir standartų laikymasis tampa vis svarbesnis [3].

Naudodamiesi konfigūracijos valdymu, organizacijos taip pat gali pagerinti savo veiklos tęstinumą ir atkūrimo planavimą. Įmonės gali geriau pasiruošti nenumatytoms situacijoms, pavyzdžiui, stichinėms nelaimėms ar kibernetinėms atakoms, ir į jas reaguoti, kai turi išsamią ir atnaujintą IT sistemų ir infrastruktūros inventorizaciją. Tai gali sudaryti sąlygas esminiams įmonės procesams tęsti nepaisant sutrikimų. Sistemos, kurias galima greitai atkurti iki žinomos geros būklės, gali sutrumpinti atkūrimo laiką ir sumažinti trikdžių poveikį.

Be to, dėl konfigūracijos valdymo įmonės gali greitai ir lengvai valdyti ir pristatyti programinės įrangos naujinimus, pataisymus ir naujinimus. Tai labai svarbu saugumui, nes tai leidžia greitai ištaisyti gedimus ir pažeidžiamumus. Atsisakius rankinio atnaujinimo ir sumažinus žmogiškųjų klaidų tikimybę, tai įgalina racionalesnes ir efektyvesnes IT operacijas.

Bendrai, konfigūracijos valdymas yra labai svarbus siekiant išlaikyti organizacijos IT sistemų patikimumą ir stabilumą, taip pat užtikrinti, kad jos ir toliau atitiktų bendrus įmonės tikslus ir uždavinius. Jis ne tik pagerina atkūrimą po nelaimių ir veiklos tęstinumo planavimą, bet ir leidžia įmonėms laikytis įstatymų ir pramonės standartų, išsaugoti savo IT sistemų ir infrastruktūros kontrolę bei racionalizuoti IT operacijas. Šiuolaikinės įmonės, kurių veikimas ir konkurencingumas priklauso nuo IT sistemų, turi ne tik vadovautis geriausia praktika, bet ir jos laikytis.

Problemos iššūkiai:

1. **Sistemų kompleksiškumas:** Nuolatinis IT sistemų didėjimas lygiagrečiai didina ir jos sudėtingumą, o tai apsunkina vientisumo užtikrinimą ir tikrinimą. Reikia nagrinėti, kaip įvairios technologijos ir tarpusavyje susijusios sistemos gali paveikti bendrą IT infrastruktūros vientisumą.
2. **Dažniausios vientisumo grėsmės:** Kadangi vientisumo pakitimo būdų gali būti **labai daug**, jas reikia teisingai atsikrinti. Būtina išanalizuoti dažniausiai pasitaikančias vientisumo problemas, kaip efektyviai jas sekti, sudaryti ir valdyti. Kitu atveju pats vientisumo sekimas gali sunaudoti didžiąją dalį *CPU* ir begalė disko vietos.
3. **Žmogiškasis faktorius**: Žmonių klaidos ar tyčiniai veiksmai gali turėti didelę įtaką sistemų vientisumui. Reikia išnagrinėti, kaip žmogiškąjį faktorių galima sumažinti naudojant konfigūracijos valdymą.

Šio darbo **tikslas** - sukurti programinės įrangos modelį užtikrinantį kompiuterinės sistemos vientisumą.

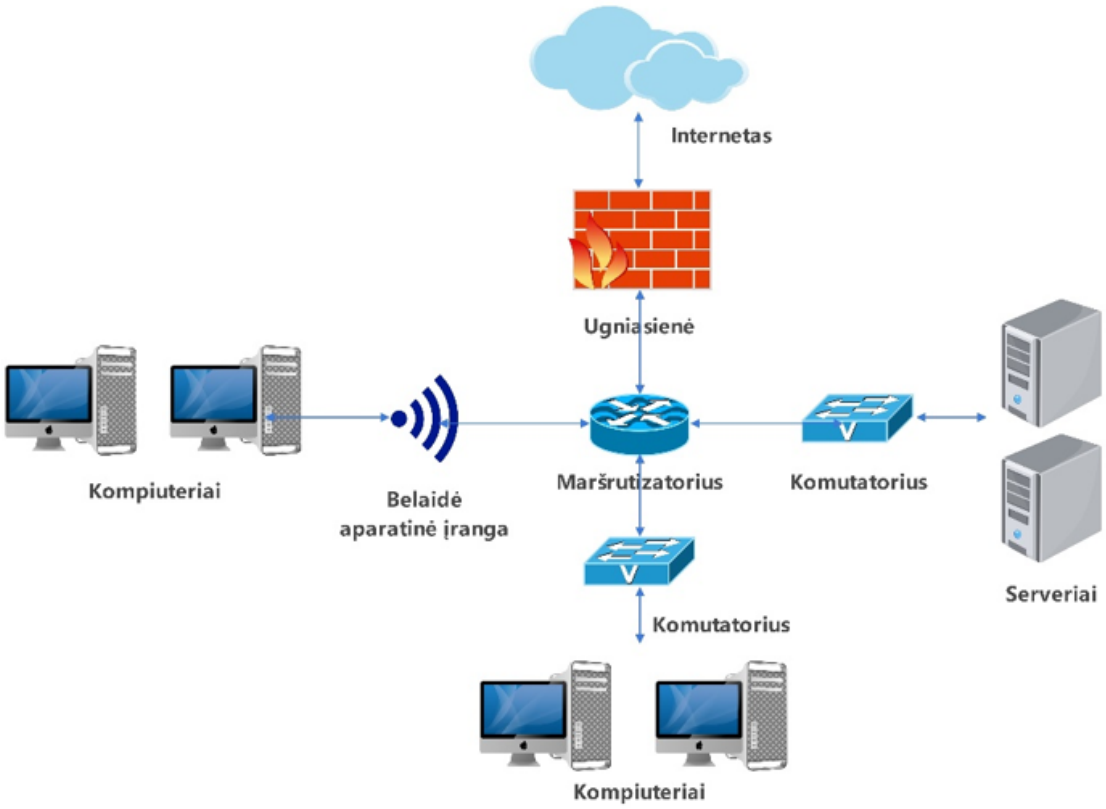
Šiame darbe keliami **uždaviniai** yra šie:

1. Išanalizuoti teorines vientisumą užtikrinančius sprendimus, egzistuojančius technologijas.
2. Sudaryti kompiuterinės sistemos vientisumą užtikrinantį modelį.
3. Pasinaudojus sudarytu modeliu, sukurti prototipą.
4. Palyginti sukurta realizacija su egzistuojančiomis technologijomis.

1. Kompiuterinės sistemos vientisumo užtikrinimo metodų analizė

1.1. Kompiuterinės sistemos konfigūracijos valdymo problema

Kompiuterinė sistema paprastai susideda iš tarpusavyje sujungtų įrenginių, tokių kaip kompiuteriai, serveriai, maršrutizatoriai, komutatoriai ir kita aparatinė įranga, kurie gali bendrauti tarpusavyje ir keistis informacija (žr. **1 pav.**). Šie įrenginiai sujungiami naudojant įvairius ryšio protokolus ir technologijas, tokias kaip *Ethernet*, *TCP/IP* ir *Wi-Fi*, kad suformuotų tinklo infrastruktūrą. Be fizinių įrenginių, tinkle taip pat yra įvairių programinės įrangos komponentų, tokių kaip tinklo operacinės sistemos, tvarkyklės ir programos, kurios naudojamos tinklui valdyti.



1 **pav.** Kompiuterinė sistema

Konfigūracijos valdymas yra plati sąvoka ir apima daug dedamųjų (žr. **2 pav.**). Konfigūracija susideda iš:

1. Konfigūracijos planavimas ir valdymas
2. Konfigūracijos identifikavimas
3. Konfigūracijos pakeitimų valdymas
4. Konfigūracijos būsenos apskaita
5. Konfigūracijos patikrinimas

Konfigūracijos identifikavimas yra pirmasis žingsnis konfigūracijos valdymo procese. Tai apima sistemos arba gaminių identifikavimą ir dokumentavimą. Tai apima fizinių ir funkcinių sistemų savybių, taip pat jų tarpusavio santykių nustatymą. Ši informacija naudojama kuriant konfigūracijos

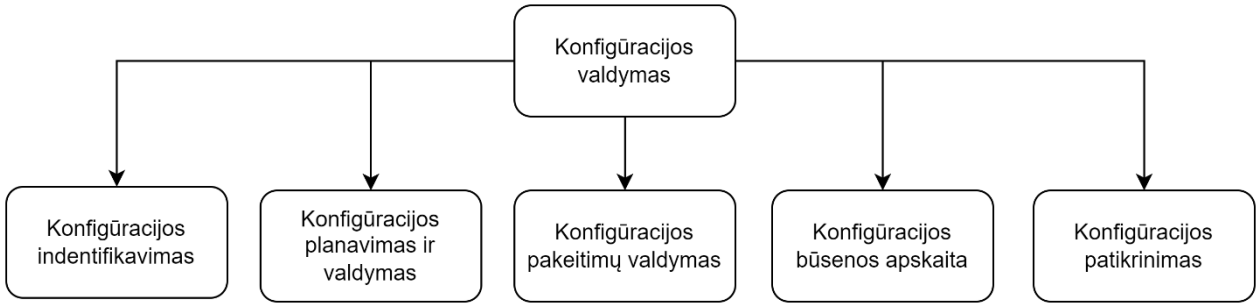
valdymo planą. Konfigūracijos identifikavimo veiksmas taip pat apima bazinės konfigūracijos nustatymą, kuri yra sistemų ir jų atributų momentinė nuotrauka tam tikru momentu, naudojama kaip nuoroda būsimiems pakeitimams [4].

Konfigūracijos planavimas ir valdymas yra konfigūracijos valdymo proceso žingsnis, apimantis konkrečių komponentų ir jų ryšių pokyčių valdymą ir valdymo plano sukūrimą laikui bėgant. Šis veiksmas apima konfigūracijos elementų, kuriuos reikia valdyti, nustatymą, jų atributų dokumentavimą ir jų pakeitimų valdymo proceso nustatymą.

Konfigūracijos pakeitimų valdymo tikslas yra užtikrinti, kad pakeitimai būtų atliekami kontroliuojamai ir nuosekliai ir kad jie neturėtų neigiamos įtakos sistemos ar produkto funkcionalumui ar veikimui. Šiame etape nustatomas pakeitimų poveikis sistemai. Šis valdymas yra nuolatinis procesas per visą produkto ar sistemos gyvavimo ciklą, siekiant užtikrinti, kad pakeitimai išliktų nuoseklūs ir tikslūs [5].

Konfigūracijos būsenos apskaita apima informacijos apie dabartinę sistemos ar gaminio konfigūracijos elementų būseną, istorijos rinkimą ir ataskaitų teikimą.

Konfigūracijos tikrinimas apima testavimą ir įvertinimą, siekdamas užtikrinti, kad viskas atitinka nurodytus reikalavimus ir patvirtintą konfigūraciją. Tai procesas, kuriuo užtikrinama, kad tikroji sistemos ar gaminio konfigūracija atitiktų numatytą arba nurodytą konfigūraciją.



2 pav. Konfigūracijos valdymas

Be konfigūracijos valdymo sistemos gali būti sudėtinga stebėti ir valdyti failų, programų ar aparatinės struktūros pasikeitimus tinkle, todėl gali atsirasti klaidų, neatitikimų ir saugos spragų.

1.2. Kompiuterinės sistemos programinės įrangos konfigūracijos valdymo metodai

Turėjimas programinės įrangos konfigūracijos metodą į kurį įeina tiek pačios programinės įrangos, tiek ir infrastruktūros atnaujinimai, paverčia pastaruosius darbus kur kas efektyvesniais ir sumažina klaidų tikimybę. Galime išskirti pagrindines naudas:

1. Nuoseklumas
2. Efektyvumas
3. Išplečiamumas
4. Patikimumas

5. Atitikimas (reglamentų, reikalavimų, politikų)
6. Saugumas

1.2.1. Scenarijų rašymas

Primityviausias ir jau laiko patikrintas būdas tai scenarijų (angl. *script*) rašymas (žr. **3 pav.**). Tai tikriausiai labiausiai iki šiolei paplitęs programinės įrangos konfigūravimo būdas. Pagrindinis šio metodo bruožas, kad tai buvo pirmasis plačiai paplitęs metodas, kai technologija panaudojama užduotims atlikti nedalyvaujant žmogui. Metodo gali būti panaudojamas pradedant paprastų scenarijų aprašymu iki sudėtingų programinės įrangos sistemų konfigūravimų. Vieną kartą parašius scenarijų jį galima pernaudoti galybę kartų. Jei scenarijus apima išsamią konfigūracijos logiką, skirtą įvairioms ir sudėtingoms situacijoms, tai gali sutaupyti administratoriaus laiko, ypač kai tą pačią konfigūraciją reikia taikyti keliems kompiuteriams. Tuo tarpu rankiniu būdu reikėtų įvykdyti komandas vieną po kitos, jas atsiminti ir atlikti viską eilės tvarka [6]. Daugelį užduočių galima automatizuoti naudojant scenarijų rašymą, įskaitant:

1. Kompiuterinių sistemų konfigūravimas ir priežiūra
2. Programinės įrangos diegimo automatizavimas
3. Nuolatinis integravimas ir nuolatinis diegimas (angl. *Continuous deployment*)
4. Stebėjimo ir įspėjimo integravimas

```
#!/bin/bash

mkdir -p /usr/share/naujas_katalogas;
```

3 pav. Katalogo sukūrimo scenarijus

1.2.2. Traukimo modelis

Traukimo modelis (angl. *Pull*) yra sistemų atnaujinimo metodas, kai sistemos pačios tikrina, ar nėra naujinimų, ir atsisiunčia juos iš centrinės saugyklos. Jis naudojamas programinės įrangos konfigūracijos valdymui. Vienas pagrindinių šio metodo pranašumų yra tai, kad sistemos gali nuspręsti, kada tikrinti, ar yra naujinimų, ir kada juos taikyti, naudodamos traukimo paradigmą. Tai suteikia daugiau lankstumo. Sistemos taip išvengia prastovų svarbiausiomis darbo valandomis, kuomet atliekami atnaujinimai. Šis metodas lengvai išplečiamas, nes galima naudoti keletą centrinių saugyklų taip neapkraunant vienos saugyklos. Taip pat, kuomet administratorius nori išbandyti naujinimą prieš išplatindamas jį visoms sistemoms, jis gali pakoreguoti tik mažą dalį sistemų, kurios vėliau parsitrauks naujos versijos atnaujinimus ir juos įsidiegs.

Apibendrinant, traukimo modelis suteikia didesnį lankstumą, mastelį, mažesnį tinklo srautą, mažiau prastovų ir savitarnos galimybes, kartu išsaugant versijų valdymą, todėl tai yra efektyvus programinės įrangos konfigūracijos valdymo metodas.

1.2.3. Stūmimo modelis

Sistemų atnaujinimo metodas, kai naujinimus į sistemas išsiunčia centrinis serveris, programinės įrangos konfigūracijos valdyme vadinamas stūmimo modeliu (angl. *push model*). Pagal šią paradigmą

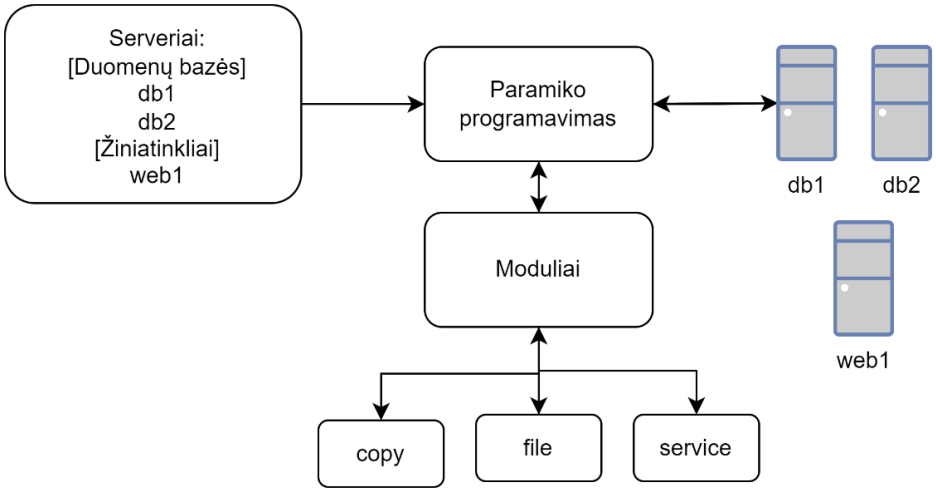
centrinis serveris yra atsakingas už atnaujinimų priežiūrą ir platinimą sistemoms, kurioms jų reikia. Pagrindinis šio modelio privalumas yra platesnis pritaikomumas įvairioms sistemoms. Atsimenant traukimo modelį, norint, kad jis tinkamai veiktu reikia į kiekvieną kompiuterinės sistemos komponentą įdiegti po agentą, kuris vis kreipsis į centrinį serverį. Kai kurias atvejais to padaryti gali būti neįmanoma. Kaip pavyzdys maršrutizatorius gali turėti mažai talpos, nepalaikyti tam tikrų servisų, kurių reikia norint paleisti agentą. Be to, šis metodas panaikina reikalavimą atskiriems komponentams tikrinti, ar nėra naujinimų, o tai gali sumažinti tinklo srautą ir pagerinti našumą. Tai gali būti ypač svarbu didelio masto kontekstuose, kur gali būti labai daug sistemų, pvz., duomenų centrų ar debesyse esančių sistemų.

Trumpai tariant stūmimo modelis yra naudingas programinės įrangos konfigūracijos valdymo metodas, nes jis suteikia centralizuotą valdymą, proaktyvius atnaujinimus, masinio diegimo galimybes yra paprastesnis nei, kad traukimo modelis. Tai geriausiai tinka scenarijams, kai administratorius nori būti pats atsakingas, kad visos sistemos būtų atnaujintos pagal tam tikrą tvarkaraštį ir su tam tikru naujinimų rinkiniu.

1.3. Kompiuterinės sistemos programinės įrangos konfigūracijos valdymo produktai

1.3.1. *Ansible*

Ansible yra atvirojo kodo programa, skirta konfigūracijos valdymui ir automatizavimui, kuri gali būti naudojama kompiuterinėms sistemoms valdyti ir konfigūruoti automatiškai (žr. **4 pav.**). Kadangi jos dizainas yra be agentų t.y stūmimo modelio, valdomose sistemose nereikia įdiegti jokios programinės įrangos, kad ji veiktų [7].



4 pav. *Ansible* architektūros dizainas

Naudojant *Ansible* dažniausiai atliekami šie veiksmai:

- Instaliuoti *Ansible* kompiuteryje ar serveryje iš kurio bus inicijuojami konfigūracijos pakeitimai.
- Apibrėžti savo inventorių. Tai yra sistemos, kurias norite valdyti naudodami *Ansible*. Inventorius gali būti nurodytas įvairiais būdais, pvz., paprastu teksto failu arba išoriniu duomenų šaltiniu, pvz., debesijos paslaugų teikėjo teikiama *API*.

– Aprašyti infrastruktūrą *YAML* formatu „knygelėse“ (angl. *playbooks*) (žr. **5 pav.**). Knygelėse aprašoma sistemų konfigūracija, programos, kurias reikia nustatyti, konfigūracijos failai, kurie turi būti sukurti ar nukopijuojami, ir paslaugos, kurios turi būti aktyvios.
– Paleidus Ansible ir jau sukūrus infrastruktūrą, kodą turėtumėte laikyti bendro valdymo saugykloje, pvz., *Git*. Naudodami versijos valdymą galite lengvai grįžti prie ankstesnės infrastruktūros versijos, jei kas nors negerai. Taip pat bendra versijavimo saugykla padeda programuotojų komandom lengviau dalintis kodu, daryti pakeitimus. Šis žingsnis visgi yra tik rekomendacija.

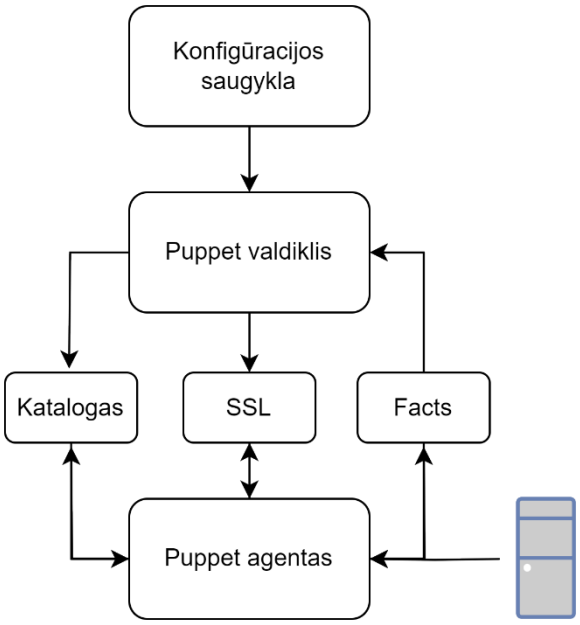
```
---
- name: Konfigūruoti žiniatinklį
  hosts: žiniatinkliai
  remote_user: root
  tasks:
  - name: Parsiųsti naujausios versijos Apache servisą
    ansible.builtin.yum:
      name: httpd
      state: latest
  - name: Nukopijuoti httpd.conf failą iš vietinės failų sistemos į nutolusį serverį
    ansible.builtin.template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf
```

5 pav. *Ansible* knygelė

Šie veiksmai suteikia apžvalgą, kaip naudotis *Ansible* sistemai valdyti. *Ansible* sukurtas taip, kad būtų galima greitai ir paprastai ja naudoti. Visgi didelėms įmonėms, kurios siekia įsitikinimo, kad *Ansible* sukonfigūruotas ir naudojamas kuo veiksmingesniu būdu, gali tekti imtis papildomų priemonių, nes plečiasi infrastruktūra ir plečiasi reikalavimai. Pavyzdžiui, kai norima tvarkyti savo knygeles ir inventorių, gali tekti naudoti *Ansible Tower* arba *Ansible Galaxy*, kuriuose aprašyti įvairus moduliai įvairiems veiksmams atlikti. Jų naudojimas gali sutaupyti laiko programuotojui, kad nereikėtų dažnai pasikartojančių konfigūracijų rašyti nuo nulio. Taip pat galima sukurti *Ansible* vaidmenis (angl. *roles*), kad infrastruktūros paruošimo scenarijai būtų logiškiau sudėlioti. Norint įsitikinti, jog infrastruktūros paruošimas įvyko sėkmingai galima integruoti papildomus stebėjimo servisus, kurie įspės apie iškilusias klaidas [8].

1.3.2. *Puppet*

Puppet taip pat yra atvirojo kodo konfigūracijos valdymo sprendimas. Šis įrankis automatizuoja IT sistemos administravimą (žr. **6 pav.**). Tai leidžia greitai ir nuosekliai valdyti ir konfigūruoti serverius, paslaugas ir kitus įrenginius visoje organizacijoje. Tai leidžia automatizuoti infrastruktūros valdymą, užtikrinant, kad visos jūsų sistemos būtų nustatytos nuosekliai, saugiai ir pagal jūsų įmonės reikalavimus. *Puppet* veikia traukimo modelio principu [9].



6 pav. *Puppet* architektūros dizainas

Puppet pagrindiniai komponentai:

1. *Puppet* valdiklis: ji atlieka lemiamą vaidmenį tvarkydamas visus su konfigūravimu susijusius darbus. Jis taiko nustatymus visiems klientų kompiuteriams arba tiksliniams mazgams.
2. *Puppet* agentas: programa instaliuota konfigūruojamame įrenginyje. Ji valdo mazgų konfigūracijas. Šie agentai nedelsdami atnaujins savo konfigūraciją, kad atspindėtų naują norimą būseną, jei bus atlikti pakeitimai. Žinoma, galima nustatyti intervalus, kuomet bus tikrinami konfigūracijos pasikeitimai.
3. Konfigūracijų saugykla: bendrinama vieta, kurioje visos konfigūracijos išsaugomos ir nuskaitomos, kai to reikalauja *Puppet* valdiklis. Šią saugyklą koreguoja programuotojai norėdami pridėti naujų funkcionalumų.
4. Faktai: Tai yra duomenys apie esama situaciją konfigūruojamame įrenginyje, kurie yra perduodami *Puppet* valdikliui. Konfigūruojamame įrenginyje yra klasifikacijos, funkciniai, regioniniai ir kiti susiję atributai, leidžiantys *Puppet* valdikliui juos ištirti, koreguoti ir pritaikyti reikiamas konfigūracijas.
5. Katalogas: sukompiliuota konfigūraciją, kurią reikia perkelti į tikslinius įrenginius ir ją vėliau įvykdyti.

Puppet panašiai kaip ir *Ansible* turi galimybę valdyti infrastruktūrą kaip kodą (angl. *IaC*) . Tai reiškia, kad galite bendradarbiauti su savo komandos nariais nustatydami infrastruktūros sistemą ir valdyti versijas. Be to, automatizuodami diegimo procedūrą galite sumažinti žmogiškųjų klaidų skaičių ir užtikrinti, kad jūsų sprendimai diegiami nuosekliai. Debesų paslaugų tiekėjai kaip *AWS*, *Azure* ir *Google Cloud* yra tik keletas iš daugelio naudojamų paslaugų ar kitaip tariant servisų, kurie turi *Puppet* integraciją ir siūlo daugybę integruotų modulių, kad būtų paprasta valdyti infrastruktūrą keliose platformose. Dėl šios priežasties jis šis įrankis yra patogus įmonėms, kurios naudoja skirtingas technologines platformas.

Apibendrinant galima teigti, kad *Puppet* yra galingas ir universalus konfigūracijos valdymo įrankis, leidžiantis automatizuoti IT infrastruktūros valdymą, kartu užtikrinant, kad visos sistemos būtų nuosekliai ir saugiai sukonfigūruotos. Tai padeda valdyti infrastruktūrą kaip kodą, automatizuoti diegimo procesą ir užtikrinti atitiktį bei saugumą. Tai puikus sprendimas visų dydžių organizacijoms, nes gali būti naudojamas valdyti įvairias sistemas, paslaugas ir įrenginius keliose platformose. Visgi reikia paminėti, kad šis įrankis ne visuomet tinka tokioms įmonėms, kurios specializuojasi daiktų interneto arba tinklinių įrenginių versle. Tai yra dėl to, nes šiuose gaminiuose ne visada galima įdiegti agentus dėl jų mažų procesoriaus kaštų arba talpos.

1.3.3. *Terraform*

Terraform yra *IaC* įrankis, leidžiantis teikti ir valdyti infrastruktūros išteklius. Svarbus pranašumas yra *Terraform* galimybė paskirstyti išteklius keliuose debesijos paslaugų tiekėjuose. Tai darant galima sukurti infrastruktūrą per keletą debesijos tiekėjų, o tai pasiūlo didesni sistemos patikimumą. Infrastruktūrą keliose platformose galima lengvai valdyti modulių dėka. *Terraform* naudoja deklaratyviąją kalbą, vadinamą *HCL* (žr. **7 pav.**), kad nustatytų norimą infrastruktūros būseną. Būsenos valdymas yra viena pagrindinių Terraform funkcijų. Šis mechanizmas leidžia stebėti infrastruktūros pasikeitimus ir skirtingiems žmonėms saugiai ir reguliuojamai dirbti toje pačioje infrastruktūroje. Tai suteikia galimybę dirbti kartu su komandos nariais ir konfigūruoti sistemas nesijaudindami dėl nesutarimų ar duomenų praradimo, nes vienam programuotojui atliekant pakeitimus, būsenos failas būna užrakinamas, todėl kiti programuotojai tuo metu bandydami atlikti infrastruktūros pakeitimus gauna klaidos pranešimą taip nesukeldami jokių konfliktų. Iš esmės būsenos failas yra parengiamas automatiškai ir yra pagrindinis infrastruktūros „tiesos šaltinis“. Šis įrankis fokusuojasi labiau į pačios infrastruktūros sukūrimą t.y įvairių debesijos resursų sukūrimą, o ne konkrečių servisų įdiegimą ar konfigūravimą viduje resursų. Labai dažnu atveju *Terraform* veikia šalia tokių įrankių kaip anksčiau aprašyti *Ansible* ar *Puppet*.

```
resource "aws_instance" "žiniatinklio_virtuali_mašina" {
  ami                    = "ami-a0cfeed8"
  instance_type          = "t2.micro"

  tags = {
    Name = "Žiniatinklio virtuali mašina 1"
  }
}
```

7 pav. *HCL* sintaksė

Siekdama palyginti norimą infrastruktūros būseną su esama infrastruktūros būkle, *Terraform* pirmiausia nuskaito ją nurodančius *HCL* konfigūracijos failus. Jei bus neatitikimų, *Terraform* automatiškai sukurs, atnaujins arba ištrins išteklius tam, kad infrastruktūra atitiktų nurodytą būseną. Automatizuodami infrastruktūros valdymą, galima drastiškai sumažinti žmogiškąsias klaidas ir užtikrinti, kad jūsų sistemos būtų nuosekliai sukurtos.

Reziumuojant galima pasakyti, kad *Terraform* yra galinga infrastruktūra kaip kodo įrankis, leidžiantis teikti ir valdyti infrastruktūros išteklius keliuose debesijos paslaugų teikėjuose. Jis naudoja deklaratyviąją kalbą, kad nustatytų pageidaujamą infrastruktūros būseną ir automatiškai aprūpintų

tuos išteklius, kad užtikrintų, jog jie atitinka apibrėžtą būseną. Tai leidžia lengvai valdyti daugybę išteklių ir užtikrina, kad jie visi būtų nuosekliai ir saugiai sukonfigūruoti. Be to, tai leidžia automatizuoti atitikties ir saugos procesą.

1.4. Kompiuterinės sistemos vientisumo konfigūracijos valdymo metodai

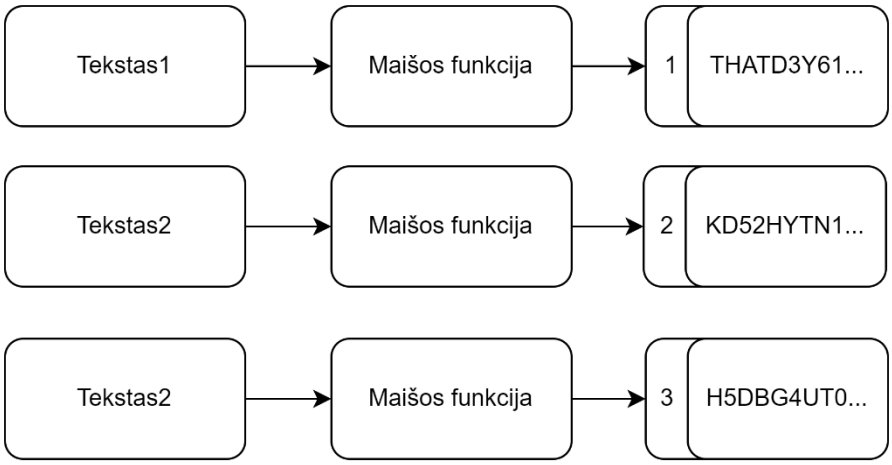
Duomenų vientisumas kompiuterinės konfigūracijos valdyme reiškia sistemos duomenų tikslumą, nuoseklumą ir išsamumą. Tai yra procesas, kuriuo užtikrinama, kad duomenys būtų tikslūs, nuoseklūs ir išsamūs bei išliktų tokie laikui bėgant [10].

1.4.1. Failų struktūros patvirtinimas

Šis duomenų tikrinimo sprendimas yra vienas paprasčiausių ir akivaizdžiausių. Jį sudaro keletas tikrinimo būdų, kurie sudaro sprendimo visumą. Pirmasis būdas – failo formato tikrinimas. Jis susideda iš priimtinų failų formatų apibrėžimo, t.y kuriuos galima naudoti konfigūracijos valdymo procese, sąrašo sudarymas, pvz., PDF, JPEG arba CSV. Sudarius sąrašą galime tikrinti konkrečius failus. Turėti tokį sąrašą svarbu, nes galima greitai atmesti netinkamus failus nedetalizuojant jų pačių vidaus. Taigi šis žingsnis svarbus užtikrinti, kad sistema ar programa, kuriai jis skirtas, galėtų tinkamai skaityti ir naudoti failą. Kitas tikrinimo būdo žingsnis yra patikrinti ar failas yra numatyto dydžio. Verta paminėti, kad norint atlikti šį tikrinimo būdą reikia turėti išsisaugojus visus failus vietiniame kompiuteryje arba lyginti su identiškai kitais sukonfigūruotais kompiuteriais. Tai galios ir sekantiems būdams. Toliau – failų vardų pasikeitimo tikrinimas ir failo struktūros patikrinimas. Šis žingsnis negalioja visiems failams. Jis aktualus labai dideliems failams kuomet prieš tikrinant visą failo turinį, norima patikrinti iš anksto priimta failo struktūrą ir patikrinti tam tikras sekcijas. Jeigu viskas iki toliai yra tinkama, tuomet galima pereiti prie viso failo turinio tikrinimo.

1.4.2. Failo maišos arba kontrolinės sumos patvirtinimas

Iš failų struktūros patvirtinimo sprendimo buvo galima pastebėti neoptimalius tikrinimo būdus, tokius kaip viso failo turinio tikrinimas. Tai neefektyvus tikrinimas užimantis daug laiko ir kompiuterio resursų. Failo maišos arba kontrolinės sumos patvirtinimas yra metodas, naudojamas failo vientisumui užtikrinti, sukuriant unikalu failo identifikatorių ir palyginant jį su išsaugota iš anksto verte, kuri buvo gauta tik įkėlus failą į kompiuterinės sistemos įrenginį. Šis patikrinimas atliekamas pasinaudojant maišos funkciją, kuri yra matematinis algoritmas, kuris priima failą kaip įvestį ir sukuria fiksuoto dydžio simbolių eilutę (žr. **8 pav.**), vadinamą maiša arba kontroline suma. Maišos arba kontrolinės sumos generavimą galime gauti iš tokių funkcijų kaip *SHA-256* arba *MD5*, norint sukurti unikalų failo identifikatorių. Jei failas kaip nors modifikuojamas, maiša arba kontrolinė suma pasikeis, todėl bus lengva nustatyti, ar failas buvo sugadintas arba pakeistas [11].



8 pav. Kontrolinės sumos gavimas

Svarbu atkreipti dėmesį į tai, kad reikia atsižvelgti ir į naudojamą maišos funkciją, kai kurios iš jų laikomos saugesnėmis nei kitos, todėl priklausomai nuo reikalaujamo saugumo lygio, maišos funkciją reikėtų pasirinkti atitinkamai.

1.4.3. Failo parašo patvirtinimas

Failo parašo patvirtinimas tai failo skaitmeninio parašo palyginimo su kitu žinomu parašu procesas, siekiant patvirtinti failo autentiškumą ir vientisumą. Skaitmeniniui parašui sukurti yra naudojamas matematinis metodas, kuris sukuria parašą. Parašo patvirtinimo procesas dažnai apima viešojo rakto infrastruktūros (angl. *PKI*) sistemos naudojimą, kuri užšifruoja ir iššifruoja parašą naudodama viešųjų ir privačių raktų porą. Šis metodas dažniausiai dirba ir su anksčiau aprašytu kontrolinės sumos metodu. Prieš pasirašant failą skaitmeniniu būdu, būna naudojamas maišos algoritmas ir sukuriama failo maišą. Tai paspartina pasirašymo darbą. Tada ši maiša užšifruojama naudojant pasirašančiojo asmeninį privatų raktą. Kai failą gauna gavėjas, skaitmeninis parašas iššifruojamas naudojant pasirašančiojo viešąjį raktą ir perskaičiuojama failo maiša. Jei naujai apskaičiuota maiša sutampa su pradine maiša, parašas laikomas galiojančiu, o failas patikrinamas kaip autentiškas ir nepakeistas. Šis metodas ypatingas tuo, kad užtikrina ne tik failo vientisumą, tačiau ir autentiškumą.

1.4.4. Failo laiko žymų patvirtinimas

Laiko žymų patvirtinimas naudojamas siekiant užtikrinti su failu susietų laiko žymų tikslumą konfigūracijos valdymo procese, dažnai laiko žymos pasitarnauja įvairiuose vidiniuose bei išoriniuose audito procesuose. Laiko žymos naudojamos norint nurodyti, kada failas buvo sukurtas, paskutinį kartą modifikuotas arba paskutinį kartą pasiektas, ir jas galima naudoti norint sekti failo istoriją ir nustatyti, ar failas buvo sugadintas arba pakeistas. Tai ypač svarbu tais atvejais, kai reikia atsekti pradinę failo būseną, pvz., programinės įrangos klaidos ar saugumo incidento atveju. Daugelis reglamentų ir standartų reikalauja, kad organizacijos tvarkytų tikslius savo failų įrašus, įskaitant jų kūrimo, modifikavimo ir prieigos datas. Turint laiko žymas, organizacijos gali užtikrinti, kad jų failai atitinka šias taisykles ir standartus.

1.4.5. Failo metaduomenų patvirtinimas

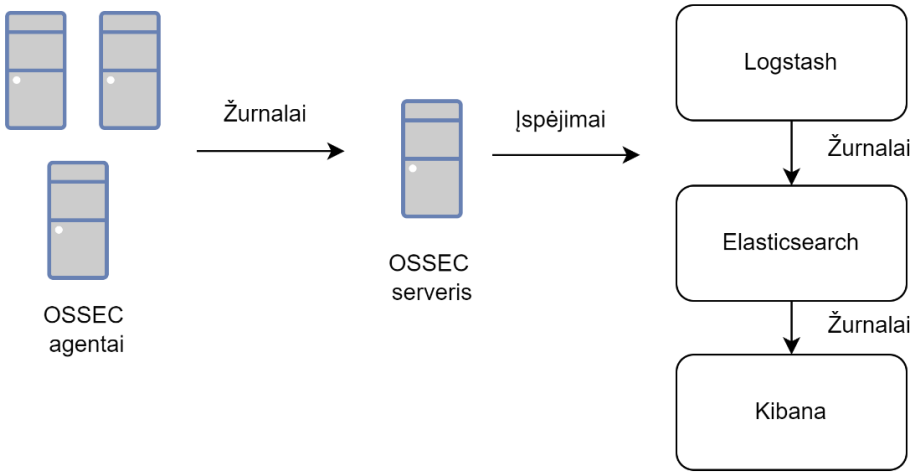
Metaduomenų patvirtinimas naudojamas siekiant užtikrinti su failu susietų metaduomenų tikslumą. Šis metodas yra vienas greičiausių tikrinimo metodų, kadangi jis netikrina failo turinio, ar nebando iššifruoti failo, kas užima nemažai laiko. Metaduomenys tai informacija, apibūdinanti failą, pvz., sukūrimo data, paskutinio modifikavimo data, failo dydis ir autorius. Ši informacija paprastai saugoma kaip paties failo dalis, tačiau ji taip pat gali būti saugoma atskirame faile arba duomenų bazėje. Norint naudoti šį metodą irgi reikia turėti vietinę duomenų bazę su metaduomenim apie failus, kuriais pasinaudojus bus atliekamas tikrinimas.

1.5. Kompiuterinės sistemos vientisumo konfigūracijos valdymo produktai

1.5.1. *OSSEC*

OSSEC yra gerai žinomas atvirojo kodo įsibrovimų aptikimo ir prevencijos sprendimas, padedantis įmonėms apsaugoti savo tinklus ir sistemas nuo saugumo rizikos. Tai efektyvus saugos sprendimas, siūlantis centralizuotą žurnalų stebėjimo ir analizės, saugumo grėsmių identifikavimo ir šalinimo bei integravimo su kitais saugos produktais platformą. *OSSEC* gebėjimas identifikuoti ir reaguoti į įvairius saugumo pavojus yra viena iš pagrindinių jos savybių. Jis aptinka saugumo grėsmes ir imasi atitinkamų veiksmų naudodamas taisyklėmis pagrįstą metodiką. Tai apima iš anksto nustatytų taisyklių rinkinį, skirtą nustatyti žinomą riziką, įskaitant tinklo invazijas, sistemos ir programų pažeidžiamumą, bandymus įgauti neteisėtą prieigą ir žinoma failų vientisumo pažeidimus. Šios gairės yra pagrįstos gerai žinomais saugumo pavojais ir geriausia praktika, todėl *OSSEC* gali nedelsiant nustatyti ir spręsti šias problemas. Be to, *OSSEC* gali lanksčiai kurti unikalias taisykles, kurios atpažintų tam tikras grėsmes. Tai leidžia įmonėms nustatyti savo taisykles ir nustatyti tik jų sričiai būdingus pavojus. Šis įrankis turi daugybę funkcionalumų kalbant apie reagavimą į grėsmes, įskaitant įspėjimų teikimą, tinklo srauto pristabdymą ir paslaugų nutraukimą. *OSSEC* serveryje papildomai yra sukurta žiniatinklio sąsaja, skirta sistemai valdyti ir įvairiems pavojaus signalams peržiūrėti. Naudojant šį įrankį administratoriai gali lengvai peržiūrėti saugumo problemas realiuoju laiku ir imtis papildomų veiksmų, kurie nėra aprašyti pačioje sistemoje [12].

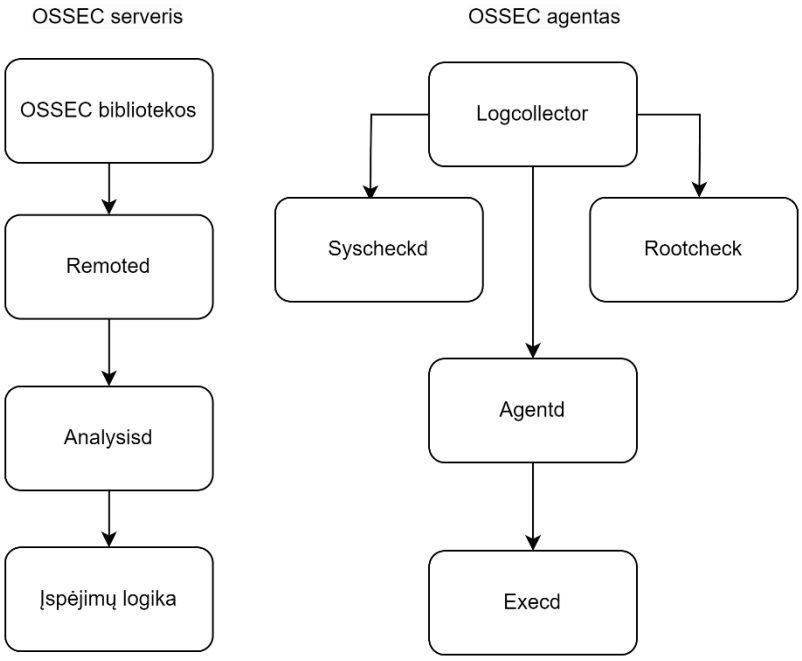
Žinoma, ne visiems užtenka *OSSEC* teikiamo žiniatinklio informacijos, dėl to galime pasinaudoti kitu svarbiu *OSSEC* aspektu – lengva integracija su kitais trečiųjų šalių įrankiais (žr**. 9 pav.**).



9 pav. *OSSEC* integracija su trečiųjų šalių įrankiais

Nors jis turi įvairius prietaisų skydelius, kuriuose atvaizduojama įvairi informacija visgi ji nėra ideali, todėl būna pasirenkami kiti įrankiai kaip *Grafana*. Neretai taip elgiamasi ir *OSSEC* atveju. Norint pasiūlyti išsamesnį saugos sprendimą, organizacijos gali dalytis saugos duomenimis ir automatizuoti atsaką į grėsmes, integruodamos *OSSEC* su trečiųjų šalių saugumo sprendimais, pvz., *SIEM* platformomis ir kitas reagavimo į incidentus platformomis. Koreliuodamos duomenis iš kelių šaltinių organizacijos gali reaguoti į riziką, kuri kitu atveju būtų nepastebėta.

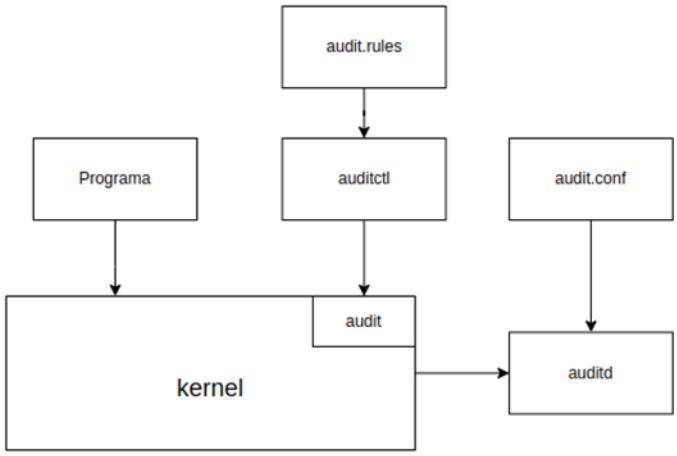
OSSEC diegimas susideda iš daugelio agentų, kurie nuolat renka žurnalus, išsaugo juos vietoje ir siunčia pagrindiniam *OSSEC* serveriui. Visi agento žurnalai lyginami su iš anksto nustatytomis taisyklėmis, o tada atliekami įvairūs veiksmai pagal apibrėžtus nustatymus. Pagrindinės OSSEC dizaino sudedamosios (žr. **10 pav.**) *OSSEC* pagrindiniame serveryje, yra *OSSEC* bibliotekos failai, kas yra kitaip tariant dvejetainiai failai, kuriuos naudoja *OSSEC* serveris. Jos reikalingos vidinėms funkcijom veikti, nes jos yra nuo jų priklausimos. Norėdami gauti kiekvieno agento žurnalus arba siųsti komandas į jas, *Remoted* užmezga ryšį su agento *Agentd*. Dekodavimo bloke *Analysisd* agentų žurnalai analizuojami ir iškoduojami, kai juos gauna *Remoted* valdymo pultas. Jei žurnalai atitinka vieną ar kitą taisyklę iš taisyklių bloko, *Analysisd* signalizuoja apie įvykį įspėjimų logikos blokui, kuris siunčia pranešimus į atitinkamas vietas. *OSSEC* agente yra 3 komponentai atsakingi už žurnalų rinkimą ir išsamios analizės vietinėje sistemoje atlikimą. Šie komponentai yra - *Logcollector, Syscheckd* ir *Rootcheckd.* Surinkti žurnalai iš šių komponentų siunčiami *Agentd,* kuris juos normalizuoja ir vėliau perduoda informaciją *OSSEC* pagrindiniam serveriui. Jei *OSSEC* serveris pagal gautus žurnalus turi sukonfigūruotą tam tikrą veiksmą tai jis išsiųs įspėjamąjį pranešimą administratoriui ir susisieks su *Agentd* komponentu, kuriam perduos kokie veiksmai turi būti atlikti agentų mašinoje. Šiuos veiksmus atliks ne kas kitas, o *Execd* komponentas [13].



10 pav. OSSEC architektūra

1.5.2. *Auditd*

Viena žymiausių ir galingiausių audito sistemų yra *auditd.* Ji yra universali ir galinga audito sistema, pagrinde naudojama „*Linux*“ operacinėse sistemose. Ji neatsiejama nuo sistemos saugumo ir atitikties didinimo. Ji veikia stebėdama ir įrašinėdama sistemos iškvietimus (žr. **11 pav.**) pagal iš anksto nustatytas saugumo taisykles, todėl yra labai svarbi priemonė atliekant kriminalistinę analizę ir nustatant saugumo pažeidimus. Auditd siūlo išsamias registravimo galimybes, kurios labai svarbios atliekant atitikties ir saugumo vertinimus, todėl ji yra mėgstamas pasirinkimas aplinkoje, kurioje reikia griežtų saugumo priemonių, pavyzdžiui, vyriausybinėse sistemose, finansų sektoriuose ir sveikatos priežiūros pramonėje. Patobulintos auditd konfigūracijos ir modifikacijos leidžia padidinti saugumo ir atitikties valdymą, dar labiau sustiprindamos jo, kaip pagrindinės saugumo priemonės, vaidmenį.



11 pav. *auditd* architektūra

1.5.3. *Beats*

Beats, lengvas ir efektyvus duomenų rinkimo agentas, sukurtas *Elastic*, gali būti greitai nustatomas stebėti konkrečius sistemos failus ar aplankus. Jis gali būti naudojamas norint nustatyti bet kokius pakeitimus, pvz., failo nuosavybės, leidimų ar net failo turinio pakeitimus. Šis įrankis leidžia lengviau pastebėti ir sustabdyti tolesnius nepageidaujamus pakeitimus, kurie gali būti simptomas saugumo pažeidimo. Be to, *Beats* siūlo įvairius išvesties pasirinkimus, leidžiančius siųsti informaciją į *Elasticsearch, Logstash, Kafka* ir kt.

Beats tipų yra daug ir įvairių. Keletas populiaresnių:

- *Filebeat*
- *Metricbeat*
- *Packetbeat*
- *Winlogbeat*
- *Auditbeat*
- *Heartbeat*
- *Functionbeat*

Filebeat nuolat stebi konfigūracijoje nurodytus žurnalo failus arba katalogus, ar nėra naujų ar atnaujintų ar pridėtų failų. Kai aptinkamas naujas failas, *Filebeat* nuskaito failą, ištraukia atitinkamus duomenis ir siunčia jį į nurodytą išvesties paskirties vietą.

Metricbeat periodiškai renka metrikas iš įvairių sistemos ir paslaugų modulių ir siunčia jas į nurodytą išvesties paskirties vietą. *Metricbeat* pristatomas su įvairiais integruotais moduliais, kurie gali rinkti metriką iš populiarių paslaugų ir sistemų, tokių kaip *Apache, MySQL* ir *Redis*. Jis taip pat gali būti lengvai išplėstas, kad būtų galima rinkti kitų paslaugų ir sistemų metriką, sukuriant pasirinktinius modulius.

Packetbeat fiksuoja tinklo srautą ir analizuoja jį realiuoju laiku, kad išgautų naudingą informaciją, pvz., šaltinio ir paskirties *IP* adresus, prievadus ir protokolus. Jis taip pat gali išgauti informaciją iš *OSI* taikymo programų sluoksnio, pvz., *HTTP* užklausą ir atsakymo antraštes.

Winlogbeat veikia stebėdamas *Windows* įvykių žurnalus, ar nėra naujų įvykių, ir persiunčia juos į nurodytą išvesties paskirties vietą. Jis gali būti sukonfigūruotas rinkti įvykius iš konkrečių įvykių žurnalų, pvz. saugos arba programų žurnalų, ir rinkti tik įvykius su tam tikrais įvykių ID arba lygiais. Ji taip pat palaiko duomenų filtravimą ir apdorojimą prieš juos siunčiant į išvestį, pvz., pridedant ar pašalinant laukus arba konvertuojant įvykio duomenis į konkretų formatą.

Kompiuterinės sistemos vientisumo kontekste šis *Beat* yra pats aktualiausias. Sistemos lygio audito duomenis *Auditbeat* renka iš operacinės sistemos branduolio ir siunčia į nurodytą išvesties vietą. Jis gali rinkti informaciją iš įvairių vietų, įskaitant failų sistemą, audito sistemą ir sistemos iškvietimus [14].

Heartbeat veikia reguliariai siųsdama užklausas į konkrečius paslaugos galinius taškus ir stebėdama atsakymo laiką. Galima žiūrėti daugybę paslaugų rūšių, įskaitant *HTTP, HTTPS, TCP* ir *ICMP*. Be to, jis gali būti nustatytas perduoti konkrečias užklausų antraštes, naudingąsias apkrovas ir eilutes, taip pat ieškoti konkrečių atsakymų kodų ir eilučių [15].

Functionbeat veikia siųsdamas debesies funkcijos įvykius į nurodytą išvesties vietą. Palaikoma daugybė debesies funkcijų teikėjų, įskaitant *AWS Lambda, Google Cloud Functions* ir *Azure Functions.* Jis gali būti nustatytas taip, kad klausytis tam tikrų įvykių, kuriuos išleidžia tam tikros funkcijos, ir filtruoti tuos įvykius prieš juos išsiunčiant [16].

Apibendrinat, *Beats* naudojimas failo vientisumui ir kitoms sistemos metrikoms patikrinti yra veiksmingas būdas užtikrinti svarbiausių duomenų tikslumą, nuoseklumą ir tinkamą programos veikimą. Šis įrankio integravimas veikia labai panašiai kaip *OSSEC* su trečiųjų šalių įrankiais. (žr. **9 pav.**). Skirtumas tik toks, kad nėra naudojamas tarpinis valdymo serveris, o *Beats* tiesiogiai siunčia metrikas į centralizuotą žurnalų surinkimo servisą. *Beats* suteikia administratoriams informaciją, kurios reikia, kad jie galėtų būti užtikrinti dėl savo sistemų saugumo ir stabilumo. Tai mažos apimties, bet efektyvus sprendimas, kurį paprasta įtraukti į dabartines saugos ir stebėjimo sistemas.

1.6. Analizės išvados

1. Konfigūracijos valdymas yra svarbus norint išlaikyti sistemų ir programų nuoseklumą ir stabilumą bei užtikrinti, kad jos būtų sukonfigūruotos saugiai ir suderintai.
2. Failų vientisumo stebėjimas yra būtinas norint aptikti ir užkirsti kelią neteisėtiems svarbių failų pakeitimams, kurie gali reikšti saugumo pažeidimą ar kitą problemą.

3. Konfigūracijos valdymo ir failų vientisumo stebėjimo įrankių, pvz., *Beats*, naudojimas gali padėti organizacijoms greitai nustatyti problemas ir į jas reaguoti bei išlaikyti savo sistemų saugumą ir stabilumą.
4. Automatizuotas konfigūracijos valdymas ir failų vientisumo stebėjimas gali padėti sumažinti žmogiškųjų klaidų riziką ir pagerinti bendrą saugumą bei atitiktį.
5. Tinkamas konfigūracijos valdymas ir failų vientisumo stebėjimas gali padėti organizacijoms atitikti norminius ir atitikties reikalavimus bei sumažinti duomenų pažeidimų ir kitų saugumo incidentų riziką.

2. Kompiuterinės sistemos reikalavimų specifikacija ir modelis

2.1. Siūlomas modelis

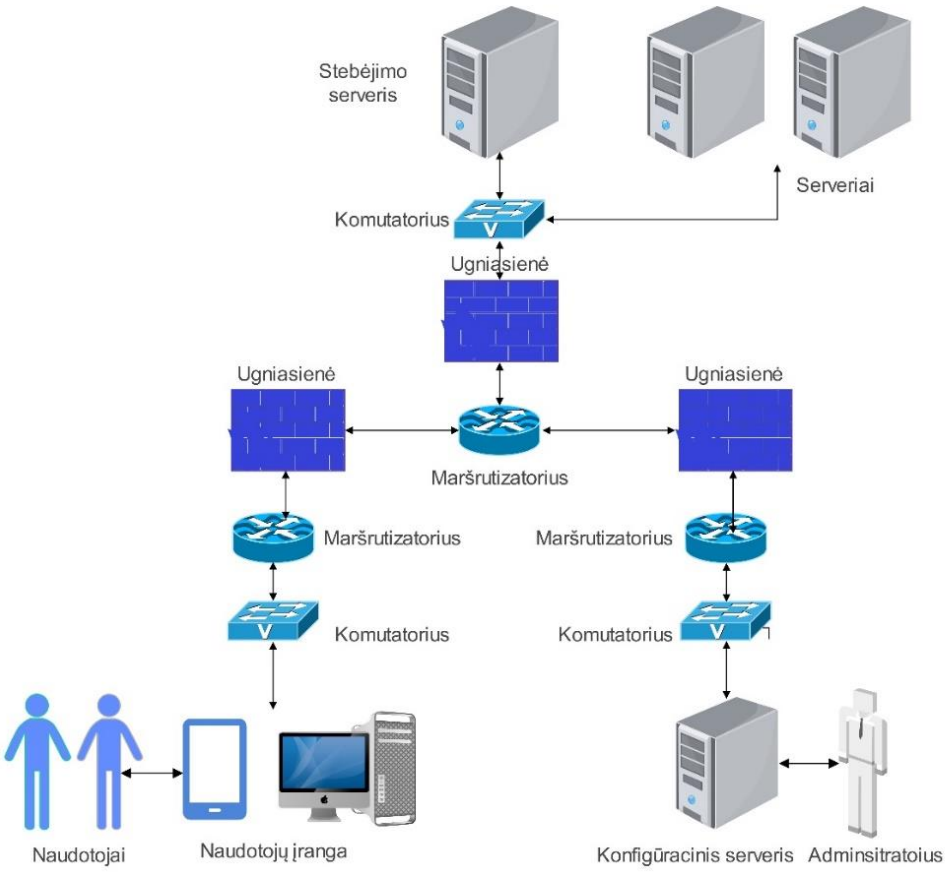
Šiame skyriuje įvardijamas modelis ir jo komponentai leidžiantys efektyviai išlaikyti sistemos vientisumą. Naudojantis siūlomu metodu galima sekti pokyčius kompiuterinėje sistemoje, sugebėti atstatyti failus, konfigūruoti sistemą išlaikant vientisumą.

Modeliui iškelti reikalavimai:

- Užtikrinti, kad visi tinklo, aparatinės įrangos ir laisvai pasirenkamų failų sistemų konfigūracijos pakeitimai būtų stebimi ir registruojami audito ataskaitose.
- Palaikomas būsenos failas, kuriame įrašoma visų stebimų sistemų esama ir buvusi konfigūracija: Šis failas veikia kaip centralizuota duomenų bazė, suteikianti išsamią informaciją apie visų sistemų būklęĮdiegti pakeitimų grąžinimo į ankstesnę būseną mechanizmą naudojant atšaukimo komandas: Tai suteikia galimybę greitai atstatyti sistemą į veikiančią būseną, sumažinant prastovų laiką ir veiklos sutrikimų poveikį
- Pasyviai, bet reguliariai fiksuojama konfigūruojamų mašinų būseną.
- Optimizuotas našumą, kad būtų galima apdoroti didelius duomenų srautus ir aptikti dažnus pokyčius: Tai svarbu siekiant užtikrinti, kad konfigūracijos valdymo sistema galėtų efektyviai veikti net ir esant dideliam duomenų apdorojimo poreikiui. Tai padeda užtikrinti greitą ir tikslų pokyčių aptikimą bei reagavimą į juos.

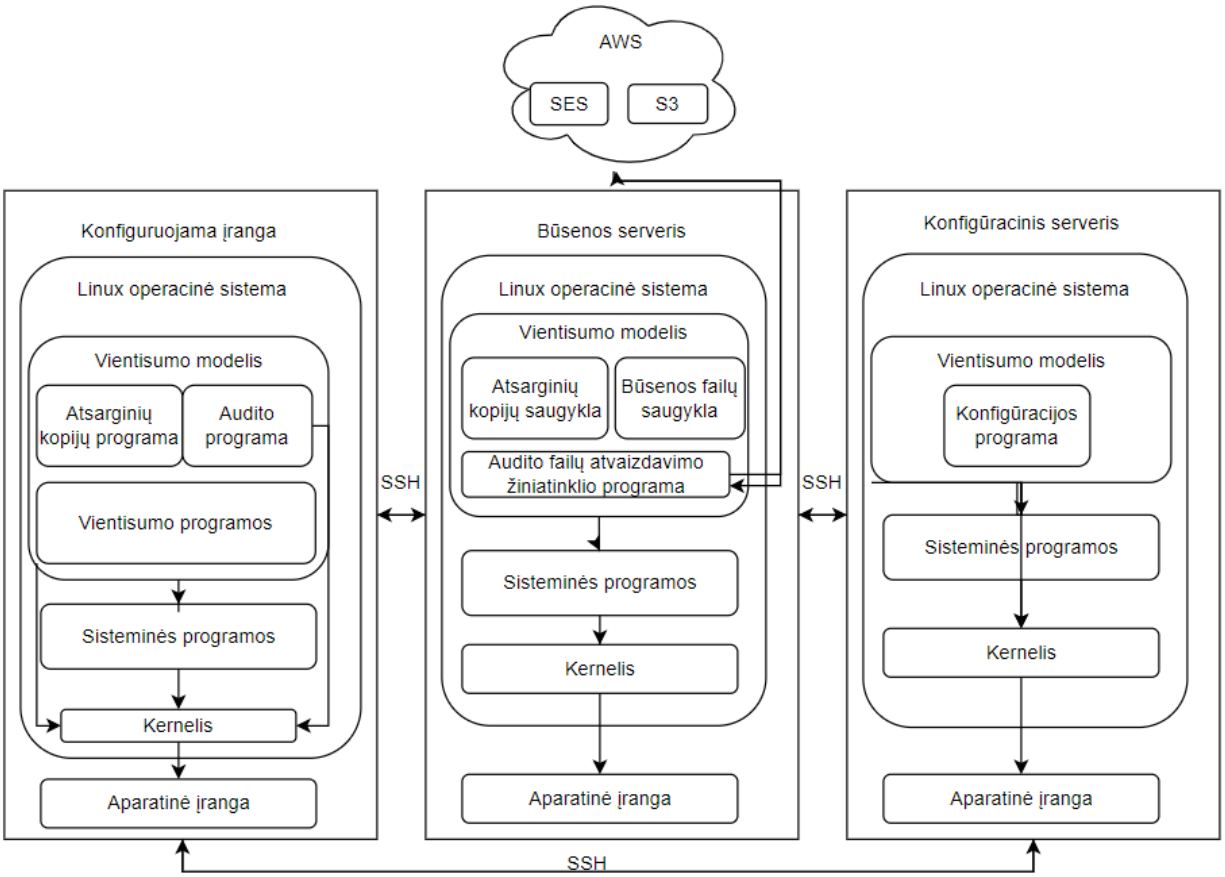
2.2. Kompiuterinės sistemos vientisumo užtikrinimo modelio koncepcija

Sistemos struktūros modelį sudaro įvairūs tarpusavyje sujungti komponentai, kurie komunikuoja tarpusavyje ir sudaro funkcionalų kompiuterių tinklą (žr. **12 pav.**). Ši modelį sudaro kompiuteriai, serveriai, komutatoriai, maršrutizatoriai ir ugniasienės. Serveriai tvarko centralizuotas paslaugas ir teikia išteklius klientams. Ugniasienės atlieka svarbų vaidmenį užtikrinant tinklo saugumą, nes stebi ir kontroliuoja įeinantį ir išeinantį tinklo duomenų srautą, užtikrindamos apsaugą nuo neteisėtos prieigos ir galimų grėsmių. Kiti modelio komponentai nėra esminės šios struktūros dalys, tačiau jas verta taip pat paminėti norint susidaryti detalesnį architektūros vaizdą. Taigi, komutatoriai palengvina ryšį tarp tinklo įrenginių, persiųsdami duomenų paketus į numatytą paskirties vietą. Maršrutizatoriai sujungia skirtingus tinklus ir leidžia perduoti duomenis iš vieno tinklo į kitą. Šis įrenginių derinys sudaro išsamų sistemos modelį, leidžiantį ištekliams veiksmingai bendrauti.



12 pav. Bendrinis vientisumo užtikrinimo modelis kompiuterinei sistemai

Dabar galime padetalizuoti esminius kuriamo modelio architektūros komponentus ir akcentuoti ryšio kanalus (žr. **13 pav.**).



13 pav. Sistemos architektūra

Aptariamos sistemos architektūrą sudaro keli skirtingi komponentai, kurių kiekvienas skirtas konkrečiam vaidmeniui atlikti. Architektūra veikia pagal tinklo modelį, jungiantį kelis įrenginius ir serverius į mechanizmą, užtikrinantį veiksmingą informacijos apie sistemos būseną perdavimą.

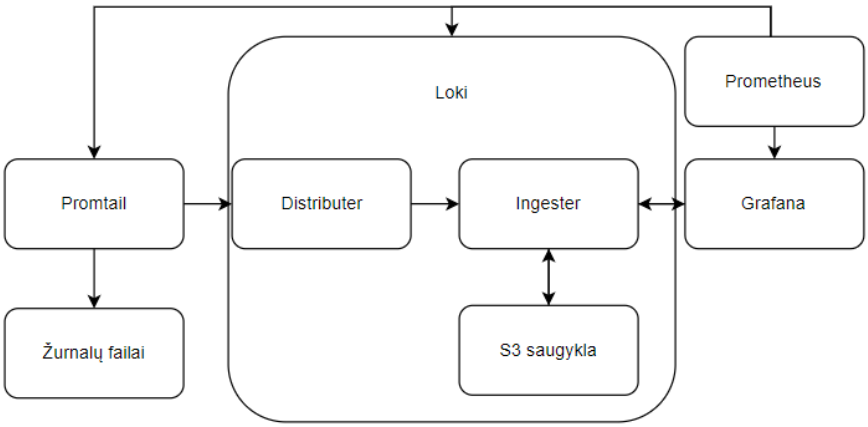
1. Būsenos serveris:

Pagrindinės būsenos serverio paskirtys yra kaupti duomenis apie konfigūruojamų serverių būsena, laikyti atsarginių failų kopijas ir pateikti žiniatinklį, kuris atvaizduoja renkama audito informacija ir dalį reikiamų metrikų. Kiekviena iš šių paskirčių galima detalizuoti giliau.

- Būsenos failų saugykla: Konfigūracijos būsenos informacija saugoma *Mongo* duomenų bazėje. Šis centralizuotas saugojimo mechanizmas leidžia lengvai pasiekti duomenis ir jais manipuliuoti. Taip pat užtikrinamas išsamus sistemos operacijų žurnalas, kuris gali būti labai svarbus trikčių šalinimui ir našumo optimizavimui. Taip pat būsenos serveriui siunčiami duomenys apima informaciją apie techninę įrangą, tinklo konfigūraciją, kernelio modulius, pasirinktinai tam tikrus katalogus, failus ir naudotojus. Ši informacija yra siunčiama iš pačių konfigūruojamų serverių, kuriuose yra įdiegta eilė *Cron* užduočių. *Cron* užduotys - tai automatiniai scenarijai, suplanuoti paleisti tam tikrais intervalais.

- Būsenos failų saugykla: Tai bus paprasta katalogų struktūra, kurioje bus saugomi duomenis. Ji dirbs tandemu su atsarginių kopijų programa, kuri bus instaliuota konfigūruojamoje įrangoje.

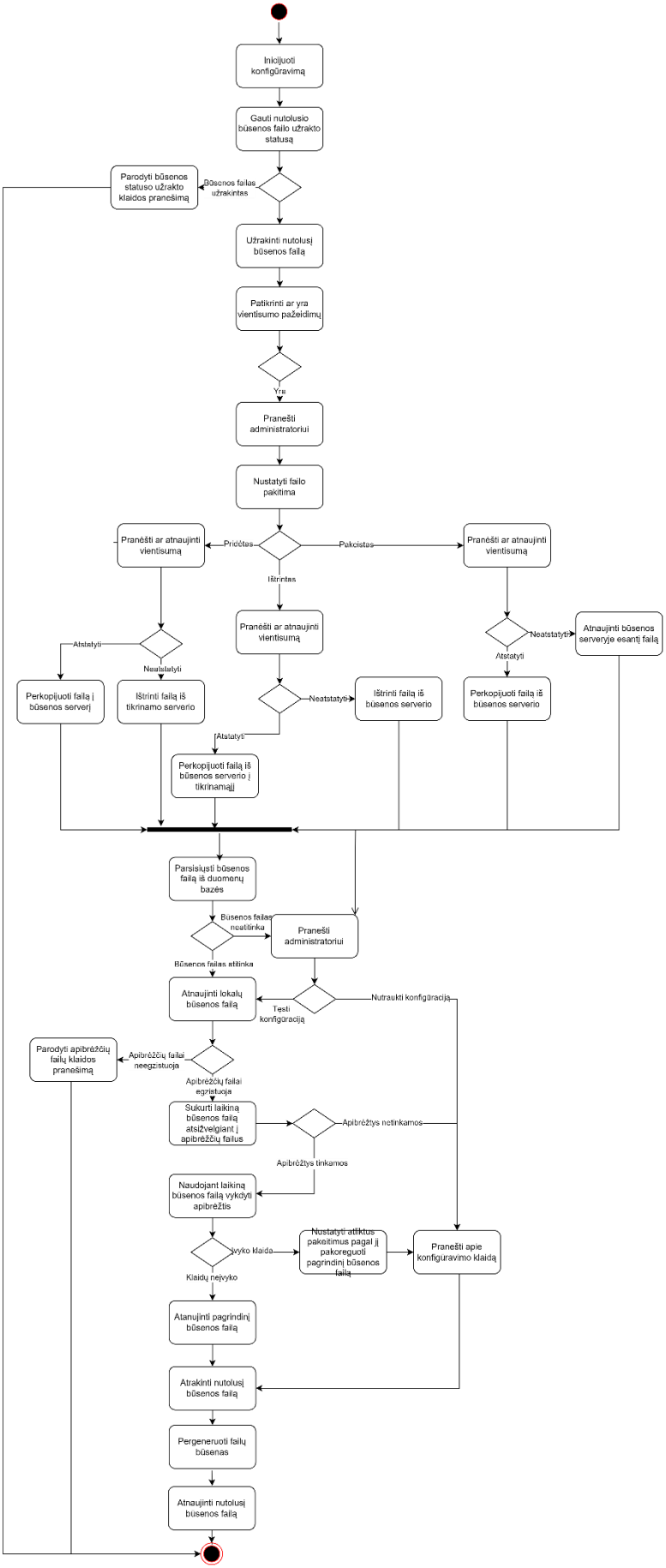
– Atvaizduojama audito informacija: Šiame darbe dėmesys skiriamas ir audito duomenų surinkimui (žr. **14 pav.**) ir jų vizualizacijai naudojant *Grafana* įrankį, kuris yra plačiai vertinamas dėl savo lankstumo ir efektyvumo vizualizuojant didelius duomenų kiekius. Audito duomenys renkami iš konfigūruojamų serverių, kuriuose įdiegta *Promtail* programinė įranga. *Promtail* šiame kontekste yra atsakingas už *Auditd* audito failų stebėjimą ir metrikų siuntimą į *Loki* duomenų saugyklą. *Loki* komponentas yra konfigūruotas kaip žurnalų agregavimo sistema, kuri veiksmingai tvarko ir saugo surinktus duomenis *AWS S3* paslaugoje, užtikrindama didelį duomenų patikimumą ir prieinamumą. Pagrindiniai du šios programos komponentai tai yra *Distributer*, kuris atlieka pirminį duomenų apdorojimą, nustato limitus ir atlieka kitas patvirtinimo operacijas prieš perduodant duomenis toliau. Kitas komponentas yra *Ingester*. Jis atsakingas už tinkamą duomenų perdavimą į saugyklą. *Loki* yra integruotas su *Prometheus* stebėjimo sistema, kuri fiksuoja ir analizuoja duomenis iš dviejų pagrindinių šaltinių: *Loki* ir *Promtail*. Ši integracija leidžia *Prometheus* surinkti plačią metrikų spektrą, įskaitant vidutinį žurnalų partijos priedų kiekį, bendrą baitų skaičių, duomenų segmentų kiekį. Šios metrikos yra svarbios nustatant duomenų apdorojimo efektyvumą ir sistemos veikimo stabilumą.



14 pav. Audito failų atvaizdavimo žiniatinklio programa

2. Konfigūracinis serveris:

– Konfigūracijos programa: Tai yra sistemos konfigūravimas tikrinant būseną (žr. **16 pav.**). Sukurtas išsamus konfigūracijos valdymo įrankis (žr. **15 pav.**) kuris sklandžiai integruojamas su *Ansible*.



15 pav. Sistemos konfigūravimas

Ši programa ne tik leidžia konfigūruoti aplinkas, bet ir pašalina esminį pačios *Ansible* apribojimą - konfigūracijų būsenos saugojimo trūkumą. Kad tai būtų įveikta, įrankis sukuria ir palaiko būsenų saugyklą nuotolinėje duomenų bazėje. Prieš vykdydama bet kokias konfigūravimo užduotis, ji gauna naujausią saugomą būseną ir palygina ją su dabartine konfigūracija, užtikrindama, kad būtų atliekami tik būtini pakeitimai.

Būsenos tikrinimai veikia trimis būsenos failų tikrinimo principais. Pirmasis yra vietinis failas, kuris yra *JSON* formato ir apibrėžia pageidaujamą išteklių būseną. Šiame faile yra visos specifikacijos, reikalingos norint sukurti arba atnaujinti išteklius, taip pat kiekvienai komandai saugomos atvirkštinės komandos. Tai užtikrina, kad bet kokie atlikti pakeitimai gali būti lengvai atšaukti, jei to prireiktų.

Antrasis principas yra paskutinės pritaikytos konfigūracijos failas. Tai taip pat *JSON* failas, kuris saugo anksčiausiai pritaikytą konfigūraciją. Šis failas leidžia įrankiui sekti, kokie pakeitimai buvo atlikti nuo pradinės konfigūracijos, ir palyginti juos su dabartine būsena, siekiant nustatyti, ar reikia atlikti naujų pakeitimų.

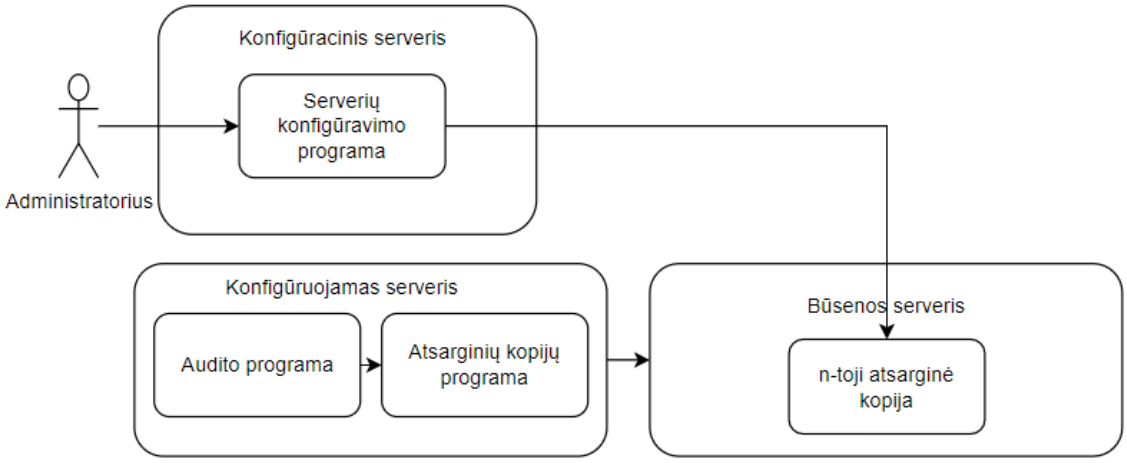
Trečiasis būsenos tikrinimo principas yra dabartinė konfigūracija. Dabartinė konfigūracija yra sugeneruota išteklių būsena, atspindinti esamą sistemos būklę. Ši konfigūracija yra lyginama su vietiniu ir paskutinės pritaikytos konfigūracijos failais, siekiant nustatyti, kokie pakeitimai yra būtini, ir užtikrinti, kad sistema visada būtų sinchronizuota su pageidaujama būsena.

Tokiu būdu programa užtikrina, kad visi konfigūracijų pakeitimai yra tikrinami ir valdomi nuosekliai, pašalinant nereikalingus veiksmus ir užtikrinant sistemos stabilumą bei saugumą. Nustatant, ką naudotojas ketina atnaujinti, programa palygina vietinį failą su naujausia pritaikyta konfigūracija. Norint nustatyti tikrąją infrastruktūros būseną ir sužinoti, ar po paskutinio konfigūracijos taikymo įvyko kokių nors papildomų pakeitimų. Po to programa taiko visus vietinio failo pakeitimus, kurie neprieštarauja nepriklausomiems konfigūracijos pakeitimams, sujungdama šių palyginimų rezultatus į naudojamąją konfigūraciją. Šie būsenos tikrinimai padeda išvengti konfigūracijos nukrypimo (angl. *drift*) tai dažna problema, kai dėl rankinių pakeitimų ir atnaujinimų esama aplinkos būsena skiriasi nuo infrastruktūros kodų apibrėžtos būsenos. Vis dėlto svarbu pažymėti, kad nors ši programa seka ir valdo konfigūracijos pakeitimus, ji turi apribojimų. Tam tikrose vietose negalime tiesiog įvykdyti atvirkštinės komandos. Taigi, tokių kraštutinių atvejų yra daug ir įvairių. Nepaisant to, bendras šios konfigūracijos valdymo programos funkcionalumas ir veiksmingumas gerokai padidina *Ansible* konfigūracijos valdymo ir stebėjimo galimybes.

16 pav. Būsenos ataskaitų palyginimas

3. Konfigūruojama įranga:

– Failų kopijos: Šis sprendimas (žr. **17 pav.**) saugo failus iš konfigūruojamų serverių ištikus vientisumo nesutapimams. Tarp būsenos serverio ir konfigūruojamų serverių naudojamas *rsync* servisas, kuris turi lankstų pritaikymą ir galimybę turėti failų įtraukimo ir neįtraukimo taisykles. Jei kuris nors iš failų jau yra nutolusioje sistemoje, jis gali būti atnaujintas perduodant tik skirtumus naudojant *remote-update* protokolą.



17 pav. Kopijų saugojimo aukšto lygio architektūra

– Audito programa: Audito programa bus naudojama *Auditd.* Ši programa turi ekstensyvų taisyklių rinkinį, kuris išsamiai atliks sistemos monitoringą. Vientisumo palaikymui taisyklių rinkinys sudarytas atsižvelgiant, bet neapsiribojant *MITRE ATT&CK* rekomendacijomis. Pilni taisyklių sąrašai aprašyti žemiau pateiktose lentelėse (žr. **1 lentelė**, **2 lentelė**).

Bendrai, rašant audito taisykles pravartu laikytis geriausiu rašymo praktiku. Vienas iš svarbiausių dalykų yra naudoti taisyklių žymeklius, kurie būna vėliau labai naudingi norint greitai rasti reikiamą informaciją apie atitinkamus įvykius. Šie žymekliai padeda efektyviau valdyti ir analizuoti audito duomenis, ypač kai reikia atlikti greitus užklausimus ar filtracijas pagal tam tikrus kriterijus.

Taip pat pravartu apjungti susijusius sisteminius iškvietimus į vieną taisyklę. Kiekvienas iškvietimas traktuojamas individualiai, o esant aktyvioms aplinkoms kiekvieno šių iškvietimų apribojimas išnaudoja dalį resursų. Pavyzdžiui, laiko keitimo operacijos gali būti sujungtos į vieną taisyklę: *-a always,exit -S time -S stime -S adjtimex -S gettimeofday -S settimeofday -S clock_settime -S set_time*. Taip pat failo pervadinimo operacijos gali būti sujungtos: *-a always,exit -S rmdir -S unlink -S rename_file*. Tokiu būdu sumažinamas taisyklių kiekis ir optimizuojamas resursų naudojimas.

Galiausiai, svarbu sekti tik tuos įvykius, kurie yra svarbūs konkrečiai sistemai. Pavyzdžiui, sistema, kuri yra uždara ir minimaliai naudoja internetą, turėtų būti įdėmiai stebima dėl naujų sudarytų tinklo ryšių. Tuo tarpu sistema, kuri yra viešai prieinama arba joje sukasi duomenų bazė, kiekvienas prisijungimas dažnu atveju sugeneruos kelis audito įrašus, stipriai apkraudamas sistemą. Todėl audito taisyklės turėtų būti pritaikytos prie konkrečios sistemos poreikių, siekiant išvengti nereikalingo apkrovimo ir užtikrinti efektyvų stebėjimą.

Tam tikrose sistemose gali pasirodyti neįmanoma stebėti visų audituojamų įrašų tipų dėl daugybės įrašų rūšių, dėl to apart stebimų žurnalų taisyklių taip pat yra ir neįtraukimo taisyklės (žr. **3 lentelė**). Šios taisyklės riboja audito žurnalų kiekį bereikalinga informacija.

1 lentelė Procesų audito taisyklės

Taisyklės pavadinimas	**Taisyklė**
Veiksmai su virtualiu atminties failu (angl. *swap*) failu ir jo konfigūracijomis.	*-a always,exit -F arch=b64 -S swapon -S swapoff -F auid!=-1 -k swap*
Naujų procesų sukūrimas	*-a always,exit -F arch=b32 -S exit,fork,execve,clone,vfork,exit_group -F key processes*
Prijungimo ir atjungimo (angl. *mount*) operacijos	*-a always,exit -F path=/sbin/mount.nfs -F perm=x -F auid>=500 -F auid!=4294967295 -k mount* *-a always,exit -F path=/usr/sbin/mount.nfs -F perm=x -F auid>=500 -F auid!=4294967295 -k mount* *-a always,exit -F arch=b64 -S mount -S umount2 -F auid!=-1 -k mount*
Laiko keitimo operacijos	*-a exit,always -F arch=b32 -S adjtimex -S settimeofday -S clock_settime -k time*

	-a exit,always -F arch=b64 -S adjtimex -S settimeofday -S clock_settime -k time *-a always,exit -F arch=b32 -S clock_settime -k time* *-a always,exit -F arch=b64 -S clock_settime -k time* *-w /etc/localtime -p wa -k time* *-a always,exit -F arch=b32 -S utimes -k time* *-a always,exit -F arch=b64 -S utimes -k time* *-a always,exit -F arch=b32 -S utimensat -k time* *-a always,exit -F arch=b64 -S utimensat -k time* *-a always,exit -F arch=b64 -F uid!=ntp -S adjtimex -S settimeofday -S clock_settime -k time*
Naujų tinklo ryšių sudarymai	*-a always,exit -F arch=b64 -S connect -k network* *-a always,exit -F arch=b64 -S connect -F a2=16 -F success=1 -k network* *-a always,exit -F arch=b64 -S connect -F a2=28 -F success=1 -k network* *-a always,exit -F arch=b32 -S connect -k network* *-a always,exit -F arch=b32 -S connect -F a2=16 -F success=1 -k network* *-a always,exit -F arch=b32 -S connect -F a2=28 -F success=1 -k network* *-a always,exit -F arch=b32 -S socketcall -F a0=2 -k network* *-a always,exit -F arch=b64 -S socket -k network* *-a always,exit -F arch=b32 -S socket -k network* *-a always,exit -F arch=b64 -S bind -k network* *-a always,exit -F arch=b32 -S bind -k network* *-a always,exit -F arch=b64 -S accept -k network* *-a always,exit -F arch=b32 -S accept -k network* *-w /etc/hosts -p wa -k network* *-w /etc/sysconfig/network -p wa -k network* *-w /etc/sysconfig/network-scripts -p w -k network* *-w /etc/network/ -p wa -k network* *-a always,exit -F dir=/etc/NetworkManager/ -F perm=wa -k network* *-a always,exit -F arch=b64 -S connect -k network* *-a always,exit -F arch=b32 -S connect -k network*
Duomenų suspaudimo operacijos	*-w /usr/bin/zip -p x -k compression* *-w /usr/bin/gzip -p x -k compression* *-w /usr/bin/tar -p x -k compression* *-w /usr/bin/bzip2 -p x -k compression* *-w /usr/bin/lzip -p x -k compression* *-w /usr/local/bin/lzip -p x -k compression* *-w /usr/bin/lz4 -p x -k compression*

	-w /usr/local/bin/lz4 -p x -k compression *-w /usr/bin/lzop -p x -k compression* *-w /usr/local/bin/lzop -p x -k compression*
Paketų instaliavimai	*-w /usr/bin/dpkg -p x -k packages* *-w /usr/bin/apt-add-repository -p x -k packages* *-w /usr/bin/apt-get -p x -k packages* *-w /usr/bin/aptitude -p x -k packages* *-w /usr/bin/rpm -p x -k packages* *-w /usr/bin/yum -p x -k packages* *-w /usr/bin/dnf -p x -k packages* *-w /sbin/yast -p x -k packages* *-w /sbin/yast2 -p x -k packages* *-w /bin/rpm -p x -k packages* *-w /usr/bin/zypper -k packages* *-w /usr/bin/wajig -p x -k packages* *-w /usr/bin/snap -p x -k packages* *-w /usr/bin/pip -p x -k packages* *-w /usr/local/bin/pip -p x -k packages* *-w /usr/bin/pip3 -p x -k packages* *-w /usr/local/bin/pip3 -p x -k packages* *-w /usr/bin/pipx -p x -k packages* *-w /usr/local/bin/pipx -p x -k packages* *-w /usr/bin/npm -p x -k packages* *-w /etc/pacman.conf -p x -k packages* *-w /etc/pacman.d -p x -k packages* *-w /usr/bin/luarocks -p x -k packages* *-w /usr/bin/gem -p x -k packages* *-w /usr/bin/cpan -p x -k packages*
Branduolio modulių pakitimai	*-a always,exit -F perm=x -F auid!=-1 -F path=/sbin/insmod -k modules* *-a always,exit -F perm=x -F auid!=-1 -F path=/sbin/modprobe -k modules* *-a always,exit -F perm=x -F auid!=-1 -F path=/sbin/rmmod -k modules* *-a always,exit -F arch=b64 -S finit_module -S init_module -S delete_module -F auid!=-1 -k modules*

2 lentelė Failų audito taisyklės

Taisyklės pavadinimas	**Taisyklė**
Cron konfigūracijos ir suplanuoti darbai	*-w /etc/cron.allow -p wa -k cron* *-w /etc/cron.deny -p wa -k cron*

	-w /etc/cron.d/ -p wa -k cron *-w /etc/cron.daily/ -p wa -k cron* *-w /etc/cron.hourly/ -p wa -k cron* *-w /etc/cron.monthly/ -p wa -k cron* *-w /etc/cron.weekly/ -p wa -k cron* *-w /etc/crontab -p wa -k cron* *-w /var/spool/cron/ -p wa -k cron*
Naudotojų, grupių ir slaptažodžių failai	*-w /etc/group -p wa -k passwd* *-w /etc/passwd -p wa -k passwd* *-w /etc/gshadow -k passwd* *-w /etc/shadow -k passwd* *-w /etc/security/opasswd -k passwd*
Sisteminiai konfigūraciniai failai	*-w /etc -k system_config_files*
Pasikeitimai tinklo konfigūravimo failuose	*-w /etc/hosts -p wa -k network* *-w /etc/sysconfig/network -p wa -k network* *-w /etc/sysconfig/network-scripts -p w -k network* *-w /etc/network/ -p wa -k network* *-a always,exit -F dir=/etc/NetworkManager/ -F perm=wa -k network*
Pasikeitimai *PAM* konfigūraciniuose failuose	*-w /etc/pam.d/ -p wa -k pam* *-w /etc/security/limits.conf -p wa -k pam* *-w /etc/security/limits.d -p wa -k pam* *-w /etc/security/pam_env.conf -p wa -k pam* *-w /etc/security/namespace.conf -p wa -k pam* *-w /etc/security/namespace.d -p wa -k pam* *-w /etc/security/namespace.init -p wa -k pam*
Pašto konfigūraciniai failai	*-w /etc/aliases -p wa -k mail* *-w /etc/postfix/ -p wa -k mail* *-w /etc/exim4/ -p wa -k mail*
SSH konfigūraciniai failai	*-w /etc/ssh/sshd_config -k sshd* *-w /etc/ssh/sshd_config.d -k sshd*
Pasikeitimai sisteminiuose dvejetainiuose programų failuose	*-w /usr/sbin -k system_progam_files* *-w /usr/bin -k system_ progam _files* *-w /usr/local/bin -k system_progam_files*
SELinux taisyklių pasikeitimai	*-w /etc/selinux/ -p wa -k selinux*
Profilių ir terminalų konfigūraciniai pakeitimai	*-w /etc/profile.d/ -p wa -k shell_profiles* *-w /etc/profile -p wa -k shell_profiles* *-w /etc/shells -p wa -k shell_profiles* *-w /etc/bashrc -p wa -k shell_profiles*

	-w /etc/csh.cshrc -p wa -k shell_profiles *-w /etc/csh.login -p wa -k shell_profiles* *-w /etc/fish/ -p wa -k shell_profiles* *-w /etc/zsh/ -p wa -k shell_profiles*
Sudoers failas	*-w /etc/sudoers -p wa -k sudoers* *-w /etc/sudoers.d/ -p wa -k sudoers*

3 lentelė Neįtraukimo taisyklės

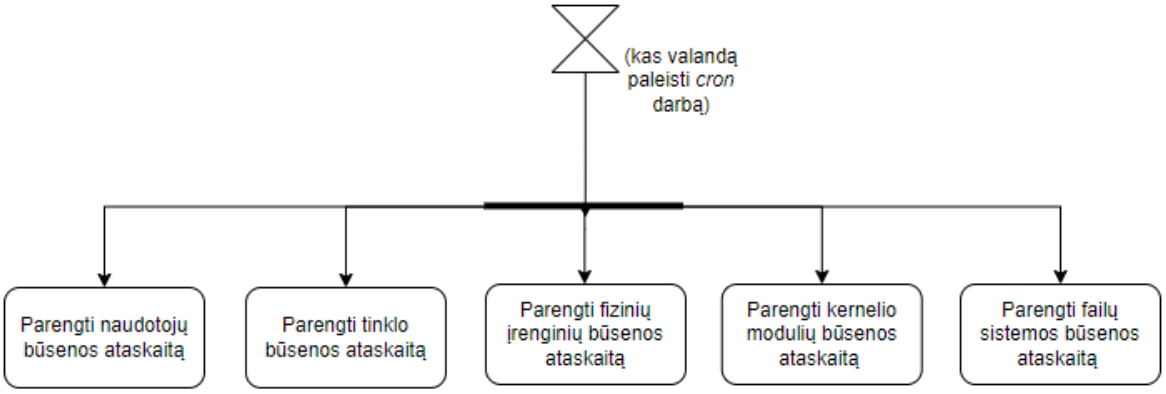
Taisyklės pavadinimas	**Taisyklė**
Ignoruoti klaidas	*-i*
Filebeat servisas	*-a never,exit -F arch=b64 -F path=/opt/filebeat -k filebeat*
Didelės apimties įvykių filtras	*-a never,exit -F arch=b64 -F dir=/dev/shm -k sharedmemaccess* *-a never,exit -F arch=b64 -F dir=/var/lock/lvm -k locklvm*
Virtualių mašinų pranešimai	*-a exit,never -F arch=b64 -S all -F exe=/usr/bin/vmtoolsd*
Viešų resursų pasiekimui naudojamas kriptografinio rakto indentifikatorius	*-a always,exclude -F msgtype=CRYPTO_KEY_USER*
Chrony servisas	*-a never,exit -F arch=b64 -S adjtimex -F auid=-1 -F uid=chrony -F subj_type=chronyd_t*
Cron darbų žurnalai	*-a never,user -F subj_type=crond_t* *-a never,exit -F subj_type=crond_t*
Vienas iš *auditd* įrašų tipų, parodantis darbo katalogo įvykio vietą	*-a always,exclude -F msgtype=CWD*

– Vientisumo programos: Programos, kurios stebės ir saugos infrastruktūros pokyčius. Ši programa dalinai persidengia, bet ir padengia dalį panaudojimo atvejų, kurių audito programa nepajėgia apimti.

Šios programos apima kelias svarbias būsenos ataskaitas. Pirma, tai naudotojų būsenos ataskaita, kuri stebi ir registruoja visus naudotojų veiksmus bei pokyčius jų paskyrose. Tinklo būsenos ataskaita teikia informaciją apie tinklo konfigūraciją ir ryšius, užtikrindama, kad bet kokie nenormalūs pokyčiai būtų pastebėti ir užfiksuoti. Kernelio būsenos ataskaita yra svarbi, nes ji stebi operacinės sistemos branduolio būseną ir užtikrina, kad visi branduolio pokyčiai

būtų registruojami. Galiausiai, failų sistemos būsenos ataskaita yra atsakinga už failų sistemos stebėjimą, registruojant bet kokius failų kūrimo, keitimo ar trynimo veiksmus.

Šios programos, kaip anksčiau minėta, dirbs išvien su audito programa. Jos atsakingos už žurnalų agregavimą, pranešimų išsiuntimą ir vientisumo būsenų ataskaitų sudarymą ir saugojimą, Kadangi šios programos veiks pasyviu būdu (žr. **18 pav.**) tai laikotarpyje tarp pakeitimų gali būti tokių pat ir atkeitimų, todėl būtina sekti dalį vientisumo ir audito programoje pasinaudojant sudarytu sisteminiu iškvietimų žurnalais.



18 pav. Pasyvios būsenos naujinimo programos

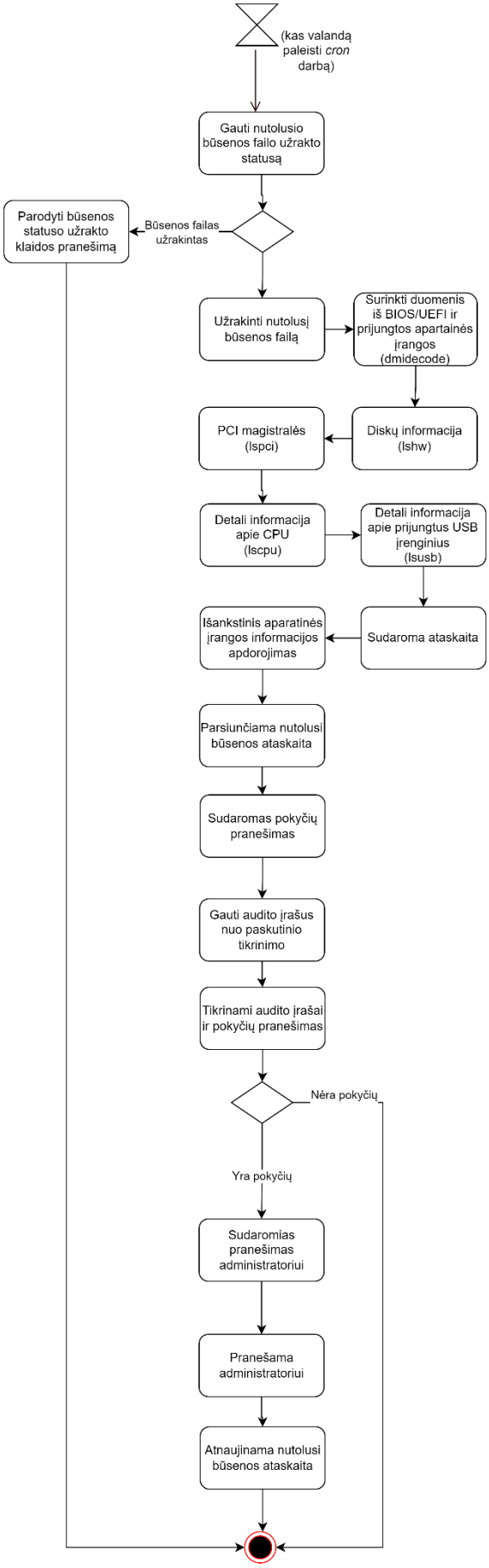
Dabar galime aptarti detaliau kiekvieną *cron* darbą. Kokią informaciją ir kaip ji bus surenkama.

Parengti fizinių įrenginių būsenos ataskaitą:

Šiai ataskaitai parengti bus pagrinde naudojamas *dmidecode*. Jos paskirtis - pateikti išsamią ir konkrečią informaciją apie sistemos konfigūraciją ir aparatinės įrangos komponentus. Ši komanda pasirinkta dėl to, nes pateikia išsamiausia ir patikimiausia sistemos būsenos ataskaitą. Kadangi *dmidecode* nerenka tam tikrų duomenų kaip diskų informaciją tai jai talkins *lshw* komanda. Toliau bus naudojamos specializuotos komandos, kurios detaliausiai parodo informaciją. *Lspci* – pateikia informaciją apie populiariausias PCI magistrales kaip *USB, PCIe, SATA, Thunderbolt* ir prie jų pridėtus įrenginius. *Lscpu* – informacija apie procesorių ir *lsusb* apie prijungtus USB įrenginius. Taigi bendrai šios komandos (žr. **19 pav.**) apima visą reikiamą sistemos informaciją. Bus apimami tiek virtualūs įrenginiai kaip tinklo sąsajos, tiek ir fiziniai įrenginiai prijungti per daugelį populiarių magistralių kaip: *USB, PCIe, SATA, Thunderbolt* ir panašiai. Su šiomis komandomis bus renkama:

- OS sistemos versija.
- BIOS versija, išleidimo data ir charakteristikos.
- Duomenys apie procesorių kaip autorius, versija.
- Įdiegtų atminties modulių dydis, tipas, greitis ir konfigūracija.
- Pagrindinės plokštės gamintojas, gaminio pavadinimas ir serijos numeris.
- Maitinimo šaltinio tipas ir būsena.
- Ventiliatoriai ir jutikliai su atitinkamomis būsenomis.

- Informacija apie diską kaip modelis, talpa, informacija apie skyrius ir disko valdiklio informacija.
- USB valdiklio modelis, prievadai ir prijungti USB įrenginiai.
- Garso plokštės modelis ir garso galimybės.

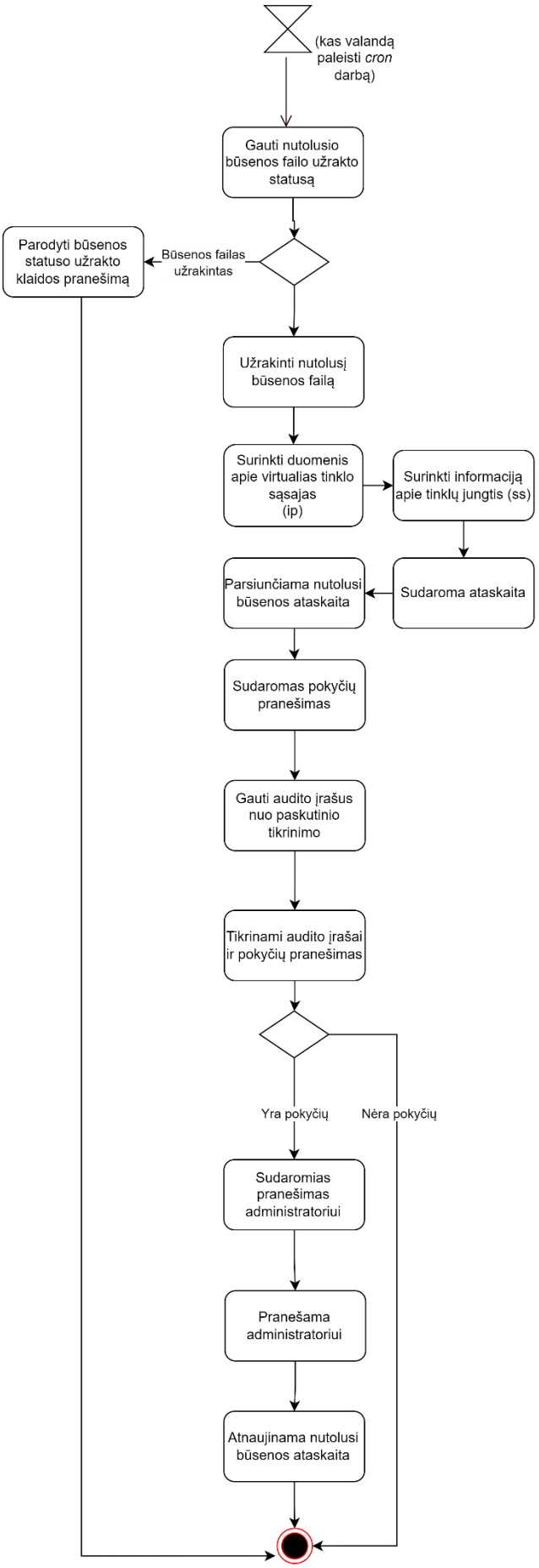


19 pav. Parengti fizinių įrenginių būsenos ataskaita veiklos diagrama

Parengti naudotojų būsenos ataskaitą:

Renkant informaciją apie naudotojus sistemoje (žr. **20 pav.**), svarbu užtikrinti, kad būtų gauta išsami ir tiksli duomenų apžvalga, kuri padeda stebėti ir valdyti vartotojų teises bei prieigą prie išteklių. Šiuo atveju, norint efektyviai rinkti informaciją, naudojami sistemų failai:

- */etc/passwd*: Šiame faile saugoma informacija apie kiekvieną sistemos naudotoją. Čia yra įrašyti naudotojo vardas, naudotojo ID (UID), grupės ID (GID), naudotojo aprašymas, namų katalogas ir terminalą.
- */etc/group*: Failas, kuriame yra informacija apie kiekvieną grupę.
- */etc/gshadow*: Failas naudojamas saugoti grupių slaptažodžių informacijai.
- */etc/shadow*: Čia saugomi saugoti naudotojų slaptažodžiai ir slaptažodžių keitimo informacija.



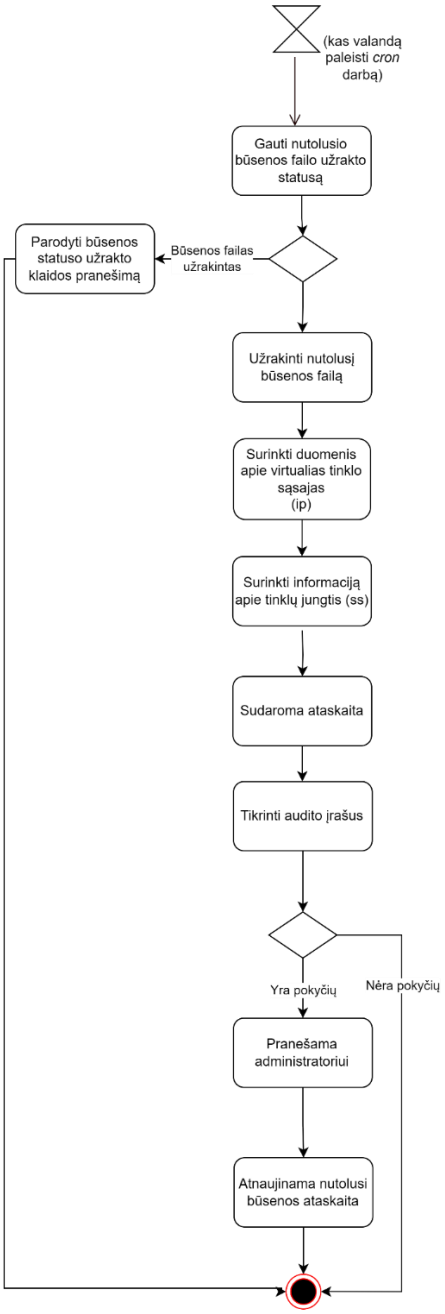
20 pav. Parengti naudotojų būsenos ataskaita veiklos diagrama

Parengti tinklo būsenos ataskaitą:

Norint patikrinti dabartinę tinklo konfigūracijos būseną bus naudojama komanda *ip a*. Ji pateikia glaustą ir išsamią tinklo sąsajų ir su jomis susijusių IP adresų apžvalgą. Paleidę *ip a*, galime išgauti šią informaciją:

- Parodyti informaciją apie tinklo sąsajas, įskaitant jų pavadinimus, *MAC* adresus ir būseną.
- Ištraukti kiekvienai sąsajai priskirtus IP adresus ir potinklio kaukes.
- Transliavimo adresai, MTU.

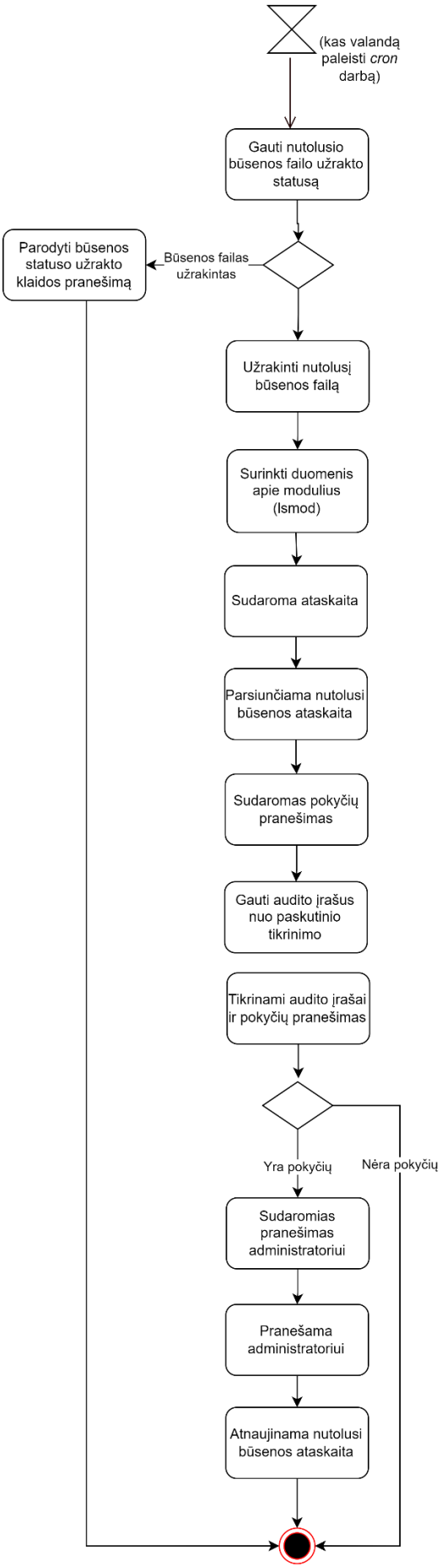
Taip pat, pasinaudosime *ss* komandą kuri ištrauks visas sudarytas tinklų jungtis. Šis parengimas (žr. **21 pav.**) turėtų sudaryti gerą tinklo vaizdą tolesniems vientisumo tikrinimams.



21 pav. Parengti tinklo būsenos ataskaitą veiklos diagrama

Parengti kernelio modulių būsenos ataskaitą:

Norint gauti informaciją apie kernelio modulius bus naudojama komanda *lsmod (*žr. **22 pav.**). Komandos *lsmod* išvestyje pateikiamas kernelio modulių sąrašas ir tam tikra informacija apie kiekvieną modulį, įskaitant jo pavadinimą, dydį ir šiuo metu naudojamo modulio skaičių kitų programų ar modulių pagal hierarchinę struktūrą.



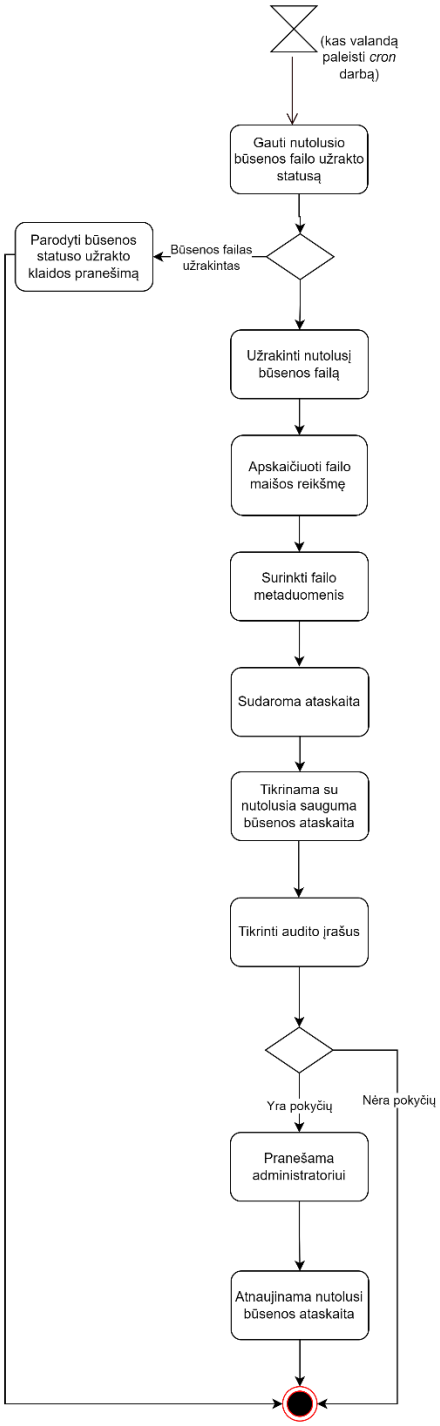
22 pav. Parengti kernelio modulių būsenos ataskaita veiklos diagrama

Parengti failų sistemos būsenos ataskaitą:

Visi failai bus saugomi pasinaudojus *md5* komandą. *Md5* naudojamas failų vientisumui patikrinti. Jis generuoja fiksuoto ilgio unikalią tam tikro failo maišos vertę. Toks sprendimas (žr. **23 pav.**) leis sutapyti disko vietos, kadangi nereikės laikyti didelių failų ir vientisumo tikrinimai užtruks mažiau laiko.

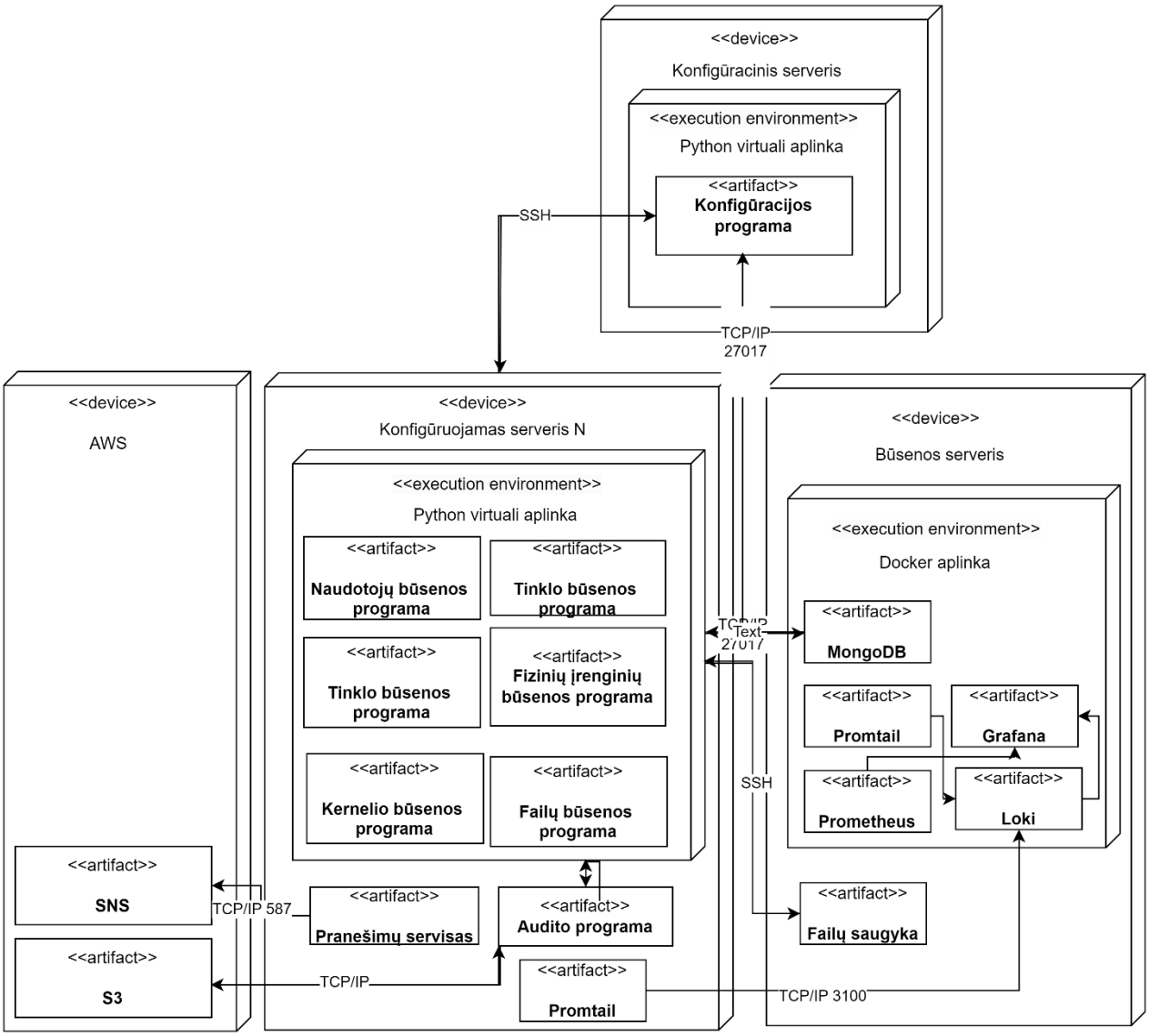
Taip pat, bus saugoma ir failų metaduomenys:

1. Failų atributai:
   - Failo dydis: Bendras failo dydis baitais.
   - Teisės: Įrašomos skaitymo, rašymo ir vykdymo teisės, kurios nurodo, kas gali pasiekti ir keisti failą.
   - Savininkas: Informacija apie failo savininką ir grupę, kuriai priklauso failas.
   - Jungčių skaičius
   - Failo tipas
2. Failo vieta: Pilnas kelias į failo vietą failų sistemoje.



23 pav. Parengti failų sistemos būsenos ataskaita veiklos diagrama

Bendra, reali vientisumo modelio sistemos diegimo diagrama aptariama (žr. **24 pav.**), kuri užtikrina efektyvų sistemos valdymą.



24 pav. Diegimo diagrama

2.3. Išvados

1. Sukurtas modelis leidžiantis išlaikyti kompiuterinės sistemos vientisumą.
2. Pasiūlytas modelis ne vien seką sistemos būseną, bet sugebą ją ir konfigūruoti.
3. Ši sistema yra nepriklausoma nuo konfigūruojamų sistemų ir leidžia lengvai išplėsti mastelį.
4. Dalis servisų konteinerizuoti arba panaudoti virtualiose aplinkose taip leidžiant disponuoti sistema tarp skirtingų infrastruktūrų dar lengviau.

3. Kompiuterinės sistemos realizacija ir tyrimas

3.1. Kompiuterinės sistemos vientisumo modelio aprašymas

Sukurtas modelis yra sudarytas iš dviejų dalių. Pirmoji dalis yra atsakinga už reguliarius sistemos vientisumo patikrinimus. Tai atliekama naudojant Python programas, kurios periodiškai tikrina specifinius sistemos katalogus, failus ir komandas nustatydamos, ar sistema buvo vienaip ar kitaip pakeista. Pastaroji dalis įgyvendinama pasinaudojant *Cron* (žr. **25 pav.***)* užduotimis. Jos iškviečia atitinkamą programą, kurių viso yra 5 (žr. **18 pav.**). Kiekviena jų saugo atitinkamus sistemos metaduomenis *MongoDB* duomenų bazėje ir įvykus sistemos vientisumo pakitimui būna suformuojamas pranešimas, kuris toliau būną panaudojamas formuojant el. pranešimą, kuris bus naudojamas informuoti administratorių. Kadangi šios programos aktyviai netikrina pakitimų, o susivykdo tik tam tikrais nustatytais momentais, kiekviena šių programų turi atitinkamas *auditd* taisykles. *Auditd* padeda išvengti įvykių įvykusių tarp tikrinimo laikų. Pastarosios programos audito ataskaitos taip pat yra pridedamos į suformuotą pranešimą administratoriui. Šios programos privalo būti įrašytos į kiekvieną tikrinamą kompiuterinę sistemą, užtikrintas reikalingų programų automatinis paleidimas.

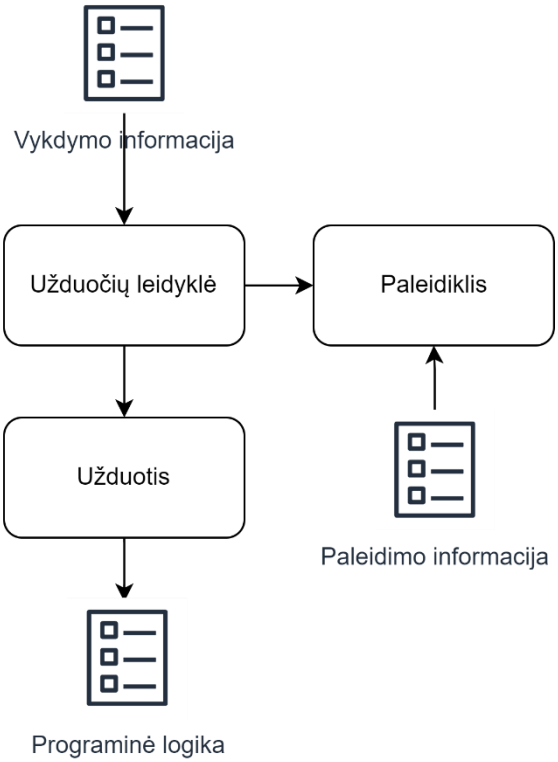
Antroji dalis užtikrina kompiuterinės sistemos vientisumo išlaikymą per visą jos eksploatavimo laikotarpį. Šioje dalyje yra parašytą *Python* programa kuri tandemu dirba su *Ansible* biblioteka. Ši programa sudaryta iš neriboto skaičiaus galimų *Ansible* scenarijų ir 20 atstatymo scenarijų (žr.). Šie atstatymo scenarijai svarbūs tuo, nes jie išplečia funkcionalumu *Ansible* dalį, nes ji savyje nesaugo jokios būsenos apie konfigūruojamą aplinką, dėl to tikimybė palikti užsilikusius pasenusius ir pažeidžiamus programų paketus, konfigūracijas ar ką kitą yra didelė. Taip pat ši vientisumo programa dirba išvien su failų pakitimo programa ir esant poreikiui sugeba atstatyti ar atnaujinti sistemą pagal pasirinktus kriterijus.

Apart vientisumo tikrinimo dalių taip pat sukurti grafikai (žr. **14 pav.**) lengvesnei audito duomenų peržiūrai pasinaudojant *Grafana, Loki, Promtail* ir *Prometheus.*

Visi prototipo galimi funkcionalumai:

- Užtikrinama, kad sinchronizuojant failų pakeitimus nuotolinėse sistemose būtų sukurtos arba palaikomos reikiamos katalogų struktūros, taip užtikrinant sklandų failų valdymą įvairiose aplinkose.
- Ryšiai su *MongoDB* duomenų bazėmis, kad būtų galima saugoti ir tvarkyti duomenis įvairiems stebėjimo tikslams (naudotojo veiklai, aparatinės įrangos būsenoms, tinklo įvykiams ir failų pokyčiams).
- Paleidžiamos *Python* programos sistemos informacijai rinkti ir konkrečioms užduotims atlikti, įskaitant pakeitimų atšaukimo komandų vykdymą ir audito žurnalų tvarkymą.
- Periodiškai stebi ir lygina įvairių sistemos komponentų (failų, aparatinės įrangos, tinklo) dabartinę būseną su ankstesnėmis būsenomis, kad nustatytų pokyčius.
- Apibrėžia ir tvarko audito taisykles, skirtas svarbiausiems sistemos failams ir veiksmams stebėti, įskaitant konkrečias išimtis ir didelės apimties įvykių filtrus.
- Renka ir analizuoja audito žurnalus, kad stebėtų sistemos veiklą, susijusią su naudotojo veiksmais, tinklo pokyčiais ir kitais svarbiais įvykiais.

- Lygindamas esamus metaduomenis su anksčiau išsaugotais duomenimis, nustato ir kategorizuoja pokyčius (pridėtus, pašalintus, pakeistus).
- Generuoja aptiktų pakeitimų ataskaitas ir el. paštu pranešta administratoriams apie svarbius pakeitimus ir įvykius.
- Renka ir saugo nurodytų katalogų failų metaduomenis, įskaitant *md5* maišos skaičiavimą, ir stebi failų turinio, leidimų ir kitų atributų pokyčius.
- Naudojant įvairias komandas (*dmidecode, lshw, ip a, ss -tulnp*) renkama išsami aparatinės įrangos ir tinklo informacija ir konsoliduojami stebėjimo duomenys.
- Įdiegiami rakinimo mechanizmai, užtikrinantys, kad vienu metu veiktų tik vienas stebėsenos proceso egzempliorius, ir sinchronizuoja pokyčius tarp vietinių ir nuotolinių sistemų, įskaitant būtinų failų perkėlimų ir atnaujinimų vykdymą.
- Audito žurnalai atvaizduojami trečiųjų šalių įrankyje per žiniatinklį.



25 pav. Tipinė būsenos programa

Vykdymo informacija: Tai gali atspindėti vykdomąją informaciją, pavyzdžiui, konfigūracijos duomenis, būsenos informaciją ar paleidimo metaduomenis, kuriuos naudoja sistema užduočių vykdymui sekti.

Užduočių leidyklė: Tai komponentas ar modulis, kuris yra atsakingas už užduočių paleidimą. Jis gali būti sukonfigūruotas paleisti užduotis pagal nustatytą grafiką ar įvykius.

Paleidiklis: Šis elementas veikia kaip inicijuojantis mechanizmas užduočių leidyklėje, leidžiantis aktyvuoti užduotis tam tikru laiku arba atsakyti į tam tikrus įvykius. Paleidiklis gali būti suprogramuotas naudojant cron išraiškas arba kitokias laiko nustatymo schemas.

Paleidimo laikas: Tai, greičiausiai, yra susiję su paleidiklio komponentu ir nurodo konkrečius laiko parametrus, kada užduotis turėtų būti paleista.

Užduotis: Tai konkrečios veiklos ar operacijos, kurias sistema turi atlikti. Užduotys gali būti įvairios, nuo paprastų scenarijų vykdymo iki sudėtingų procesų, kurie apdoroja duomenis ar atlieka kitokius kompiuterinius darbus.

Programinė logika: Tai nurodo programinės įrangos dalį, kuri apima visus algoritmus ir procedūras, reikalingas užduoties atlikimui. Tai gali apimti verslo logiką, duomenų apdorojimą, ryšių su kitais sistemos komponentais valdymą ir kt.

3.2. Kompiuterinės sistemos vientisumo modelio naudojama įranga

Projekte naudojama įranga yra pritaikyta efektyviam modelio tyrimui.

4 lentelė Programinė įranga

Pavadinimas	**Versija**
Docker	25.0.4
Prometheus	2.45.5
MongoDB	8.0.0
Grafana	10.0.13
Visual Studio Code	1.89
Loki	2.9.2
Promtail	2.9.2
Python	3.12.3
Ansible	2.14.16
Bash	5.2

5 lentelė Python bibliotekos

Pavadinimas	**Versija**
Ansible	6.7.0
Ansible-core	2.13.12
Cffi	1.16.0
Cryptography	41.0.4
Deepdiff	6.7.1
Jinja2	3.1.2
MarkupSafe	2.1.3
Ordered-set	4.1.0
Packaging	23.2
Bcrypt	4.1.3

Boto3	1.34.104
Botocore	1.34.104
Dnspython	2.4.2
Jmespath	1.0.1
Paramiko	3.4.0
Pycparser	2.22
Pymongo	4.6.0
PyNaCl	1.5.0
Python-dateutil	2.9.0.post0
S3transfer	0.10.1
Urllib3	1.26.18
Six	1.16.0

Programinė įranga (žr. **4 lentelė**, **5 lentelė**) buvo naudojama naujausios tuo metu buvusios versijos apart *Grafana*, kadangi nuo 11 versijos buvo atsisakyta *Angular*, dėl to dalis grafikų nebeveikė.

6 lentelė Fizinė ir virtuali aparatinė įranga

Įranga	**Kiekis**	**OS**	**Paskirtis**	**Specifikacija**
Virtualus serveris	10	Ubuntu 20.04	Konfigūruojama įranga	2 vCPU, 4GB
Nešiojamas kompiuteris	1	Ubuntu 20.04	Konfigūracinis programa	6 CPU, 32 GB
Virtualus serveris	1	Ubuntu 20.04	Būsenos serveris	2 vCPU, 4GB, 512GB

7 lentelė Konfigūracinis serveris

Nešiojamas kompiuteris (Konfigūracinė programa)	
Procesorius	Intel Core i7-9750H
Atmintis	32GB
Branduolių skaičius	6
Taktinis dažnis	4.5 GHz
Pralaidumas	2.4 Gibps

8 lentelė Konfigūruojama įranga

Virtualus serveris (Konfigūruojama įranga)	
Procesorius	Intel Skylake E5 2686 v5
Atmintis	4GB
Branduolių skaičius	2
Taktinis dažnis	3.1 GHz
Pralaidumas	5 Gibps

9 lentelė Būsenos serveris

Virtualus serveris (Būsenos serveris)

Procesorius	Intel Skylake E5 2686 v5
Atmintis	4GB
Branduolių skaičius	2
Taktinis dažnis	3.1 GHz
Pralaidumas	5 Gibps

Visi virtualūs serveriai yra patalpinti tokio paties tipo (žr. **8 lentelė**, **9 lentelė**). Jie buvo patalpinti *AWS* aplinkoje pasirinkus *t3.medium* dydį. Šių virtualių serverių naudojimas yra svarbus norint užtikrinti, kad infrastruktūros konfigūracijos ir valdymo sprendimai būtų patikimi ir universalūs, veikiantys skirtingose operacinėse sistemose ir aplinkose. Tuo tarpu konfigūracinė programa buvo paleista iš nešiojamojo kompiuterio (žr. **7 lentelė**), kuri pakankamai galinga atlikti visus norimus veiksmus.

3.3. Kompiuterinės sistemos vientisumo modelio testavimas, apkrovos ir greitaveikos tyrimas

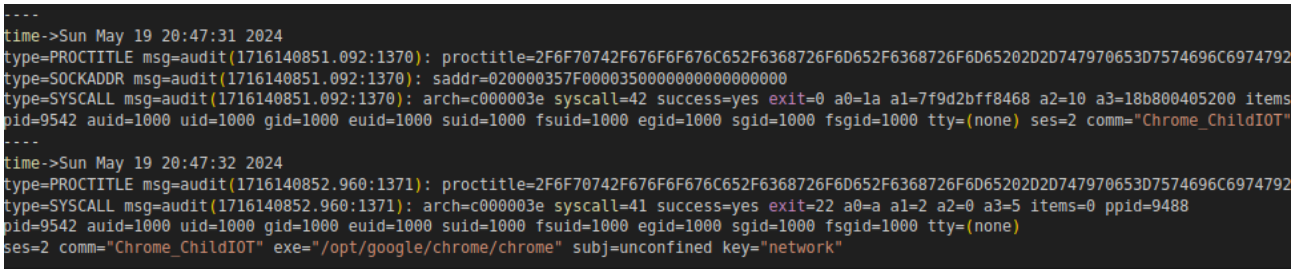
Pirmajame tyrimo teste yra atliekamas tinklo būsenos tikrinimas ir pateikiamas sugeneruotas pokyčių pranešimas (žr. **26 pav. 27 pav. 28 pav.**). Antrajame tikrinime stebimas naudotojų vientisumo pokytis, kadangi čia skirtumų įvairovė nėra ypatingai didelė, galime atlikti detalesnį įvardijimą kas konkrečiai pasikeitė (žr. **31 pav.**). Kitos likusios vientisumo tikrinimo programos veikia panašiu principu. Jas galime peržiūrėti **29 pav**. **30 pav. 32 pav. 33 pav. 34 pav.**

```
Network event 'socket' occurred initiated by '/opt/google/chrome/chrome'.
Network event 'socket' occurred initiated by '/usr/bin/ss'.
Network event 'connect' occurred initiated by '/snap/code/159/usr/share/code/code'.
Network event 'socket' occurred initiated by '/usr/bin/ip'.
Network event 'socket' occurred initiated by '/usr/bin/python3.8'.
Network event 'socket' occurred initiated by '/usr/sbin/ausearch'.
Network event 'connect' occurred initiated by '/usr/sbin/ausearch'.
Network event 'socket' occurred initiated by '/snap/code/159/usr/share/code/code'.
Network event 'bind' occurred initiated by '/usr/bin/ip'.
Network event 'bind' occurred initiated by '/opt/google/chrome/chrome'.
Network event 'connect' occurred initiated by '/usr/bin/python3.8'.
Network event 'bind' occurred initiated by '/usr/bin/ss'.
Network event 'connect' occurred initiated by '/opt/google/chrome/chrome'.
```

26 pav. Suformuoti tinklo įvykiai iš audito failų

```
IP Address Changes:
--- IP Addresses (old)
+++ IP Addresses (new)
@@ -49,3 +49,5 @@
     link/ether 0a:00:27:00:00:01 brd ff:ff:ff:ff:ff:ff
 19: vboxnet2: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
     link/ether 0a:00:27:00:00:02 brd ff:ff:ff:ff:ff:ff
+23: eth10: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 1000
+    link/ether b2:af:52:da:bb:3a brd ff:ff:ff:ff:ff:ff
```

27 pav. Virtualių tinklo įrenginių pokyčiai



28 pav. Tinklo audito įrašas

Galime pastebėti žiūrint į **28 pav.** kodėl apskritai yra formuojamas tinklo įvykių pranešimas kuomet yra ir taip sugeneruojamas audito įrašas. Tokius įrašus yra sunku skaityti ir analizuoti, jie pagrinde naudojami tik norint surasti konkrečias įvykio detales. Trumpai galima šį įrašą padetalizuoti:

– *type=PROCTITLE*: Įrašo tipas nurodo visą komandinę eilutę, kuri sukėlė audito įvykį. Tačiau ji koduojama šešiaženkliu kodu. Šiuo atveju tai yra:

```
2F6F70742F676F6F676C652F6368726F6D652F6368726F6D65202D2D747970653
D7574696C697479202D2D7574696C6974792D7375622D747970653D6E6574776F
726B2E6D6F6A6F6D2E4E6574776F726B53657276696365202D2D6C616E673D65
6E2D5553202D2D736572766963652D73616E64626F782D747970653D6E6F6E
```

Jis išsikoduoja į:

```
/opt/google/chrome/chrome --type=utility --utility-sub-
type=network.mojom.NetworkService --lang=en-US --service-sandbox-type=non
```

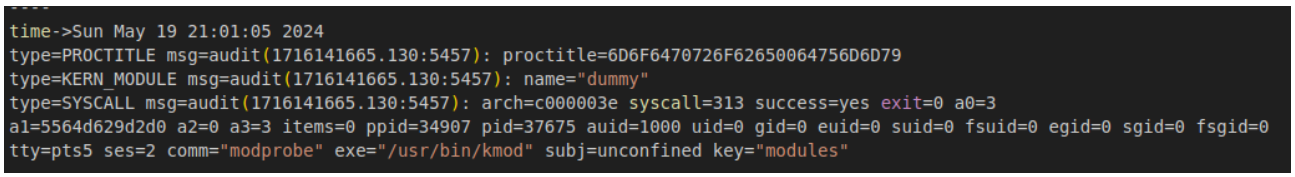
Tai reiškia, kad tai *chrome* procesas.

– *type=SOCKADDR*: Įrašo tipas nurodantis, paskirties adresą. Tai yra taip pat šešioliktainis įrašas tai atkodavimas jo identiškas ankščiau aprašytam.

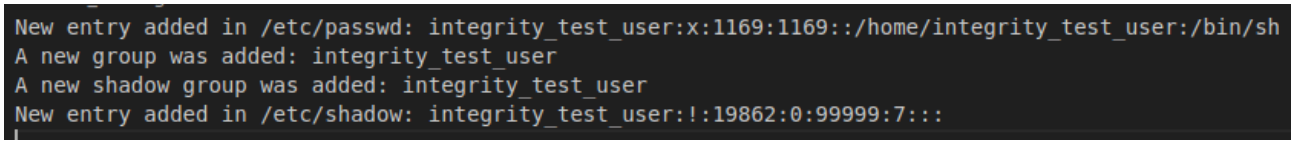
– *type=SYSCALL*: Įrašo tipas nurodantis, koks sisteminis iškvietimas buvo įvykdytas. Šiuo atveju jis – 42. Šis iškvietimas yra *connect,* o tai reiškia, kad buvo bandyta jungtis.

```
Module dummy was added.
Module algif_hash used_by count changed from 42 => 41
Module cmac was removed.
```

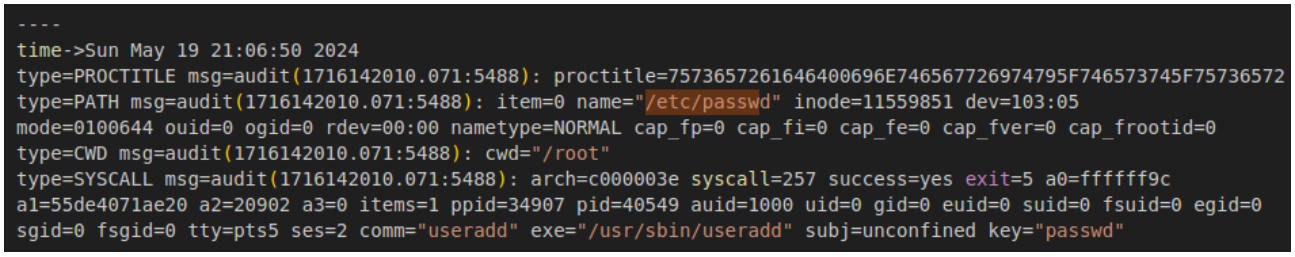
29 pav. Suformuoti modulių įvykiai iš audito failų



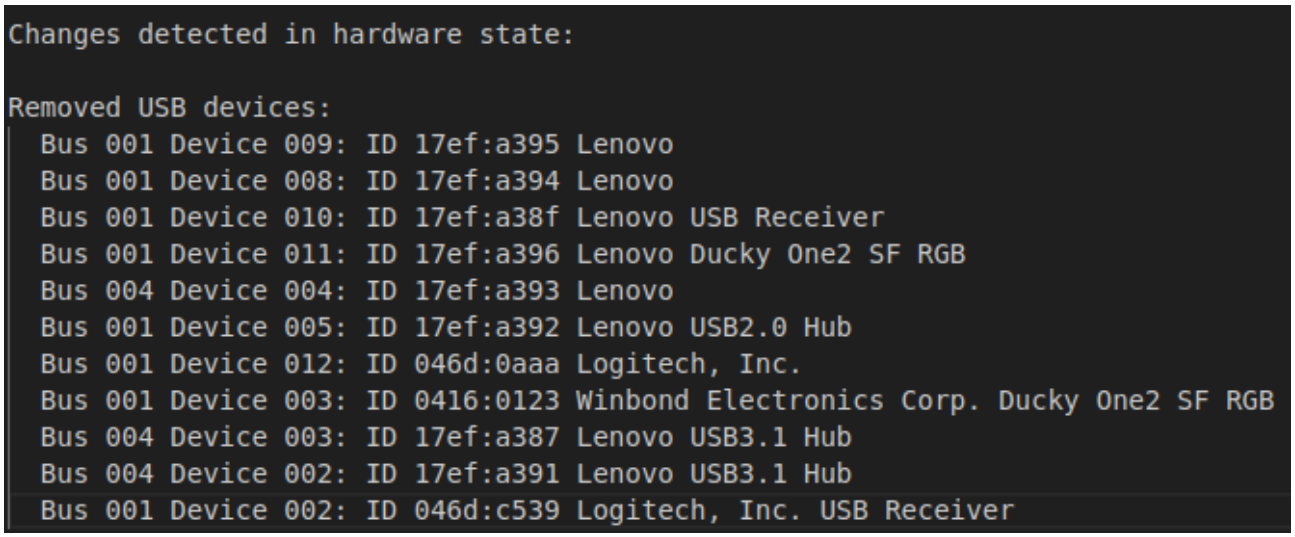
30 pav. Modulių audito įrašas



31 pav. Suformuoti naudotojų ir grupių įvykiai iš audito failų

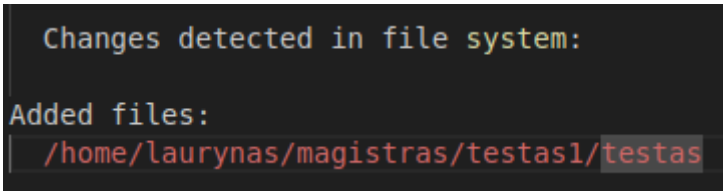


32 pav. Naudotojų ir grupių audito įrašas



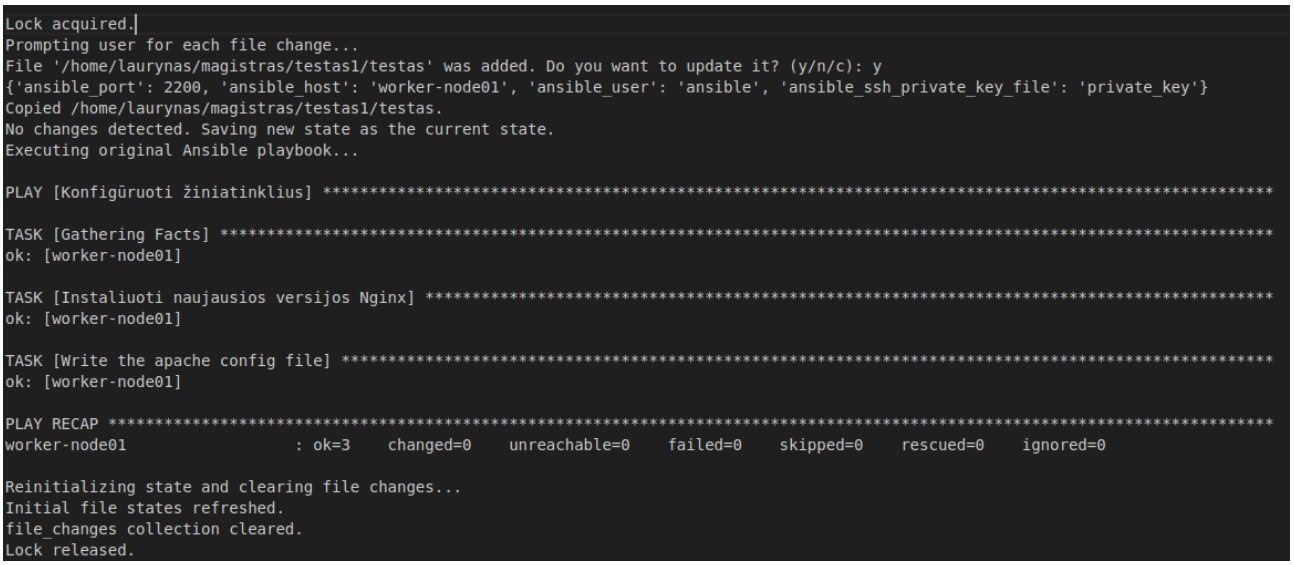
33 pav. Suformuotas aparatinės įrangos pranešimas

Aparatinės įrangos formavime yra pasitelkiamos tik komandinės eilutės programos, nes *auditd* neaptikdavo jokių audito pranešimų arba jie būdavo labai riboti.



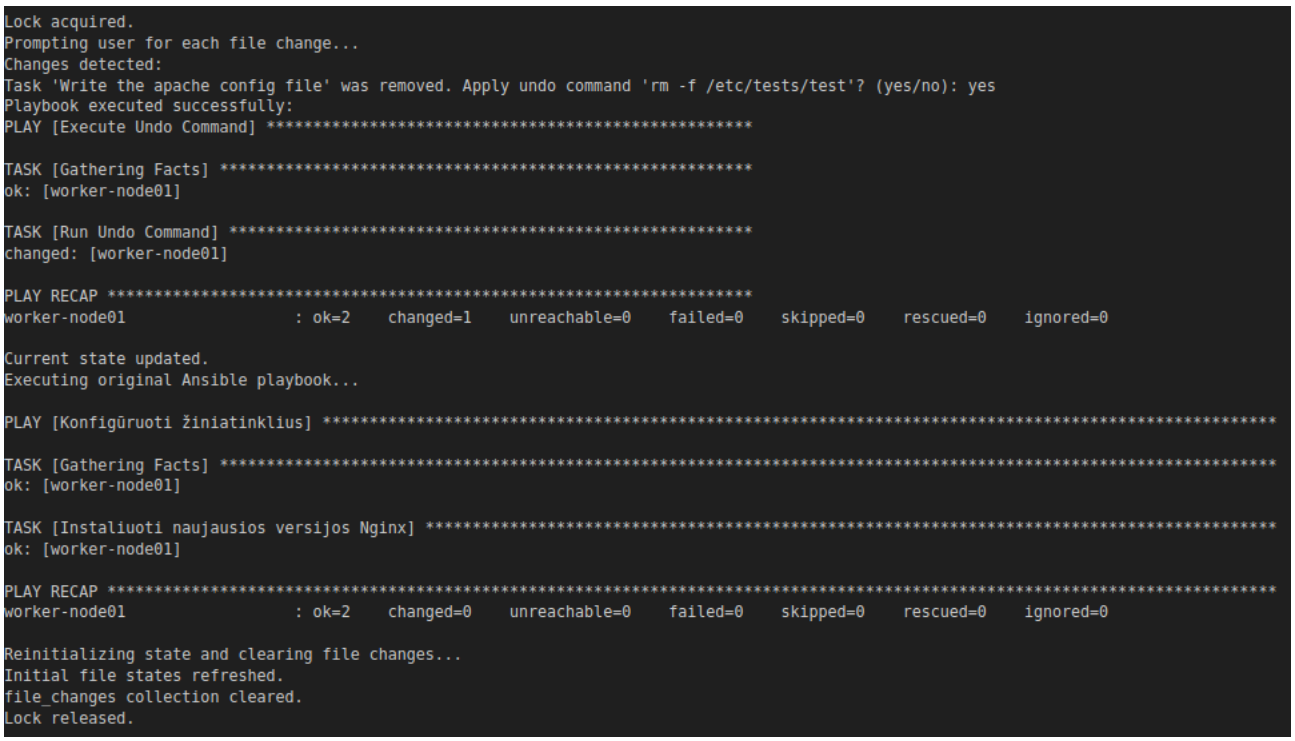
34 pav. Suformuotas failų pakitimo pranešimas

Konfigūravimo programa, jos iškvietimo metu aptinkami pakeitimai, pavyzdžiui, kai pridedamas naujas failas, administratorius yra informuojamas ir prašomas jo patvirtinti veiksmą. Nuotraukoje **35 pav**. parodytas pavyzdys, kaip programa nustato, kad failas buvo pridėtas. Administratorius turi pasirinkimą: patvirtinti veiksmą įvedant "*y*", atmesti įvedant "*n*" arba automatiškai patvirtinti visus būsimus pakeitimus įvedant "*c*". Šiuo atveju administratorius patvirtino veiksmą įvesdamas "*y*", todėl failas buvo nukopijuotas į būsenos serverį. Programa taip pat atnaujina sistemos būseną ir, jei reikia, vykdo tolesni konfigūravimą.



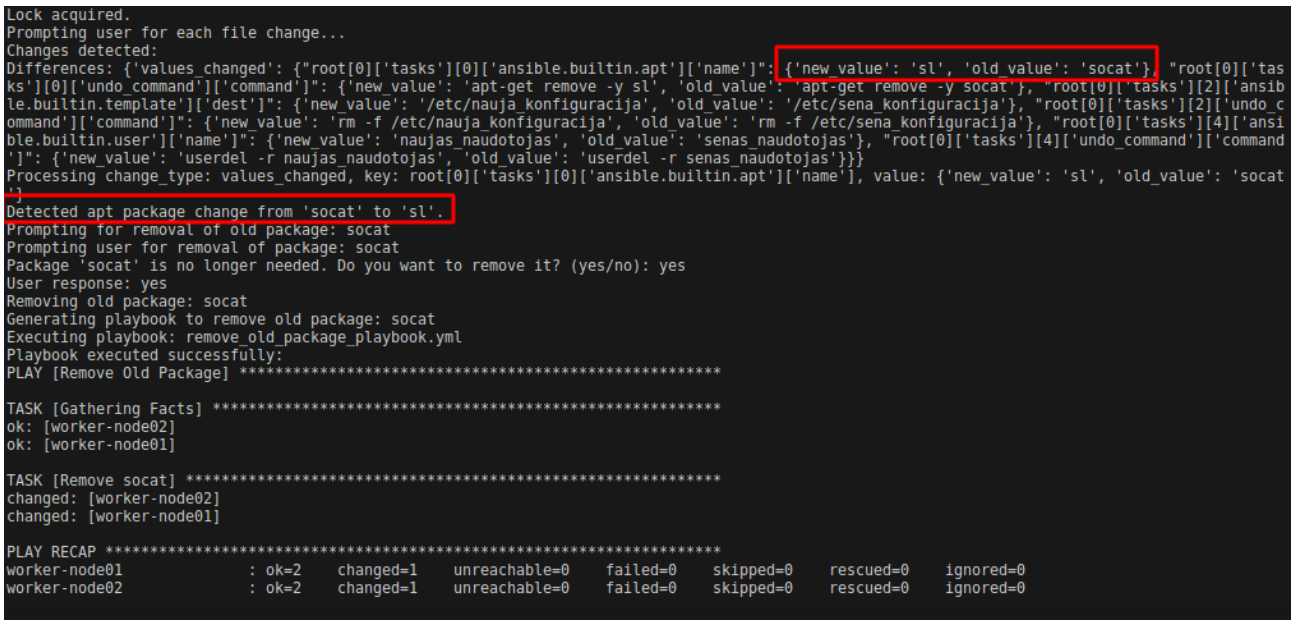
35 pav. Sistemos konfigūravimas ir vientisumo atnaujinimas

Atveju, kai programa aptinka galimus vientisumo konfigūracijoje pakitimus (žr. **36 pav.**), administratorius yra informuojamas ir prašomas patvirtinti ar atmesti veiksmus. Nuotraukoje matome, kad buvo aptiktas pakeitimas: failo "*/etc/tests/test*" pašalinimas. Vartotojui buvo pateiktas prašymas patvirtinti, ar reikia taikyti vientisumo atstatymo komandą "*rm -f /etc/tests/test*". Vartotojas įvedė "*yes*", todėl programa vykdė vientisumo užtikrinimo komandą iš sugeneruoto būsenos failo. Šis paprastas pavyzdys iliustruoja kodėl būsenos failo reikia. Tai gali būti failas, paketas, užsilikęs naudotojas ir daug kitų scenarijų. Šis procesas užtikrina, kad visi pakeitimai būtų valdomi ir jei reikia, atstatomi, taip išlaikant sistemos vientisumą.



36 pav. Sistemos konfigūravimas ir atstatymas pagal atstatymo failus

Ši programa taip pat atlieka ir egzistuojančių užduočių būsenos sekimą. Programa analizuoja du parametrus: senąją ir naująją būsenas. Kiekvienas aptiktas pakeitimas yra apdorojamas atskirai, priklausomai nuo pakeitimo tipo, programa patikrina specifinius modulius. Šis procesas yra vykdomas dvidešimčiai (žr. **38 pav.**) *Ansible* modulių tokiems kaip *file*, *template*, *user*, *cron*, *docker_container*, ir kt., užtikrinant, kad nebūtų palikti nereikalingi elementai sistemoje. Pavyzdys parodantis konfigūruojamos sistemos užsilikusį paketą ir dar keletą būsenos pakitimų matomas **37 pav**.



37 pav. Paketų būsenos pakitimas

38 pav. *Ansible* moduliai

Stebint sistemos vientisumą failų dalyje galime pasirinkti ją stebėti kokias tik norima būdais. Kaip pavyzdžiai gali:

- Konfigūraciniai failai: */etc*
- Programų katalogai */var/www*, */opt*
- Branduolio ir modulių konfigūracijos: */lib/modules*, */boot*
- Saugumo politikos ir *SELinux*: */etc/selinux*, */etc/audit*

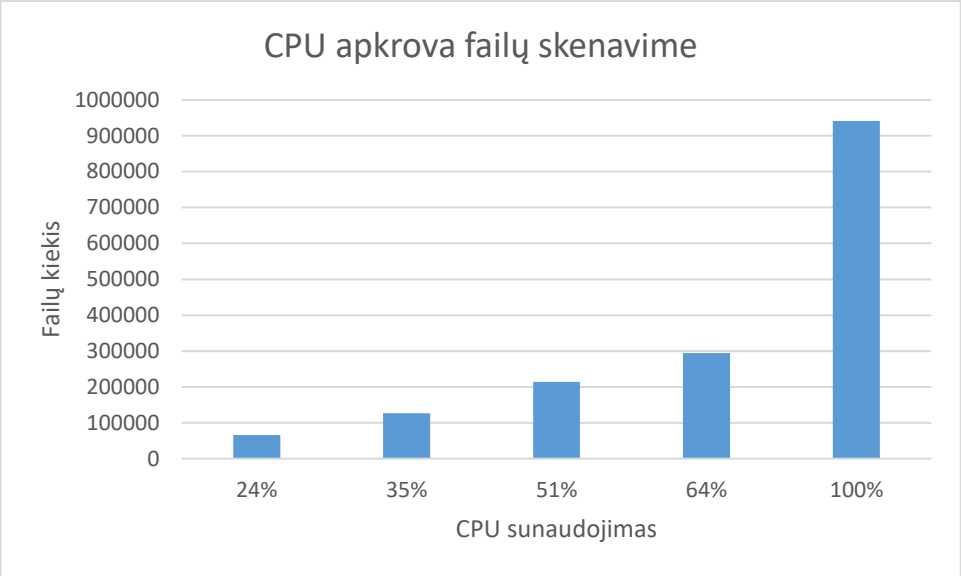
Tačiau tyrime išskiriami du būdai: programų konfigūraciją (žr. **10 lentelė**), visa sistema (žr. **11 lentelė).**

10 lentelė Programų konfigūracijos tikrinimo failai

Failas/Katalogas	**Aprašymas**
/etc	Svarbiausi sistemos konfigūracijos failai.
/usr/bin/	Standartinės vartotojo vykdomosios programos.
/usr/lib/	Bendros bibliotekos ir moduliai, naudojami /usr/bin/ ir kitų /usr/ katalogų programoms.
/usr/sbin/	Sistemos administravimo įrankiai, skirti superuser'io (root) naudojimui.
/usr/local/	Vietinės (ne sisteminės) programų įdiegimo katalogas, kurį paprastai naudoja vartotojas.
/lib/	Pagrindinės sistemos bibliotekos, reikalingos sistemos įkrovimui ir /bin/ bei /sbin/ katalogų vykdomųjų programų veikimui.
/sbin/	Svarbūs sistemos administravimo įrankiai, reikalingi sistemos įkrovimui ir gedimų šalinimui.

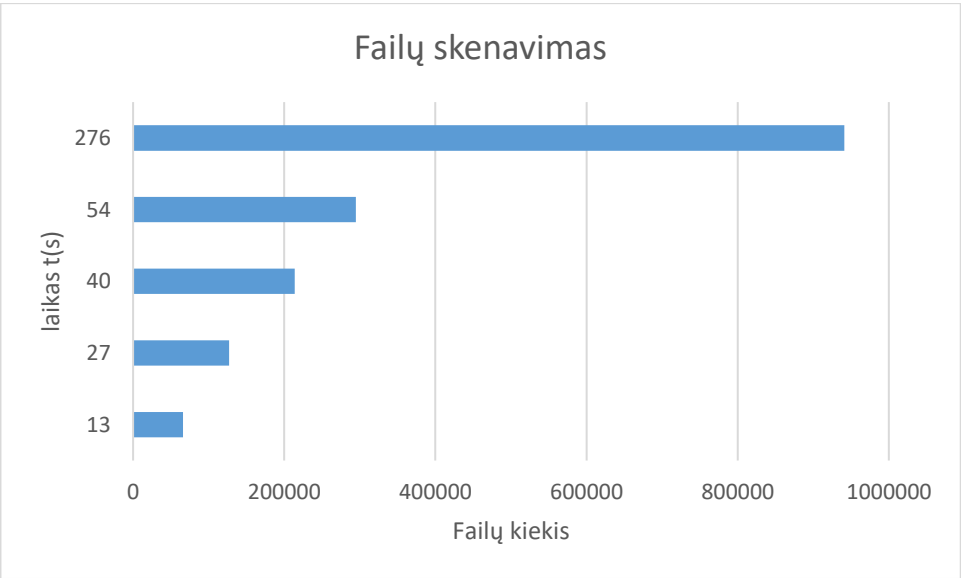
11 lentelė Visos sistemos tikrinimo failai

Failas/Katalogas	**Aprašymas**
/bin	Dvejetainės arba vykdomosios programos.
/etc	Sistemos konfigūracijos failai.
/usr/lib/	Bendros bibliotekos ir moduliai, naudojami /usr/bin/ ir kitų /usr/ katalogų programoms.
/usr/sbin/	Sistemos administravimo įrankiai, skirti superuser'io (root) naudojimui.
/usr/local/	Vietinės (ne sisteminės) programų įdiegimo katalogas, kurį paprastai naudoja vartotojas.
/lib/	Pagrindinės sistemos bibliotekos, reikalingos sistemos įkrovimui ir /bin/ bei /sbin/ katalogų vykdomųjų programų veikimui.
/sbin/	Svarbūs sistemos administravimo įrankiai, reikalingi sistemos įkrovimui ir gedimų šalinimui.
/proc	Virtualus failų sistema, teikianti informaciją apie aparatūrą ir veikiančius procesus.
/sys	Suteikia informaciją apie įrenginius, įrenginio tvarkykles ir kitas aparatūros savybes.
/dev	Specialūs failai, atstovaujantys aparatūros įrenginiams.
/boot/	Katalogas, kuriame saugomi sistemos įkroviklio (bootloader) ir branduolio (kernel) failai, reikalingi kompiuterio paleidimui.
/mnt/	Laikinas prijungimo taškas (mount point) skirtas prijungti laikmenas ar failų sistemas naudojant komandas.
/home	Namų katalogas. Tai numatytasis dabartinis katalogas.
/opt	Neprivaloma arba trečiųjų šalių programinė įranga



39 pav. Failų vientisumo skenavimo CPU apkrova

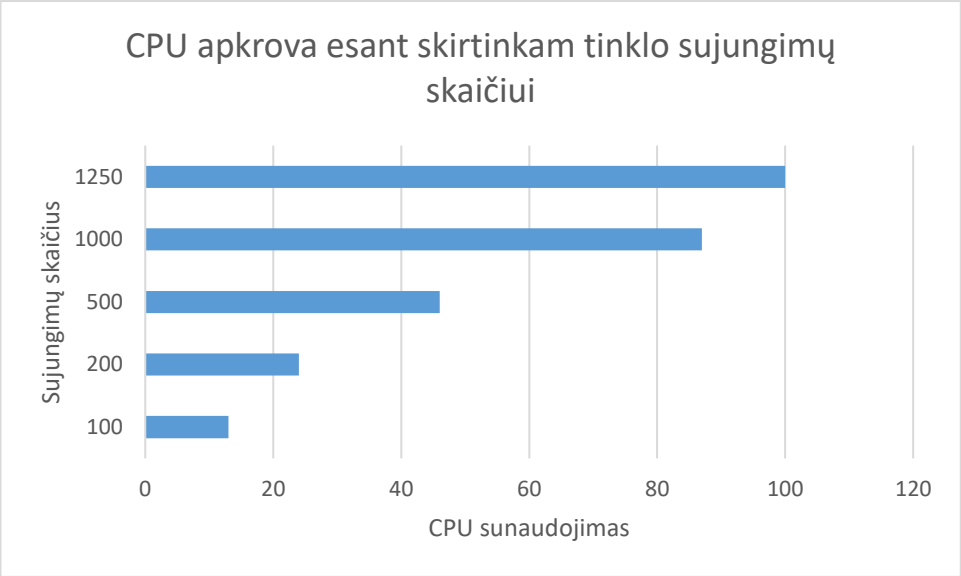
Matome, kad didėjant CPU sunaudojimui, nuskenuotų failų kiekis taip pat žymiai padidėja (žr. **39 pav.**). Prie 24% CPU sunaudojimo buvo nuskenuota apie 100 000 failų, o prie 100% CPU sunaudojimo - apie 1 000 000 failų. Tai rodo, kad šių virtualių aplinkų skenuojamų failų limitas ir iki 1 000 000 failų. Kadangi šis skaičius nėra toks didelis, galima daryti išvada, kad programa neefektyviai naudoja resursus.



40 pav. Failų vientisumo skenavimo laiko trukmė

Matome, kad nuskenuoto failų kiekio didėjimas tiesiogiai didina skenavimo laiką (žr. **40 pav.** Prie 67000~ failų skenavimas truko 13 sekundžių, prie 127000~ failų - 27 sekundes, o prie 1 000 000 failų - net 276 sekundes. Šie rezultatai rodo, kad nors failų kiekis didėja, laiko intervalai nėra visiškai tiesiniai, o tai gali būti dėl optimizacijų ar sistemos resursų efektyvumo didėjant apkrovai.

Taip pat buvo atliktas testas *auditd „*naujų tinklo ryšių sudarymai“ taisyklėms iš **1 lentelė**. Sugeneravus dirbtinę apkrova virtualiose aplinkose buvo matyti ženkli CPU sunaudojimo tendencija (žr. **41 pav.**) kylant sujungimo skaičiams.



41 pav. *Auditd* naujų tinklo ryšių sudarymo apkrova

3.4. Palyginimas su esamomis sistemomis

Šiame skyriuje yra palyginamas keletas egzistuojančių rinkoje integralumo sistemų su pasiūlyto modelio sistema (žr. **12 lentelė**).

12 lentelė Sistemų funkcionalumo palyginimas

	Kuriama sistema	OSSEC	Beats
Geba konfigūruoti sistemas išlaikant integralumą	TAIP	NE	NE
Geba išlaikyti failų integralumą	TAIP	TAIP	TAIP
Integralumo tikrinimo tipas	Pasyvus	Aktyvus	Aktyvus
Saugus ryšio kanalas	TAIP	TAIP	TAIP
Failų integralumo atstatymas	TAIP	NE	NE
Operacinės sistemos	Linux	Linux, Windows	Linux, Windows
Ekspertizių įrašų kopijų saugojimas	TAIP	TAIP	NE
Pranešimų būdai	El. paštas	El. paštas, Teams, SMS ir t.t	El. paštas, Teams, SMS ir t.t

Matome, kad kuriama sistema išsiskiria tuo, kad sugeba palaikyti failų integralumą konfigūracijos metu lyginant su kitomis sistemos. Taip pat sugebėjimas atstatyti pakitusius failus. Tuo tarpu *OSSEC* ir *Beats* pranašumai yra sugebėjimas aptikti integralumo pakitimus realiu laiku ir kur kas didesnis pritaikomumas skirtingose operacinėse sistemose. Visos sistemos užtikrina saugų ryšio

kanalą ir veikia tiek Linux, tiek Windows operacinėse sistemose. Pranešimų būdai kuriamoje sistemoje apima tik el. paštą, o kitos sistemos palaiko platų pranešimų būdų spektrą.

3.5. Tyrimo apibendrinimas

1. Visa sistemos tikrinimo trukmė yra 276 sekundės, tačiau programų konfigūracija užtrunka tik 27 sekundes. Tai rodo, kad nors visos sistemos tikrinimas yra lėtas, tam tikri specifiniai veiksmai gali būti atliekami pakankamai greitai.
2. Iš visų trejų tyrimo rezultatų matome linijinį apkrovos kylimą, kas padeda su apkrovos nuspėjamumu stebimose sistemose.
3. Atlikus *CPU* apkrovos eksperimentus galime daryti prielaidą, kad sistema yra tinkama mažo-vidutinio dydžio sistemoms, su didelėmis apkrovomis nėra optimaliai susidorojama.
4. Sugeneruoti audito duomenys ir tikslūs, tačiau iš pavyzdžių matomi, kad sunkiai skaitomi. Dėl to reikėtų kur kas pažangesnio pirminio duomenų apdorojimo.
5. Sukurta sistema turi pranašumą su lyginamomis sistemomis integralumo išlaikymo dalyje konfigūruojant ir atstatant failus.

Išvados

1. Analizės skyriuje buvo išnagrinėtos dažniausiai pasitaikantys įrangos konfigūracijos valdymo metodai. Išanalizavus metodus, buvo nuspręsta, kad kuriama sistema užtikrins sistemos integralumą ne tik ramybės būsenoje ar įprastoje eigoje, bet ir konfigūravimo metu.
2. Projekto kūrimo metu buvo pasiūlytas metodas saugoti konfigūruojamos sistemos būsenas ir turėti automatines atstatymo komandas. Tai leidžia palaikyti sistemą nuo pamirštų konfigūracijų. Taip pat naudoti mišrius sistemos integralumo tikrinimo būdus. Dalį stebėjimų atlikti pasyviai, o dalį aktyviai.
3. Kuriant būsenos prototipą, buvo pastebėta, kad didelė dalis konfigūruojamų veiksmų sistemose negali būti saugomi būsenos failuose, nes jei paprasčiausiai neturi atstatymo funkcijų.
4. Nors sudaryta failų skenavimo programa atlieka savo darbą, ji nepajėgia optimali naudoti resursus. Skenavimo laikas gali būti ilgas, ypač didelėse sistemose, kas gali apsunkinti realaus laiko vientisumo tikrinimą.
5. Atlikus metodo prototipo realizacija, buvo sėkmingai realizuotas kompiuterinės sistemos vientisumo užtikrinimo prototipas, kuris buvo sukurtas pagrinde naudojantis *Python* programavimo kalba.
6. Nenustatant vientisumo, sistema gali būti sukompromituota. Nustatytos anomalijos gali padėti greitai aptikti įsibrovimus ar kitus neleistinus veiksmus, kurie gali kelti pavojų sistemos saugumui ir stabilumui.
7. Pastebėta, kas užtikrinti visos sistemos vientisumą yra sudėtinga, ypač didelėse ir sudėtingose sistemose, kuriose yra daug failų ir katalogų. Tokios sistemos gali reikalauti daug laiko ir resursų skenavimui ir tikrinimui.

Literatūros sąrašas

[1] ASQ. *WHAT IS ISO 9001:2015 – QUALITY MANAGEMENT SYSTEMS?* [Tinkle] [Cituota: 2023 m. 1 13 d.] https://asq.org/quality-resources/iso-9001.

[2] Bigelow, Stephen J. techtarget. *What is configuration management? A comprehensive guide.* [Tinkle] 2020 m. 11. [Cituota: 2023 m. 1 13 d.] https://www.techtarget.com/searchitoperations/definition/configuration-management-CM.

[3] Hanna, Katie Terrell. techtarget. *COBIT.* [Tinkle] 2021 m. 9. [Cituota: 2023 m. 1 13 d.] https://www.techtarget.com/searchsecurity/definition/COBIT.

[4] Shea, Garrett. nasa. *Configuration Management.* [Tinkle] 2019 m. 12 12 d. [Cituota: 2023 m. 1 13 d.] https://www.nasa.gov/seh/6-5-configuration-management.

[5] Dart, Susan. *Concepts in Configuration Management Systems.* Pittsburgh : Software Engineering Institute Carnegie-Mellon University, 1991.

[6] Oliver Hanappi, Waldemar Hummer,Schahram Dustdar. *Asserting Reliable Convergence for Configuration Management Scripts.* Vienna : Vienna University of Technology, 2016.

[7] Suresh Chandra Satapathy, Jyotsna Kumar Mandal,Siba K. Udgata,Vikrant Bhateja. *Information Systems Design and Intelligent Applications.* Warsaw : Polish Academy of Sciences, 2016.

[8] Vijaya Krishna, Gspann. *Comparing Configuration Management Tools: Chef vs. Puppet vs. Ansible.* [Tinkle] 2018 m. 11 5 d. [Cituota: 2023 m. 1 14 d.] https://www.gspann.com/resources/blogs/puppet-vs-chef-vs-ansible/.

[9] Jagarapu, AtchutaRao. ais. *Puppet Use Cases: How to Troubleshoot.* [Tinkle] 2021 m. 10 8 d. [Cituota: 2023 m. 1 14 d.] https://www.ais.com/puppet-use-cases-how-to-troubleshoot/.

[10] IRS, *Configuration and Change Management.* [Tinkle] 2022 m. 11 14 d. [Cituota: 2023 m. 1 14 d.] https://www.irs.gov/irm/part2/irm_02-150-002.

[11] Yasar, Kinza. techtarget. *checksum.* [Tinkle] 2022 m. 12. [Cituota: 2023 m. 1 14 d.] https://www.techtarget.com/searchsecurity/definition/checksum.

[12] Nabil Moukafih, Soukaina Sabir,Abdelmajid Lakbabi,Ghizlane Orhanou. researchgate. *SIEM selection criteria for an efficient contextual security.* [Tinkle] 2017 m. 5. [Cituota: 2023 m. 1 15 d.] https://www.researchgate.net/publication/320649753_SIEM_selection_criteria_for_an_efficient_contextual_security.

[13] Diogo Teixeira, Leonardo Assunção,Teresa Pereira,Silvestre Malta,Pedro Pinto. researchgate. *OSSEC IDS Extension to Improve Log Analysis and Override False Positive or Negative Detections.* [Tinkle] 2019 m. 9. [Cituota: 2023 m. 1 15 d.] OSSEC IDS Extension to Improve Log Analysis and Override False Positive or Negative Detections.

[14] Elastic. *Auditbeat overview*. [Tinkle] [Cituota: 2023 m. 1 15 d.] https://www.elastic.co/guide/en/beats/auditbeat/current/auditbeat-overview.html.

[15] Elastic *Heartbeat overview*. [Tinkle] [Cituota: 2023 m. 1 15 d.] https://www.elastic.co/guide/en/beats/heartbeat/current/heartbeat-overview.html.

[16] Elastic *Functionbeat overview*. [Tinkle] [Cituota: 2023 m. 1 15 d.] https://www.elastic.co/guide/en/beats/functionbeat/master/functionbeat-overview.html.

[17] Elastic *How metricbeat works*. [Tinkle] [Cituota: 2023 m. 1 15 d.] https://www.elastic.co/guide/en/beats/metricbeat/current/how-metricbeat-works.html.

[18] Elastic. *Packetbeat overview*. [Tinkle] [Cituota: 2023 m. 1 15 d.] https://www.elastic.co/guide/en/beats/packetbeat/current/packetbeat-overview.html.

[19] Elastic. *File integrity Module*. [Tinkle] [Cituota: 2023 m. 1 15 d.] https://www.elastic.co/guide/en/beats/auditbeat/current/auditbeat-module-file_integrity.html.

[20] Elastic. *How filebeat works.* [Tinkle] [Cituota: 2023 m. 1 15 d.] https://www.elastic.co/guide/en/beats/filebeat/current/how-filebeat-works.htm.

[21] Elastic. *Configure Winlogbeat.* [Tinkle] [Cituota: 2023 m. 1 15 d.] https://www.elastic.co/guide/en/beats/winlogbeat/current/configuration-winlogbeat-options.html .