Kauno technologijos universitetas

Informatikos fakultetas

Prieigos valdymo metodas mikropaslaugų architektūroje

Baigiamasis magistro projektas

Ernestas Serkovas

Projekto autorius

prof. Algimantas Venčkauskas

Vadovas

Kaunas, 2024

Kauno technologijos universitetas

Informatikos fakultetas

Prieigos valdymo metodas mikropaslaugų architektūroje

Baigiamasis magistro projektas

Informacijos ir informacinių technologijų sauga (6211BX008)

Ernestas Serkovas

Projekto autorius

prof. Algimantas Venčkauskas

Vadovas

doc. Šarūnas Grigaliūnas

Recenzentas

Kaunas, 2024

Kauno technologijos universitetas

Informatikos fakultetas

Ernestas Serkovas

Prieigos valdymo metodas mikropaslaugų architektūroje

Akademinio sąžiningumo deklaracija

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.

Ernestas Serkovas

Patvirtinta elektroniniu būdu

Serkovas, Ernestas. Prieigos valdymo metodas mikropaslaugų architektūroje. Magistro baigiamasis projektas / vadovas prof. Algimantas Venčkauskas; Kauno technologijos universitetas, informatikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): informatikos inžinerija.

Reikšminiai žodžiai: prieigos valdymas, mikropaslaugos, mikropaslaugų architektūra, daiktų internetas.

Kaunas, 2024. 56 p.

Santrauka

Modernios ir šiuolaikiškos sistemos, daiktų internetas dažnu atveju yra didelės apimties ir pakankamai kompleksiškos struktūros. Siekiant padidinti sistemų patikimumą ir pasiekiamumą, kartu su konteinerizavimo technologijų plėtra pradėta naudoti mikropaslaugų architektūra. Šios architektūros servisų autonomiškumas ir izoliacija leido optimizuoti išskaidytos sistemos valdymą bei plėtrą. Augant mikropaslaugų architektūra paremtų sistemų skaičiui, kartu atsiranda ir iššūkiai, susiję su servisų ir įrenginių sauga. Prieigos valdymas – viena iš pagrindinių saugos problemų, su kuriomis tenka susidurti projektuojant ir vystant mikropaslaugų architektūra paremtą sistemą. Servisai projektuojami taip, kad pasitikėtų užklausomis, kurios atkeliauja iš tarpusavyje komunikuojančių servisų. Šis pasitikėjimas gali būti išnaudojamas kitų servisų ar įrenginių sutrikdymui, jeigu viename iš mikropaslaugų architektūros komponentų bus pažeistas prieigos valdymas. Tobulėjanti daiktų interneto technologija vis dažniau naudoja mažus, nedaug resursų turinčius įrenginius, kuriems taip pat turi būti užtikrinama tinkama sauga ir prieigos valdymas. Dėl šios priežasties išsikeltas darbo tikslas – pasiūlyti metodą, sprendžiantį aktualias prieigos valdymo problemas mikropaslaugų architektūroje. Atlikus mikropaslaugų architektūros ir su ja susijusių prieigos valdymo problemų analizę pasiūlytas ir suprojektuotas prieigos valdymo metodas, sprendžiantis prieigos valdymo problemą ribotų resursų aplinkoje. Pagal darbe numatytus reikalavimus buvo sukurtas ir ištestuotas prototipas bei atliktas siūlomo prieigos valdymo metodo resursų naudojimo ir greitaveikos tyrimas infrastruktūroje.

Serkovas, Ernestas. Access Control Approach in Microservices Architecture. Master's Final Degree Project / supervisor prof. Algimantas Venčkauskas; Faculty of Informatics, Kaunas University of Technology.

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

Keywords: access control, microservices, microservices architecture, internet of things.

Kaunas, 2024. 56 p.

Summary

Modern and contemporary systems, the Internet of Things, are often large-scale and have a complex structure. To increase the reliability and availability of systems, together with the development of containerization technologies, the use of microservices architecture began. The autonomy and isolation of the services of this architecture made it possible to optimize the management and development of the fragmented system. As the number of systems based on microservices architecture grows, so do challenges related to the security of services and devices. Access control is one of the main security issues faced when designing and developing a system based on microservices architecture. Services are designed to trust requests that come from communicating services. This trust can be exploited to disrupt other services or devices if access control in one of the components of the microservices architecture is compromised. As the Internet of Things technology evolves, it increasingly uses small, constrained devices that also require adequate security and access control. For this reason, the goal of the paper is to propose a method that solves the current problems of access management in microservices architecture. After analyzing the microservices architecture and access management problems related to it, an access management method was proposed and designed, which solves the access management problem in an environment of limited resources. According to the requirements set out in the paper, a prototype was created and tested, and a study of the resource use and speed of the proposed access control method in the infrastructure was carried out.

Turinys

Lentelių sąrašas

Paveikslų sąrašas

Santrumpų ir terminų sąrašas

Santrumpos:

CoAP (angl. Constrained Application Protocol) – specializuotas, UDP paremtas, protokolas, skirtas apribotiems įrenginiams, aprašytas RFC 7252 dokumente;

DTLS (angl. Datagram Transport Layer Security) – protokolas, apsaugantis duomenų paketais pagrįstas programas;

TLS (angl. Transport Layer Security) – kriptografinis protokolas, numatantis apsaugotą duomenų perdavimą tarp mazgų pasauliniame kompiuterių tinkle;

UDP (angl. User Datagram Protocol) – transporto protokolas, skirtas laikui jautrioms programoms;

REST (angl. Representational State Transfer) – architektūrinis stilius, skirtas kurti saityno paslaugas ir sistemas, kurios gali lengvai bendrauti tarpusavyje;

API (angl. Application Programming Interface) – sąsaja, kurią suteikia kompiuterinė sistema, biblioteka ar programa tam, kad programuotojas per kitą programą galėtų pasiekti jos funkcionalumą ar apsikeistų su ja duomenimis;

HTTP (angl. Hypertext Transfer Protocol) – pagrindinis metodas informacijai pasauliniame tinkle pasiekti;

mTLS (angl. Mutual Transport Layer Security) – autentifikavimo metodas, kuriame abi komunikuojančios pusės autentifikuojasi tarpusavyje naudojant TLS protokolą;

JSON – atviro standarto formatas, perduodantis duomenų objektus, sudarytus iš atributo ir reikšmės porų, lengvai skaitomame tekste;

JSON Web Token (JWT) – JSON saityno prieigos raktas, skirtas kurti duomenis su pasirenkamu parašu ir šifravimu;

ECDSA (angl. Elliptic Curve Digital Signature Algorithm) – skaitmeninis parašas, kuriame naudojama elipsinės kreivės kriptografija;

HMAC (angl. Hash-based Message Authentication Code) – kriptografinis autentifikacijos metodas, kuris naudoja maišos funkciją ir slaptą žymą;

RSA (Rivest–Shamir–Adleman) – viešojo rakto kriptosistema, kurios algoritmą sukūrė Ronald Rivest, Adi Shamir ir Leonard Adleman;

SSL (angl. Secure Sockets Layer) – saugos protokolas, numatantis konfidencialumą, autentifikaciją ir duomenų integralumą komunikuojant internete.

Terminai:

Daiktų internetas – įrenginiai su jutikliais, duomenų apdorojimo galimybėmis, programine įranga ir kitomis technologijomis, kurie internetu ar kitais ryšių tinklais jungiasi ir keičiasi duomenimis su kitais įrenginiais ir sistemomis;

Konteinerizavimo technologija – operacinės sistemos lygio virtualizavimas arba programos lygio virtualizavimas per kelis tinklo išteklius, kad programinė įranga galėtų veikti izoliuotose vartotojų erdvėse, vadinamose konteineriais, bet kurioje debesies ar ne debesies aplinkoje, neatsižvelgiant į tipą ar tiekėją;

Mikropaslaugų architektūra – programinės įrangos architektūra, kurioje programa kuriama kaip laisvai susietų, smulkių paslaugų rinkinys, bendraujantis naudojant lengvasvorius protokolus;

Ūko kompiuterija – decentralizuota infrastruktūra, kurioje duomenų apdorojimas ir skaičiavimai vykdomi tarp duomenų šaltinio ir debesies;

Vieno prisijungimo sistema (SSO) – autentifikavimo schema, leidžianti vartotojui prisijungti naudojant vieną identifikatorių prie bet kurios iš kelių susijusių, tačiau nepriklausomų sistemų;

API šliuzas – programinės įrangos sluoksnis, kuris veikia kaip vienas galinis taškas įvairioms funkcijoms, pavyzdžiui, užklausų sudarymui, maršruto parinkimui ir protokolų keitimui;

Vaidmenimis pagrįstas prieigos valdymas (RBAC) – prieigos valdymo metodas, leidžiantis vartotojui priskirti prieigos valdymo taisykles pagal jo vaidmenį sistemoje;

Maišos funkcija – matematinė funkcija, sukurianti fiksuoto ilgio simbolių eilutę iš teksto arba skaičių sąrašo;

„Docker“ – produktų rinkinys, kuris naudoja operacinės sistemos lygio virtualizaciją, kad programinė įranga galėtų veikti izoliuotose vartotojų erdvėse, vadinamose konteineriais;

„Kubernetes“ – atvirojo kodo konteinerių orkestravimo sistema, skirta automatizuoti programinės įrangos diegimą, mastelį ir valdymą;

OAuth 2.0 – atviras prieigos delegavimo standartas;

„Linux“ – atviro kodo operacinė sistema;

„Ubuntu“ – „Linux“ šeimos operacinė sistema;

„Docker Engine“ – atviro kodo programinė įranga, skirta talpinti konteineriams;

„Nginx“ – saityno serveris, kuris taip pat gali būti naudojamas kaip atvirkštinis tarpinis serveris, apkrovos balansavimo priemonė, pašto tarpinis serveris ir HTTP podėlis;

„PHP“ – dinaminė interpretuojama programavimo kalba, pritaikyta interneto svetainių kūrimui;

„MariaDB“ – reliacinių duomenų bazių valdymo sistema;

„Guzzle“ – biblioteka, skirta siųsti HTTP užklausas;

„PHP-JWT“ – biblioteka JWT žetonų užkodavimui ir iškodavimui;

„Laravel“ – atviro kodo „PHP“ saityno programos karkasas;

„Postman“ – programinė įranga, skirta testuoti API;

„cURL“ – atviro kodo programinė įranga duomenų perdavimui įvairiais transporto protokolais.

Įvadas

Modernios ir šiuolaikiškos sistemos, daiktų internetas dažnu atveju yra didelės apimties ir pakankamai kompleksiškos struktūros. Siekiant padidinti sistemų patikimumą ir pasiekiamumą, kartu su konteinerizavimo technologijų plėtra pradėta naudoti mikropaslaugų architektūra. Šios architektūros servisų autonomiškumas ir izoliacija leido optimizuoti išskaidytos sistemos valdymą bei plėtrą. Augant mikropaslaugų architektūra paremtų sistemų skaičiui, kartu atsiranda ir iššūkiai, susiję su servisų ir įrenginių sauga. Prieigos valdymas – viena iš pagrindinių saugos problemų, su kuriomis tenka susidurti projektuojant ir vystant mikropaslaugų architektūra paremtą sistemą. Servisai projektuojami taip, kad pasitikėtų užklausomis, kurios atkeliauja iš tarpusavyje komunikuojančių servisų. Šis pasitikėjimas gali būti išnaudojamas kitų servisų ar įrenginių sutrikdymui, jeigu viename iš mikropaslaugų architektūros komponentų bus pažeistas prieigos valdymas. Tobulėjanti daiktų interneto technologija vis dažniau naudoja mažus, nedaug resursų turinčius įrenginius, kuriems taip pat turi būti užtikrinama tinkama sauga ir prieigos valdymas. Dėl šios priežasties išsikeltas darbo **tikslas** – pasiūlyti metodą, sprendžiantį aktualias prieigos valdymo problemas mikropaslaugų architektūroje. Numatytam tikslui pasiekti išsikelti **uždaviniai**:

1. Susipažinti su aktualiomis prieigos valdymo problemomis mikropaslaugų architektūroje;
2. Atlikti esamų prieigos valdymo metodų analizę;
3. Palyginti esamus prieigos valdymo metodus;
4. Pasiūlyti prieigos valdymo metodą mikropaslaugų architektūrai;
5. Realizuoti pasiūlytą prieigos valdymo metodą ir atlikti metodo analizę.

Pirmajame darbo skyriuje analizuojama mikropaslaugų architektūra, jos principai ir pritaikymas daiktų internete bei aktualios prieigos valdymo problemos. Taip pat apžvelgiami esami prieigos valdymo metodai mikropaslaugų architektūroje ir atliekamas šių metodų palyginimas pagal pasirinktus kriterijus. Antrame skyriuje aprašomas siūlomas prieigos valdymo metodas, pristatomas jo projektas. Trečiasis skyrius apima prieigos valdymo metodo mikropaslaugų architektūroje prototipo aprašymą, infrastruktūros ir esminių komponentų detalizavimą. Ketvirtajame skyriuje aprašomas siūlomo prieigos valdymo metodo prototipo komponentų testavimas ir resursų naudojimo bei greitaveikos tyrimas infrastruktūroje, detaliai pristatomi rezultatai.

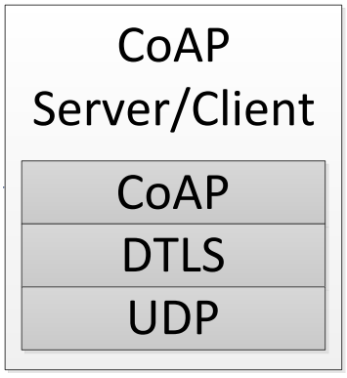
1. Mikropaslaugų architektūros analizė

1.1. Mikropaslaugų architektūra daiktų internete

Daiktų internetas – vis dažniau sutinkama technologija, kuri sudaryta iš įrenginių, sujungtų į tinklus. Tai gali būti įvairūs buities prietaisai, mechanizmai, jutikliai. Šie prietaisai renka ir perduoda duomenis į ūko sluoksnį, kuriame esantys serveriai atlieka sprendimų priėmimą ir gautų duomenų apdorojimą. Ūko kompiuterija tinka tokioms sistemoms, kuriose svarbu greitas atsakymas į užklausas, maža delsa ir realaus laiko duomenų apdorojimas. Tobulėjant daiktų interneto technologijai, galiniai įrenginiai smulkėja, o kartu mažinami ir juose esantys resursai, tokie kaip procesorius, operatyvioji atmintis. Komunikacijai tarp galinių įrenginių ir ūko sluoksnio servisų naudojami lengvasvoriai, mažiau sudėtingi komunikacijos protokolai, kurie leidžia sumažinti užklausų vykdymo laiką. Nors modernūs galiniai įrenginiai gali palaikyti tam tikras kriptografines operacijas, tačiau jos užtrunka per ilgai. Duomenų surinkimui ir apdorojimui ūko sluoksnyje naudojami daugiau resursų turintys įrenginiai, kurie apdorotus duomenis perduoda tolimesniems, išoriniams sluoksniams. [1]

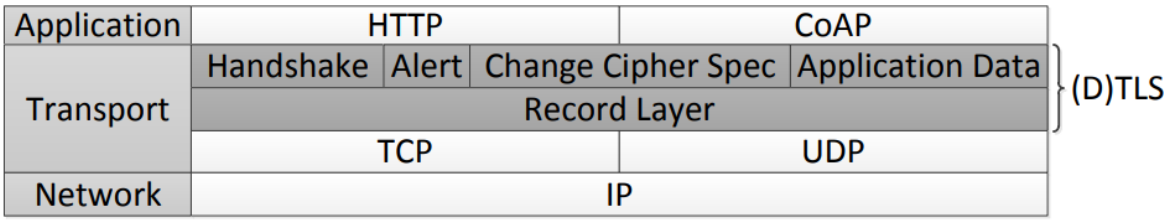
Mažėjantys įrenginių resursai kelia papildomų iššūkių, siekiant juos apsaugoti nuo įvairių grėsmių. Nors standartiniai, plačiai paplitę saugos metodai gali būti taikomi daiktų interneto įrenginiams su didesniu kiekiu resursų, tačiau galiniams įrenginiams reikalingas sprendimas, kuris leistų pasiekti patenkinamą saugumo lygį su mažiau naudojamais resursais. Dėl apribotų resursų aplinkos, konfidencialumo, integralumo ir įrenginių prieigos valdymo užtikrinimas tampa aktualia problema. Specialūs, lengvasvoriai komunikacijos protokolai, tokie kaip CoAP, kartu su kitais efektyviais, resursus taupančiais saugos algoritmais gali padėti spręsti šias problemas [1].

Komunikacijai tarp daiktų interneto įrenginių naudojamas CoAP protokolas [2]. Jis taip pat gali būti naudojamas ir komunikacijai tarp mikropaslaugų servisų. Kaip pateikiama 1 paveiksle, CoAP veikia UDP transporto protokolu todėl jam būtina užtikrinti saugą.



1 pav. CoAP serverio ir kliento naudojami protokolai [2]

CoAP saugai naudojamas DTLS protokolas, kuris paremtas TLS protokolu su tam tikrais pakeitimais dėl transporto lygmenyje naudojamo UDP protokolo [1]. Tačiau, kaip pavaizduota 2 paveiksle esančioje lentelėje, kartu su juo vyksta daug papildomų veiksmų, tokių kaip: pasisveikinimas (angl. handshake), sertifikatų duomenų apsikeitimas, pranešimai. Šie veiksmai reikalauja papildomų resursų iš įrenginio, kurių resursai, dažniausiai, yra riboti [2].

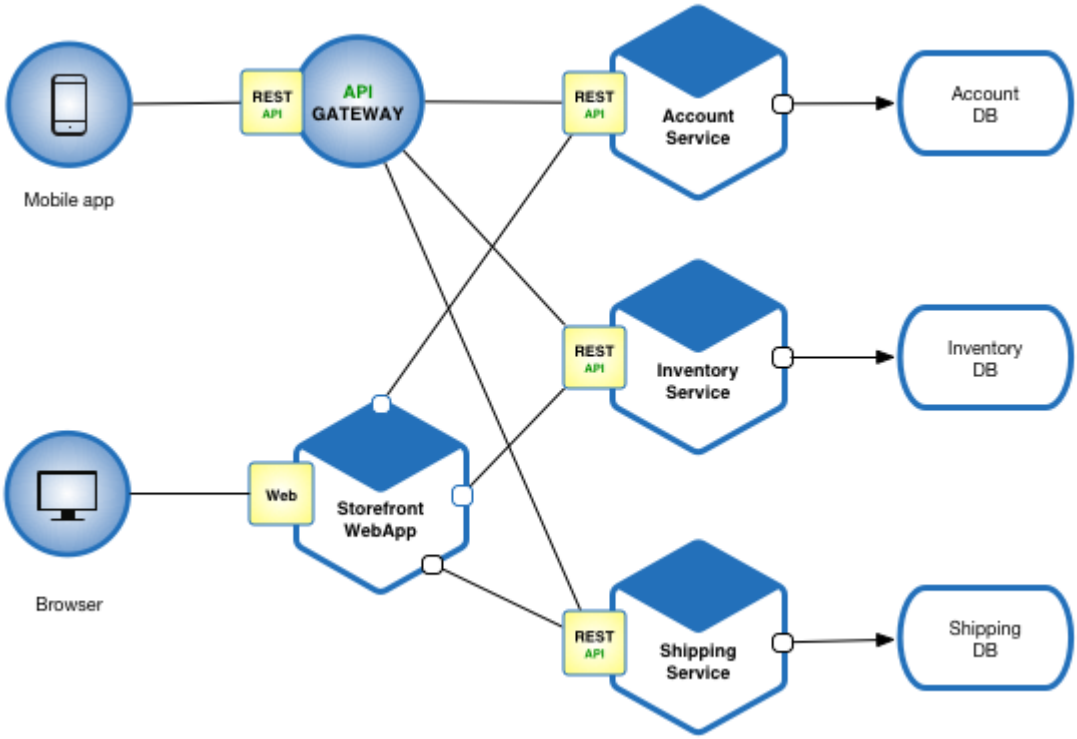


2 pav. DTLS protokolo struktūra [2]

Dėl papildomų procesų DTLS protokole ir ribotų įrenginių resursų reikalingas paprastesnis, mažiau resursų iš galinių įrenginių reikalaujantis prieigos valdymo metodas mikropaslaugų architektūrai, kuris galėtų būti alternatyva CoAP protokolui.

1.2. Mikropaslaugų architektūra ir prieigos valdymo problemos

Mikropaslaugų architektūra yra paremta funkcijų paskirstymu į individualius, mažos apimties servisus, kurie veikia savo atskiruose procesuose ir tarpusavyje komunikuoja lengvasvoriais protokolais [3]. Mikropaslaugų servisai yra autonominiai ir izoliuojami vieni nuo kitų panaudojant konteinerizavimo technologijas, tokias kaip „Docker“ [4]. 3 paveiksle pateikiama mikropaslaugų architektūra paremtos programos schema, kurioje pavaizduoti trys servisai, aptarnaujantys elektroninę parduotuvę. Kiekvienas servisas atlieka tam tikrą funkciją, pavyzdžiui, inventoriaus administravimą. Komunikacijai tarp servisų naudojama REST API, kurios dėka servisai gali keistis duomenimis nepriklausomai nuo to, kokia technologija jie paremti. Taip pat šis komunikacijos būdas leidžia programą naudoti skirtingiems įrenginiams.



3 pav. Mikropaslaugų architektūra paremta programa [5]

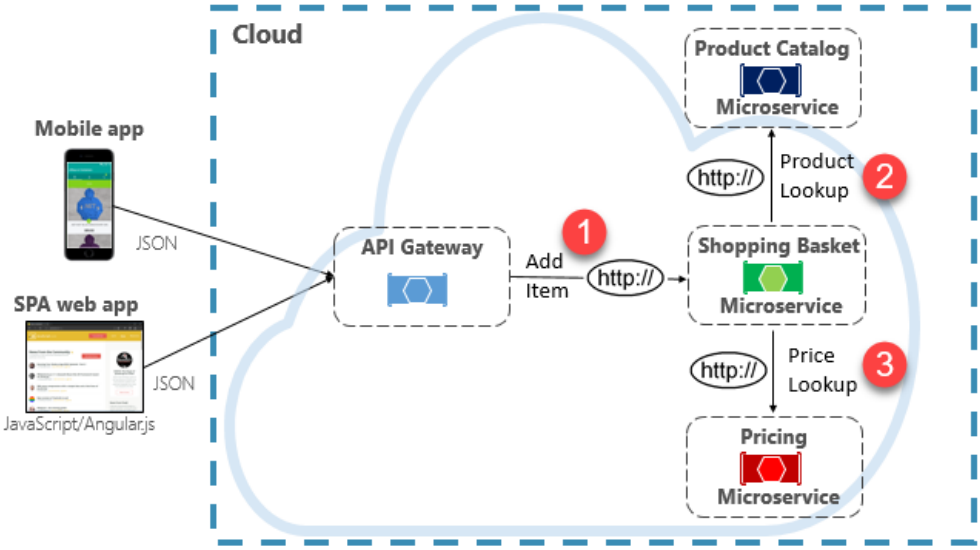
Naudojant mikropaslaugų architektūrą, didelės apimties, kompleksiškos programos išskaidomos į mažesnes dalis, kurių dėka galima pasiekti didesnį pasiekiamumą, patikimumą. Taip pat optimizuoti jų valdymą ir apimties plėtrą [3]. Be šių privalumų mikropaslaugų architektūros servisai gali būti nepriklausomai valdomi, paleidžiami ir plečiami. Taip pat dėl servisų izoliavimo vienas nuo kito, kiekvienas iš jų gali naudoti skirtingas technologijas, kurias galima pritaikyti specifiškai pagal serviso atliekamą funkciją. Mikropaslaugų architektūros servisų izoliacija prisideda ir prie programos patikimumo, kadangi įvykus klaidai servise, problema paveiks tik šio serviso funkcijas, o visos kitos programos dalys toliau sėkmingai vykdys savo procesus [6].

Mikropaslaugų architektūra išpopuliarėjo kartu su konteinerizavimo technologijų, tokių kaip „Kubernetes“ ir „Docker“, plėtra [4]. Tarp 2020 metų gruodžio ir 2021 metų sausio tyrimų bendrovė „Statista“ atliko tyrimą, kurio metu aiškinosi, kokia dalis įmonių naudoja mikropaslaugų architektūrą. Apklausus 950 įvairių dydžių įmonių visame pasaulyje paaiškėjo, kad savo organizacijoje mikropaslaugų architektūrą naudoja 75 procentai įmonių su 1000-2999 darbuotojų, 84 procentai su 3000-4999 darbuotojų ir 85 procentai su 5000 ir daugiau darbuotojų [7]. Tai rodo, kad mikropaslaugų architektūra yra pakankamai populiarus pasirinkimas projektuojant sistemas ir programas.

Nors mikropaslaugų architektūra suteikia privalumų, tačiau taip pat sukelia iššūkių, susijusių su servisų apsauga. Pagrindinės problemos kyla dėl to, kad saugumo klausimai turi būti sprendžiami dėl kiekvieno serviso individualiai [3]. Vienas iš pagrindinių mikropaslaugų architektūros saugumo klausimų – prieigos valdymas. Servisai tarpusavyje dalinasi duomenimis todėl svarbu užtikrinti komunikacijos kanalų, kuriais šie duomenys pasiekia kitus servisus, saugą. Komunikacijai tarp servisų dažniausiai naudojamas HTTP protokolas. Mikropaslaugų architektūros servisai dažnu atveju projektuojami taip, kad pasitikėtų užklausomis, atkeliaujančiomis iš tarpusavyje komunikuojančių servisų. Jeigu pažeidžiamas vieno iš servisų prieigos valdymo mechanizmas, pasitikėjimas gali būti išnaudojamas pažeisti kitus paslaugą aptarnaujančius servisus. Taip pat visa komunikacija tarp mikropaslaugų architektūros servisų vyksta tinklu, priešingai nei monolitinės architektūros programos atveju. Dėl šios priežasties iškyla būtinybė kiekvieną servisą saugoti individualiai dėl padidėjusių galimybių išnaudoti pažeidžiamumus tinklu [3].

1.3. Prieigos valdymo metodai tarp mikropaslaugų architektūros servisų

Mikropaslaugų architektūros servisai yra nepriklausomi vienas nuo kito, tačiau vykdant programos užduotis gali prireikti duomenų apsikeitimo tarp jų. Kaip vaizduojama 4 paveiksle, krepšelio servisas turi kreiptis į du kitus servisus (produktų katalogo ir kainoraščio), kad gautų visą reikalingą informaciją apie produktą ir galėtų atlikti savo funkciją.

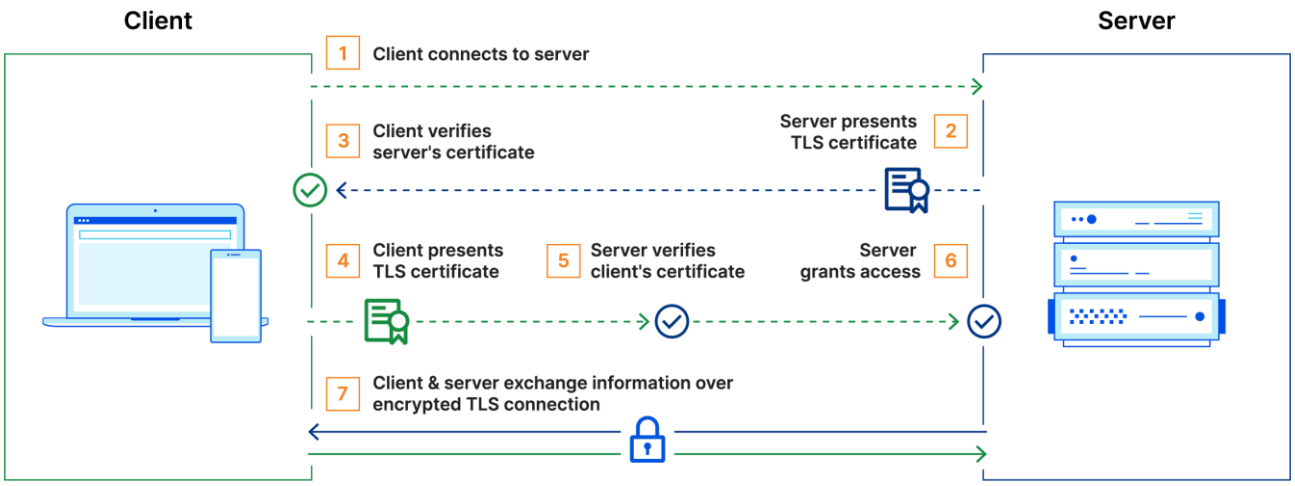


4 pav. Užklausos tarp mikropaslaugų architektūros servisų [8]

Dažnu atveju servisai turi jiems pritaikytus API, kurie yra naudojami gauti duomenis, kuriuos gali teikti servisas. Duomenys tarp servisų siunčiami tinklu todėl kyla rizika, kad perdavimo metu gali įvykti jų perėmimas arba bus bandoma pasiekti servisus be prieigos leidimo. Dėl šios priežasties servisams komunikuojant tarpusavyje yra naudojami prieigos valdymo metodai, kurie užtikrina, kad duomenimis keičiasi identifikuoti servisai. Be to, prieigos valdymo metodai saugo perduodamų duomenų konfidencialumą ir vientisumą. Pagrindiniai metodai, kurie yra naudojami prieigos valdymui tarp servisų mikropaslaugų architektūroje – mTLS ir žetonais paremti metodai [9].

1.3.1. mTLS prieigos valdymo metodas

mTLS metodas remiasi TLS protokolu. Tai dvikryptės autentifikacijos metodas, kurio metu komunikuojantys mikropaslaugų architektūros servisai patvirtinta savo tapatybę tarpusavyje naudodami viešo ir privataus raktų poras bei TLS sertifikatus [9]. 5 paveiksle pavaizduota mTLS metodo veikimo schema. Komunikacija tarp mikropaslaugų architektūros vidinių servisų vyksta kliento-serverio principu. Besikreipiantis servisas susijungia su servisu, iš kurio nori gauti duomenis. Tuomet antrasis servisas pateikia savo sertifikatą, kurį turi patvirtinti besikreipiantis servisas. Įvykus sertifikato patvirtinimui, besikreipiantis servisas perduoda savo sertifikatą servisui, į kurį kreipiasi, kad šis taip pat įvykdytų sertifikato patvirtinimą. Jeigu abu sertifikatai yra sėkmingai patvirtinami, antrasis servisas suteikia prieigą besikreipiančiam servisui ir įvykdo užklausą. Duomenys tarp servisų tinklu siunčiami užšifruoti tokiu būdu užtikrinant jų konfidencialumą ir vientisumą [10].



5 pav. mTLS metodo veikimo schema [10]

mTLS prieigos valdymo metodas yra populiariausias būdas autentifikuoti mikropaslaugų architektūros servisus jiems komunikuojant tarpusavyje. Pagrindiniais iššūkiais, naudojant šį metodą, įvardijama viešų ir privačių raktų priežiūra, jų rotacija ir sertifikatų administravimas [9].

1.3.2. Žetonais paremtas prieigos valdymo metodas

Žetonais paremtas prieigos valdymas remiasi kriptografiniais žetonais, kurie gali saugiai perduoti duomenis tarp servisų. Vienas iš standartų, naudojamų mikropaslaugų architektūroje, yra JSON Web Token (JWT) [11]. Duomenys yra perduodami JSON objekte, kuris pasirašomas skaitmeniniu parašu. Parašas gali būti sudaromas pagal slaptą žymą arba viešo ir privataus raktų porą. Kriptografinį JWT žetoną sudaro trys dalys:

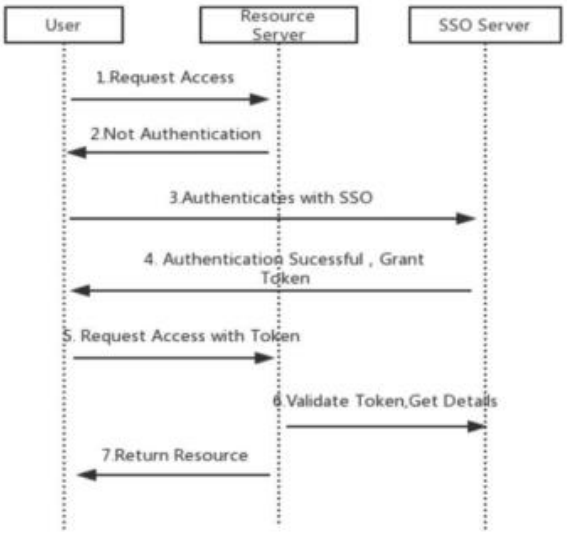
6. Antraštė;
7. Apkrova;
8. Parašas. [12]

Antraštėje saugoma informacija apie pasirašymo algoritmą ir žetono tipą. Apkrovoje įterpiami perduodami ir papildomi duomenys. Parašui sukurti naudojama užkoduota antraštė, užkoduota apkrova, slapta žyma ir viskas pasirašoma numatytu pasirašymo algoritmu [12]. Kiekvienas mikropaslaugų architektūros servisas gali kreiptis į žetonų išdavimo servisą su savo duomenimis, kad gautų servisui skirtą žetoną, kuris vėliau pridedamas prie kiekvienos užklausos, siunčiamos iš serviso [9]. Tuomet žetonas naudojamas patikrinimui, ar besikreipiantis servisas gali gauti duomenis iš serviso, į kurį kreipiasi, pagal tai, kokia informacija saugoma žetone. Kriptografiniai žetonai pasižymi lengvu naudojimu ir paprastumu [13] todėl gali būti naudojami situacijose, kuriose reikia minimizuoti programos resursų naudojimą.

1.4. Paslaugų prieigos valdymo metodai

1.4.1. Vieno prisijungimo sistema (SSO)

Vieno prisijungimo sistemos paskirtis – autentifikuoti vartotoją vieną kartą ir suteikti prieigą prie sistemoje esančių resursų be pakartotinio autentifikavimo. Autentifikacijai reikalingas SSO serveris, kuris išduoda prieigos žetonus ir tikrina jų būsenas. 6 paveiksle vaizduojama schema su vieno prisijungimo autentifikavimo algoritmu. Vartotojas kreipiasi į serverį, kuriame yra reikalingi resursai ir šis serveris tikrina, ar vartotojo užklausos antraštė turi prieigos žetoną. Jeigu antraštėje prieigos žetono nėra, užklausa yra atmetama ir vartotojas privalo autentifikuotis SSO serveryje, kad gautų prieigos žetoną. Atlikus autentifikaciją SSO serveryje, pastarasis išduoda prieigos žetoną, kuris įtraukiamas į užklausų antraštę. Jeigu resursų serveris gauna užklausą, kurios antraštėje pateiktas prieigos žetonas, tuomet jis kreipiasi į SSO serverį prieigos žetono patvirtinimui ir duomenų apie vartotoją gavimui. Sėkmingai patvirtinus prieigos žetoną SSO serveryje ir gavus vartotojo duomenis, resursų serveris grąžina užklausoje prašomus resursus vartotojui [14].



6 pav. Vieno prisijungimo autentifikavimo algoritmas [14]

Mikropaslaugų architektūros atveju, resursų serveriai yra servisai. Vieno prisijungimo sistema gali būti naudojama mikropaslaugų architektūroje tiek suteikiant prieigą vartotojams, tiek vykdant prieigos valdymą tarp atskirų servisų. Tačiau taikant vieno prisijungimo sistemos metodą prieigos valdymui tarp servisų kyla iššūkių dėl sistemos veikimo greičio [15]. Kiekvienas servisas, norėdamas patikrinti prieigos žetoną, turės kreiptis į SSO serverį ir gauti patvirtinimą prieš atsakant į užklausą. Esant dideliam kiekiui servisų arba užklausų susidaro didelis kiekis kreipinių į SSO serverį. Tai gali neigiamai paveikti užklausų įvykdymo laiką tiek SSO serveryje, tiek tarp servisų ir galiausiai sulėtinti visos sistemos darbą.

1.4.2. OAuth 2.0

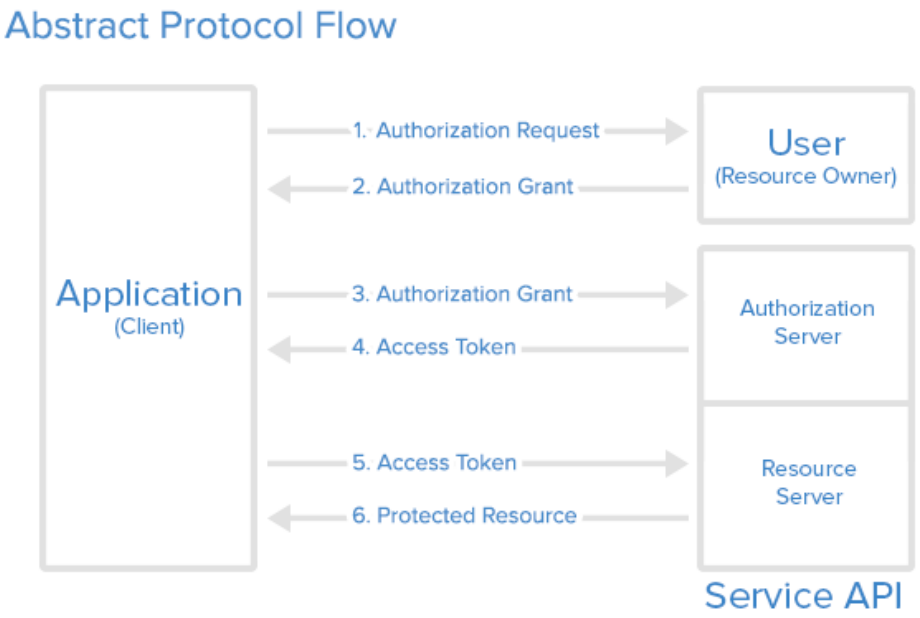
OAuth 2.0 yra autorizacijos protokolas, kuris leidžia suteikti prieigą vartotojo vardu prie kitoje programoje ar servise esančių apsaugotų resursų. Šis protokolas turi galimybę apriboti leidžiamus veiksmus, kuriuos programa atlieka su apsaugotais resursais vartotojo vardu. OAuth 2.0 nėra autentifikacijos protokolas, jis tik suteikia prieigą prie apsaugotų resursų [16].

Autorizacijai OAuth 2.0 protokolas naudoja prieigos žetonus, kurie dažnu atveju yra paremti JSON Web Token (JWT) formatu. Prieigos žetonai saugo informaciją, reikalingą autorizacijai ir apimties nustatymus, kurie nusako reikalavimus arba priežastį, dėl kurios turi būti suteikta prieiga prie tam tikrų apsaugotų resursų [16].

OAuth 2.0 autorizacijos protokolo elementus nusako keturi vaidmenys:

1. Resurso savininkas, kuris turi galimybę suteikti prieigą prie apsaugoto resurso;
2. Resurso serveris, kuriame talpinamas saugomas resursas ir jis turi galimybę priimti bei atsakyti į užklausas dėl apsaugoto resurso, panaudodamas prieigos žetonus;
3. Klientas, tai programa, kuri siunčia užklausą dėl apsaugoto resurso jo savininko vardu ir autorizacija;
4. Autorizacijos serveris, kuris išduoda prieigos žetonus klientui po sėkmingai atliktos resurso savininko autentifikacijos ir autorizacijos gavimo. [17]

7 paveiksle pateikiama OAuth 2.0 protokolo algoritmo schema. Visų pirma, klientas kreipiasi į resurso savininką, kad būtų suteiktas autorizacijos leidimas pasiekti apsaugotą resursą. Tuomet resurso savininkas suteikia autorizacijos leidimą klientui. Klientas kreipiasi į autorizacijos serverį, kuriame įvykdoma kliento autentifikacija ir pateikiamas autorizacijos leidimas. Sėkmingai autentifikavus klientą, autorizacijos serveris išduoda prieigos žetoną klientui, kuris yra panaudojamas kreipiantis į resurso serverį dėl apsaugoto resurso gavimo. Resurso serveris atlieka prieigos žetono patvirtinimą ir sėkmingai patvirtinus žetoną įvykdo kliento užklausą [17].

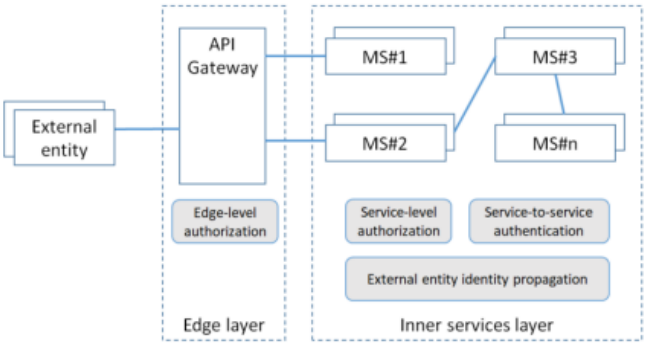


7 pav. OAuth 2.0 protokolo algoritmas [18]

OAuth 2.0 protokolas taip pat gali būti naudojamas mikropaslaugų architektūroje, tačiau esant didesniam servisų kiekiui gali iškilti problema, kuomet kiekvienas servisas turės individualiai tikrinti užklausų žetonus, o tai gali padidinti apkrovą servisams. Dėl šios priežasties rekomenduojama autorizaciją atlikti išoriniuose mikropaslaugų architektūros sluoksniuose [15].

1.5. Esami prieigos valdymo sprendimai mikropaslaugų architektūrai

Mikropaslaugų architektūros prieigos valdymą galima išskirti į du sluoksnius – išorinį ir vidinį. Kiekvienas sluoksnis turi jam būdingas funkcijas. 8 paveiksle pateikiama schema, kurioje vaizduojami išorinis ir vidinis sluoksniai bei jiems būdingos funkcijos. Išoriniame sluoksnyje atliekama tik išorinė autorizacija. Vidiniame sluoksnyje atliekama vidinė autorizacija, autentifikacija tarp servisų ir išorinės tapatybės sklaida. [9]



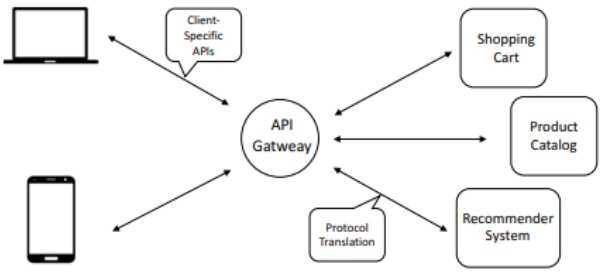
8 pav. Mikropaslaugų architektūros prieigos valdymo sluoksniai [9]

Toliau šiame skyriuje analizuojami esami prieigos valdymo sprendimai išoriniam ir vidiniam sluoksniui.

1.5.1. Išorinio sluoksnio prieigos valdymas

Mikropaslaugų architektūra paremtos paslaugos turi daug prieigos taškų [19] todėl siekiant paprastesnio prieigos valdymo ir saugumo sumažinant prieigos taškus galima naudoti API šliuzą. Kaip vaizduojama 9 paveiksle, šliuzas tampa vienu įėjimo tašku visai mikropaslaugų architektūra paremtai sistemai. API šliuzas taip pat atlieka maršrutizavimo funkciją, nukreipdamas užklausas į reikiamus servisus, gali surinkti atsakymus į užklausas iš keleto skirtingų servisų ir pakeisti užklausų protokolus, jeigu to reikalauja serviso procesai. Šliuzui galima priskirti ir tokias funkcijas, kaip:

- Autentifikacija;
- Monitoringas;
- Apkrovų valdymas. [20]



9 pav. API šliuzo schema [20]

API šliuzas skirtas tam, kad padidintų sistemos efektyvumą ir supaprastintų naudojimąsi mikropaslaugų architektūra paremta paslauga. Taip pat sumažintų užklausų, siunčiamų iš kiekvieno kliento, skaičių [20].

1.5.2. Vidinio sluoksnio prieigos valdymas

Vidiniame mikropaslaugų architektūros prieigos valdymo sluoksnyje sprendimai skirstomi pagal tai, kur vyksta sprendimų apie prieigos suteikimą vykdymas. Jeigu sprendimai priimami serviso viduje, tai yra decentralizuotas prieigos valdymas, priešingu atveju – centralizuotas.

Decentralizuotas prieigos valdymas prie mikropaslaugų architektūros serviso numato visų komponentų, reikalingų priimti sprendimą, integravimą į serviso kodą. Prieigos valdymo politika ir taisyklės nustatomos ir saugomos kiekviename servise individualiai. Kai klientas kreipiasi į servisą, jis išanalizuoja visus gautus duomenis apie klientą ir priima sprendimą dėl prieigos suteikimo pagal tam servisui pritaikytas taisykles [9].

Decentralizuotas prieigos valdymas leidžia numatyti specifines, individualiai kiekvienam servisui pritaikytas prieigos valdymo taisykles pagal jo atliekamas funkcijas. Tačiau tai reiškia, kad serviso konfigūravimo procesas tampa sudėtingesnis, iškyla klaidų padarymo rizika. Be to, plėtojant servisą turi būti užtikrinta, kad serviso prieigos valdymas atitinka bendrą sistemos politiką ir keičiantis šiai politikai, servise esančios taisykles taip pat turi būti keičiamos [9].

Centralizuoto prieigos valdymo atveju, visa informacija, reikalinga priimti sprendimui dėl prieigos suteikimo, saugoma centralizuotai tam skirtame servise. Kai klientas kreipiasi į servisą, šis siunčia užklausą į prieigos valdymo politikos priėmimo tašką, kuris pateikia sprendimą pagal gautų duomenų palyginimą su numatytomis taisyklėmis. Prieigos valdymo politikos priėmimo taškas gali būti įtraukiamas į servisą, tačiau sprendimo priėmimui reikalingos taisyklės vis tiek yra gaunamos su užklausa iš išorės šaltinių [9].

Centralizuoto prieigos valdymo privalumai yra tai, kad prieigos valdymas gali būti valdomas bendrai tarp servisų pagal numatytą politiką, politikos pakeitimai atliekami atskirai nuo servisų ir nedaro įtakos jų veikimui. Tačiau centralizuotas prieigos valdymas turi trūkumą dėl didelio kiekio papildomų užklausų, kurios reikalingos gauti sprendimų priėmimo informaciją. Tai gali neigiamai paveikti servisų užklausų vykdymo laiką [9].

1.6. Analizės apibendrinimas ir išvados

Sistemos, sukurtos pagal mikropaslaugų architektūrą, remiasi funkcijų paskirstymu į nepriklausomus, vienas nuo kito izoliuotus, mažos apimties servisus. Tai leidžia pasiekti didesnį sistemos pasiekiamumą, patikimumą, optimizuoti valdymą ir apimties plėtrą. Mikropaslaugų architektūros naudojimas sparčiai plečiasi, ypatingai verslo srityje. Dėl šios priežasties mikropaslaugų architektūra paremtų sistemų saugumas yra itin aktualus, ypač prieigos valdymas. Mikropaslaugų architektūra turi jai būdingų iššūkių, kurių nebuvo naudojant monolitinę sistemų architektūrą. Vienas iš jų – vietoje vieno sistemos taško privaloma galvoti apie kiekvieno serviso saugą individualiai. Taip pat būtina užtikrinti, kad komunikacija tarp servisų būtų saugi, pasitikėjimas užklausomis nebūtų pažeistas. Palaikyti mikropaslaugų architektūra paremtos sistemos apsaugą padeda prieigos valdymo metodai. Analizės metu išskirti 4 prieigos valdymo metodai, naudojami tiek išoriniam, tiek vidiniam prieigos valdymui: mTLS, žetonais paremtas prieigos valdymo metodas JSON Web Token (JWT), vieno prisijungimo sistema (SSO) ir OAuth 2.0 protokolas.

Metodų palyginimui parinkti kriterijai, kurie teiktų naudą sistemos saugumui, veikimui, valdymui ir naudojimui. 1 lentelėje pateikiamas prieigos valdymo metodų palyginimas pagal 5 kriterijus: komunikacijos kanalo saugos tarp servisų užtikrinimą, mažą kiekį autorizacijos ir autentifikacijos užklausų, galimybę naudoti vaidmenimis pagrįstą prieigos valdymo modelį, patogų naudojimąsi klientui, paprastą trečiosios šalies naudotojo integravimą.

1 lentelė Prieigos valdymo metodų palyginimas

	mTLS	**JWT**	**SSO**	**OAuth 2.0**
Užtikrinama komunikacijos kanalo sauga tarp servisų	Taip	Ne	Ne	Ne
Mažas kiekis autorizacijos ir autentifikacijos užklausų	Taip	Taip	Ne	Ne
Galimybė naudoti vaidmenimis pagrįstą prieigos valdymo modelį (RBAC)	Ne	Taip	Taip	Taip
Patogus naudojimasis klientui	Ne	Taip	Taip	Taip
Paprastas trečiosios šalies naudotojo integravimas	Ne	Ne	Ne	Taip

Atrinkti metodai gali būti pritaikyti išoriniame ir vidiniame mikropaslaugų architektūros sluoksnyje. Taip pat tiek centralizuotame, tiek decentralizuotame prieigos valdymo modelyje.

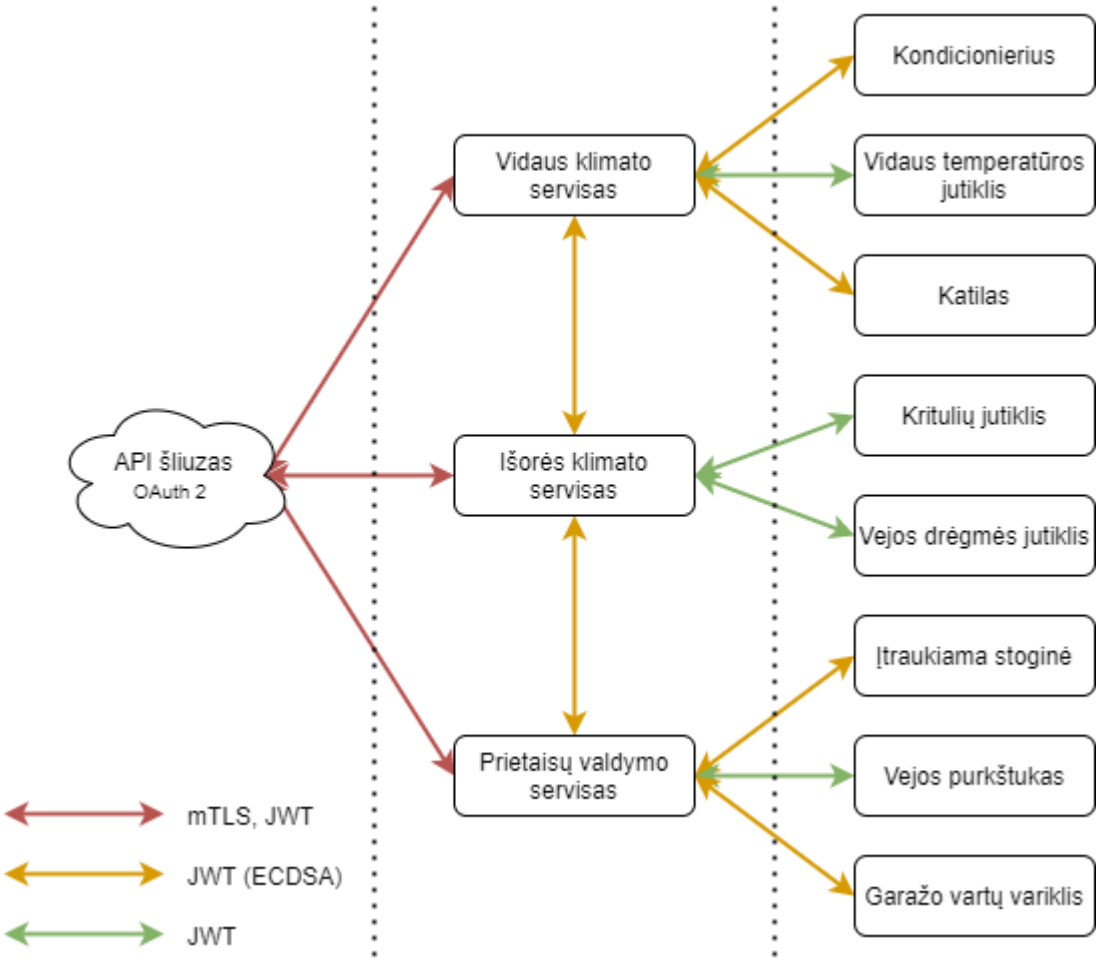
Apibendrinant analizės metu surinktą informaciją galima teigti, kad:

1. Mikropaslaugų architektūra turi ne vieną privalumą, kuris suteikia jai pranašumą prieš monolitine architektūra paremtas sistemas. Dėl šios priežasties mikropaslaugų architektūros naudojimas sparčiai plečiasi ir būtina užtikrinti šia architektūra paremtų sistemų saugą;
2. Naujus saugumo iššūkius projektuojant ir diegiant mikropaslaugų architektūra paremtą sistemą kelia tai, kad priešingai nei monolitinės architektūros atveju, sistema turi daug servisų, kuriuos reikia saugoti individualiai;
3. Pagrindiniai saugumo iššūkiai, susiję su prieigos valdymu, atsiranda komunikacijoje tarp servisų, jų tarpusavio pasitikėjimo išlaikyme. Taip pat iššūkius kelia tai, kad servisai komunikuoja atvirame tinkle, o tai padidina riziką individualių servisų pažeidžiamumams tinklu;
4. Iškylančias prieigos valdymo saugumo problemas galima spręsti su jau egzistuojančiais prieigos valdymo metodais ir tobulinant esamus prieigos valdymo sprendimaus mikropaslaugų architektūrai.

2. Prieigos valdymo metodo projektas

2.1. Mikropaslaugų architektūros sprendimas

Prieigos valdymo metodas yra projektuojamas mikropaslaugų architektūroje, sudarytoje iš trijų sluoksnių – API šliuzo, ūko sluoksnio ir galinių įrenginių. 10 paveiksle esančioje schemoje pavaizduota mikropaslaugų architektūra iš 12 servisų ir įrenginių.



10 pav. Mikropaslaugų architektūros prototipas (šaltinis: sukurta autoriaus)

Vartotojo sąsaja su API šliuzu paremta REST stiliumi, kuriame naudojamos HTTP užklausos GET, POST, PUT ir pan. Ši sąsaja gali būti naudojama programose, interneto ir mobiliųjų įrenginių programose todėl siūlomas prieigos valdymo metodas yra universalus, nepriklausomas nuo konkrečios rūšies įrenginių. API šliuzas veikia serveryje, kuris saugo visų vartotojų, galinčių naudotis mikropaslaugų architektūroje esančiais servisais, duomenis ir vykdo prieigos bei kreipinių valdymą. Jis taip pat apsaugo ūko ir galinių įrenginių sluoksniuose esančius serverius bei įrenginius nuo išorinių grėsmių, kadangi visi kreipiniai yra vykdomi per šliuzą, o tolimesni sluoksniai paslėpti nuo išorinės tinklo prieigos. API šliuzas su ūko sluoksnyje esančiais serveriais ir ūko sluoksnio serveriai su galiniais įrenginiais, kuriuose veiks mikropaslaugų servisai, komunikacijai taip pat naudos HTTP užklausas pagal REST API.

2.2. Prieigos valdymo metodas

Prieigos valdymui API šliuze bus naudojamas OAuth 2.0 protokolas. Vartotojai galės autorizuotis API šliuze ir gauti OAuth 2.0 prieigos žetoną, kuris paremtas JSON Web Token (JWT). Kadangi vartotojui suteikiamas JWT prieigos žetonas todėl jis bus suderinamas perdavimui į tolimesnius sluoksnius mikropaslaugų architektūroje. OAuth 2.0 protokolas prieigos žetone leidžia pateikti apimties nustatymus, kurie nusako prieigos teises. Šis funkcionalumas bus naudojamas tam, kad būtų įgyvendintas vaidmenimis pagrįstas prieigos valdymo modelis (RBAC) ir įrenginiai, komunikuojantys tarpusavyje, galėtų nustatyti, ar suteikti besikreipiančiam įrenginiui prieigą prie reikalaujamo resurso. Kaip papildomą privalumą galima išskirti galimybę trečiųjų šalių naudotojams pasiekti mikropaslaugų architektūroje veikiančius servisus autorizuojantis jų naudojamose paslaugose ir platformose.

API šliuzas ir ūko sluoksnio serveriai prieigos valdymui naudos mTLS metodą, kuris paremtas TLS protokolu. Šis metodas užtikrins saugą API šliuzui komunikuojant su ūko sluoksnio serveriu ir naudos viešo bei privataus raktų porą su TLS sertifikatais tam, kad abu įrenginiai patvirtintų savo tapatybę vienas kitam ir prieiga būtų suteikiama tik tiems įrenginiams, kuriems ji yra numatyta išduodant raktus. Užtikrinus saugų komunikavimo kanalą toliau perduodami JWT prieigos žetonai. mTLS metodas pasirinktas todėl, kad jis leidžia užtikrinti perduodamų duomenų konfidencialumą ir API šliuzas bei ūko sluoksnio serveriai turi pakankamai resursų, jog galėtų naudoti šį metodą.

Ūko sluoksnio serveriai tarpusavyje naudos JWT prieigos žetonus, kurie pasirašomi ECDSA algoritmo viešo ir privataus raktų pora. Šis sprendimas leidžia užtikrinti žetone esančių duomenų konfidencialumą ir vientisumą bei prieigos valdymą su raktų pora, kadangi žetono informaciją galės pasiekti tik tas įrenginys, kuris turi viešąjį žetono siuntėjo raktą. JWT žetono pasirašymas raktu reikalauja papildomų resursų todėl metodas naudojamas tik tuose įrenginiuose, kurie turi didesnį resursų rezervą ir gali atlikti pasirašymą raktu. JWT prieigos žetonų metodas su pasirašymu ECDSA algoritmo raktais bus naudojamas ir komunikacijai tarp ūko sluoksnio serverių bei kai kurių galinių įrenginių, kuriems yra numatytas didesnis resursų kiekis ir jų galimybės leidžia naudoti šį metodą. Ūko sluoksnio serveriai parinks prieigos valdymo metodą pagal turimus duomenis apie galinį įrenginį, su kuriuo bus užmezgamas ryšys ir perduodami arba gaunami duomenys. Šiuos duomenis apima galinio įrenginio resursų apribojimai, saugumo reikalavimai.

Dauguma paskutiniame, galinių įrenginių, sluoksnyje esančių prietaisų neturi daug resursų ir galimybių palaikyti didesnės saugos metodus. Dėl šios priežasties galiniams įrenginiams numatyti įprasti, lengvasvoriai JWT žetonai. Labai mažai resursų turintys įrenginiai gali naudoti statinius prieigos žetonus, kurie yra iš anksto sugeneruoti ir papildomų resursų šiam procesui nereikia. Jeigu įrenginys turi galimybę pats generuoti JWT žetonus su pasirašymu naudojant HMAC algoritmą, tuomet jame bus naudojamas minėtasis metodas. Tiek statiniai, tiek generuojami JWT prieigos žetonai suteikia duomenims saugumo užtikrinant jų integralumą bei leidžia valdyti įrenginių prieigos teises.

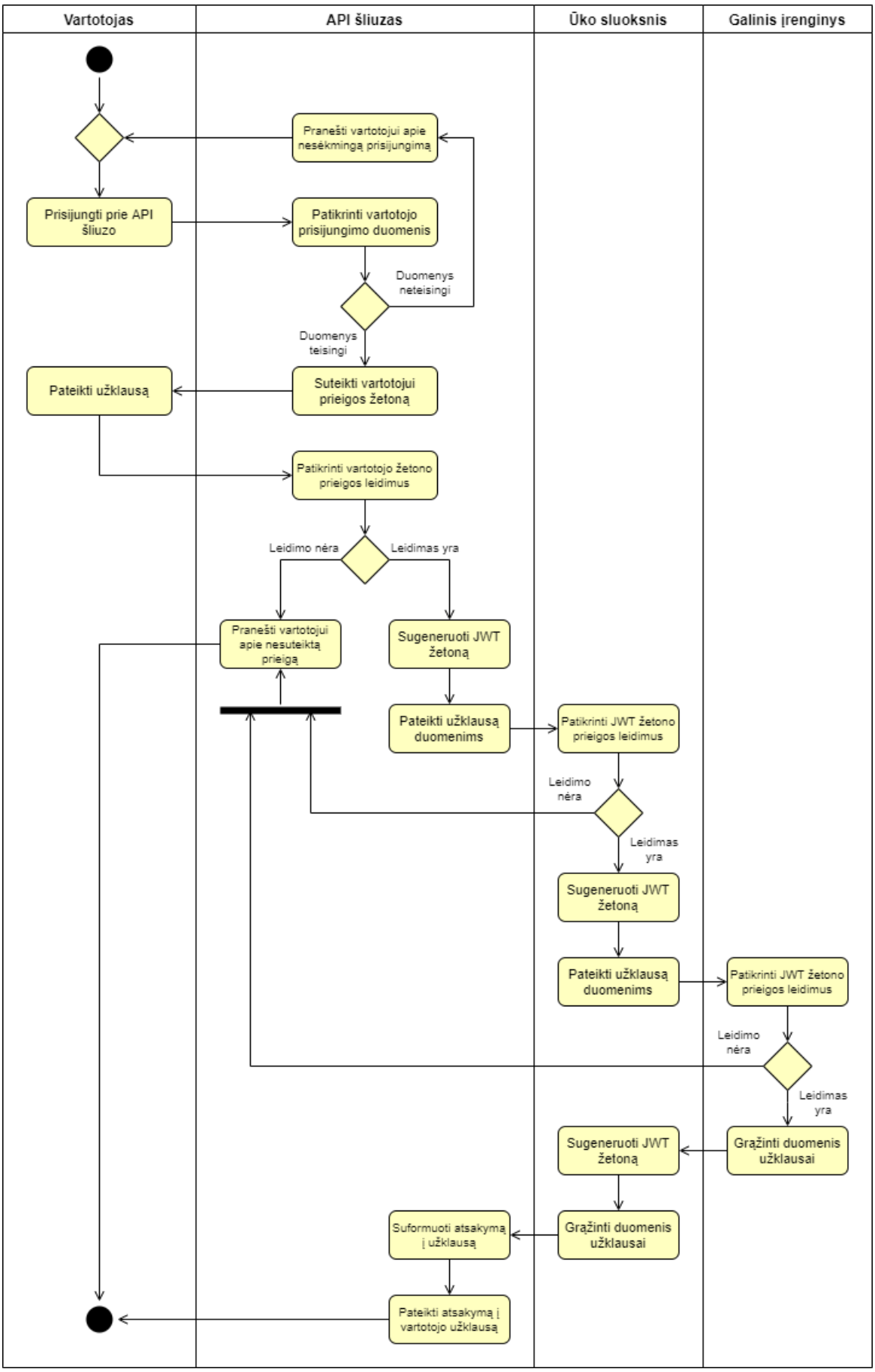
Visi pasirinkti metodai leis valdyti prieigą nuo vartotojo iki galinio įrenginio užtikrinant duomenų saugą per konfidencialumą ir integralumą. Be to, metodai yra parinkti atsižvelgiant į įrenginių galimybes ir resursų apribojimus, siekiant siūlomą prieigos valdymo metodą pritaikyti prie kuo platesnio spektro įrenginių ir padidinti jų saugą.

2.3. Prieigos valdymo metodo procesas

Vartotojas, norėdamas naudotis funkcijomis, teikiamomis mikropaslaugų architektūroje, turės kreiptis į API šliuzą, kuris yra atsakingas už kreipinių ir prieigos valdymą, duomenų persiuntimą ir surinkimą iš ūko sluoksnio. API šliuze saugomi vartotojų duomenys ir prieigos teisės, kurios yra įrašomos į šliuzo suteikiamus prieigos žetonus, naudojamus pasiekti norimas funkcijas ir duomenis iš mikropaslaugų servisų. Kaip vaizduojama 11 paveiksle pateiktoje prieigos valdymo metodo proceso diagramoje, vartotojas autentifikuojamas ir autorizuojamas API šliuze. Prieš pateikiant užklausą dėl duomenų gavimo arba funkcijos vykdymo, vartotojas autentifikuojasi šliuze pateikdamas savo vartotojo vardą ir slaptažodį. Po autentifikavimo ir prieigos žetono suteikimo, vartotojas gauna galimybę teikti užklausas šliuzui. Vartotojui pateikus užklausą funkcijos įvykdymui ar duomenų gavimui, vykdoma seka, kuri yra aprašyta 2.4. skyriuje.

API šliuzo sugeneruotame prieigos žetone yra saugomi duomenys apie vartotoją, kuriam šis žetonas buvo sugeneruotas ir sąrašas leidimų, kuriais remiantis šliuzas turi galimybę patikrinti, ar užklausa, kurią pateikė vartotojas, gali būti vykdoma pagal šliuze nustatytas prieigos valdymo taisykles. Vėliau, šliuzui generuojant naują žetoną užklausai į ūko sluoksnio serverius, prie vartotojo duomenų pridedami API šliuzo duomenys, kurie bus panaudojami ūko sluoksnyje esančiame serveryje. Analogiškai ūko sluoksnio serveris, ruošdamas užklausą galiniam įrenginiui, sugeneruos žetoną, kuriame pateiks savo duomenis. Remiantis 2 skyriuje aprašytais ribojimais, žetonai, siunčiami galiniams įrenginiams, neturės visų duomenų apie vartotoją ir API šliuzą, kadangi jie būtų pertekliniai ir be būtinybės padidintų resursų naudojimą.

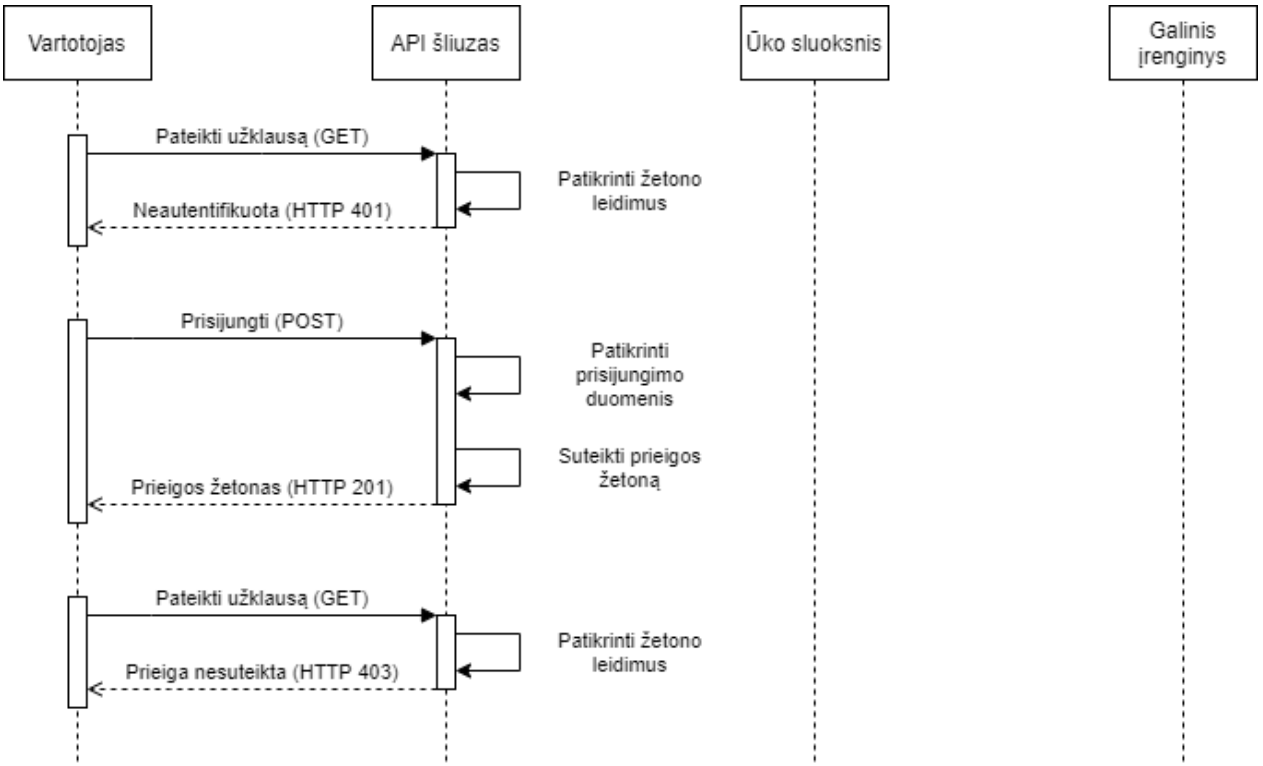
Prieigos žetonai yra generuojami kiekviename sluoksnyje, kai vyksta užklausos siuntimas tarp įrenginių, nes tai leidžia kiekvienam įrenginiui pridėti papildomus duomenis prie pradinio, vartotojui suteikto, prieigos žetono, kurie naudojami užklausą gaunančiame įrenginyje. Šie duomenys leis identifikuoti įrenginį, kuris siunčia užklausą ir patikrinti, ar įrenginiui, atsiuntusiam užklausą, galima suteikti prieigą prie duomenų ar funkcijos įvykdymo pagal gaunančiame įrenginyje numatytas prieigos valdymo taisykles.



11 pav. Prieigos valdymo metodo proceso diagrama (šaltinis: sukurta autoriaus)

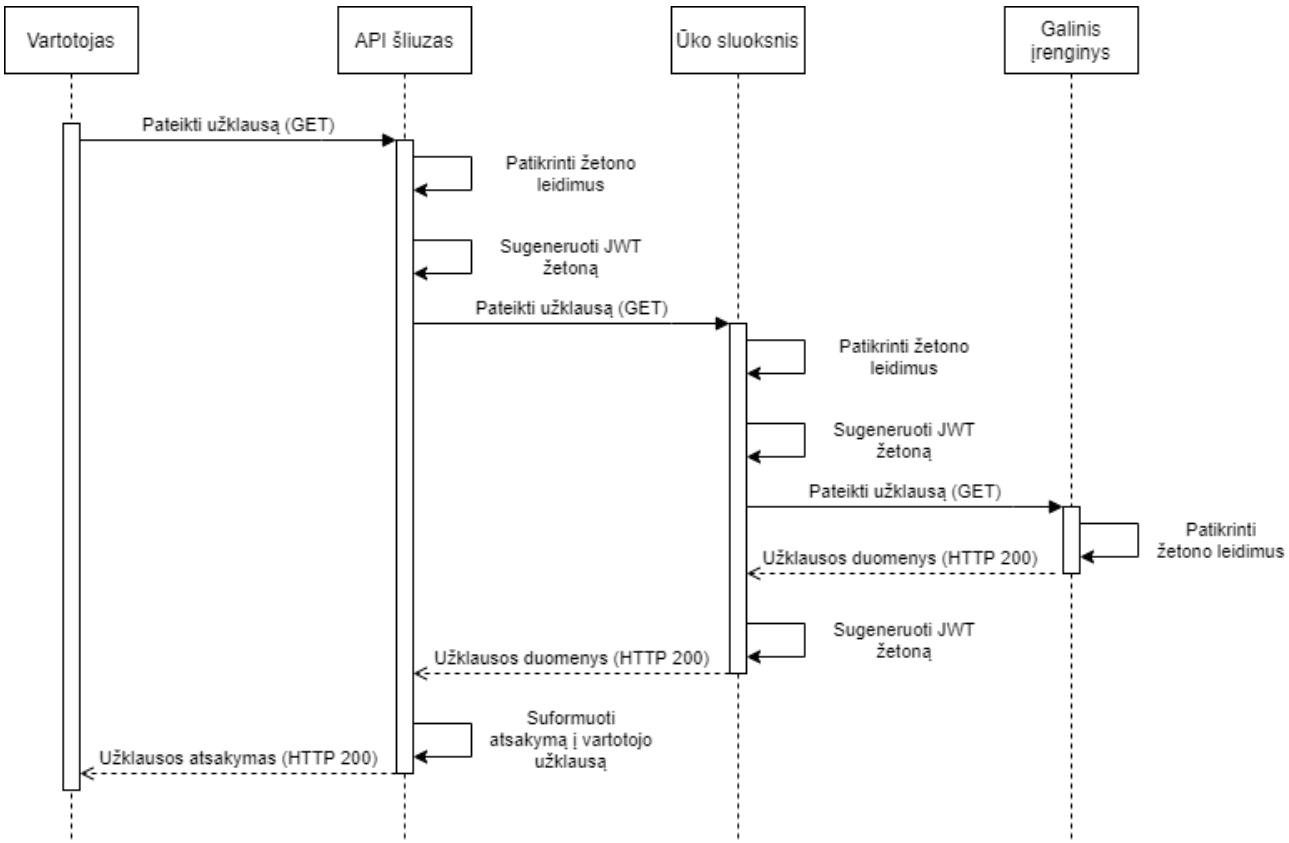
2.4. Prieigos valdymo metodo seka

Komunikacijai tarp vartotojo, API šliuzo ir servisų bus naudojama REST API, kuri paremta HTTP užklausomis. Kaip vaizduojama 12 paveiksle esančioje diagramoje, tam, kad vartotojas galėtų pasiekti API šliuzo funkcijas, jis turės autentifikuotis su vartotojo vardu ir slaptažodžiu. Jeigu vartotojas teiks užklausas be prieigos žetono, jam bus grąžinami atsakymai su 401 HTTP klaidos kodu, kuris nusako, kad vartotojas neautentifikuotas. Vartotojui pateikus prisijungimo duomenis ir API šliuzui juos patvirtinus bus suteikiamas prieigos žetonas, kuris naudojamas teikiant užklausas. API šliuzas generuoja JWT žetonus, kuriems priskiriamos atitinkamos teisės, kurias nusako API šliuze saugomi duomenys apie vartotoją. Vartotojui teikiant užklausą tam tikrai funkcijai su prieigos žetonu, API šliuzas patikrina žetono leidimus ir, jeigu žetono leidimai nesuteikia teisės naudoti funkcijos, grąžina vartotojui 403 HTTP klaidos kodą su pranešimu, kad šiai užklausai prieiga nesuteikta.



12 pav. Prieigos valdymo metodo autentifikacijos ir autorizacijos sekos diagramos (šaltinis: sukurta autoriaus)

Diagramoje, esančioje 13 paveiksle, pavaizduota seka, kai vartotojas pateikia užklausą su prieigos žetonu, API šliuzas jį patikrina ir suteikia prieigą prie tolimesnės sekos. Gavus prieigos leidimą, API šliuzas teikia naują užklausą serveriui, esančiam ūko sluoksnyje, kuris teikia arba gauna informaciją iš galinių įrenginių. Šiai užklausai naudojamas naujas JWT žetonas, kuriame saugomi tiek vartotojo, tiek API šliuzo prieigos leidimai. Ūko sluoksnyje esantis serveris patikrina žetono, gauto su užklausa, prieigos leidimus ir toliau vykdo reikalingas funkcijas arba siunčia atsakymą API šliuzui, kad prieiga nesuteikta. Jeigu žetone esantys prieigos leidimai atitinka serveryje nustatytas taisykles, tuomet ūko sluoksnio serveris kreipiasi į galinį įrenginį, kuriame veikia mikropaslauga, su savo sugeneruotu JWT prieigos žetonu ir pateikia arba gauna duomenis iš galinio įrenginio, kuris, palyginęs prieigos žetone esančius leidimus su jam numatytomis taisyklėmis, duomenis priima ir suteikia arba pateikia atsakymą dėl nesuteiktos prieigos.



13 pav. Prieigos valdymo metodo (šaltinis: sukurta autoriaus)

API šliuzas turės galimybę siųsti keletą užklausų vienu metu ir, gavus atsakymus į jas, agreguoti gautus duomenis, juos apdoroti ir suformuoti atsakymą vartotojui į jo pateiktą užklausą.

2.5. Prieigos valdymo metodo projekto išvados

Apžvelgus siūlomą prieigos valdymo metodo projektą galima daryti išvadą, kad:

1. CoAP protokolas su DTLS gali užtikrinti prieigos valdymą mikropaslaugų architektūroje, tačiau tam reikalingi papildomi resursai įrenginiams, kurių jie gali neturėti;
2. CoAP protokolui reikalinga alternatyva, galinti pasiūlyti prieigos valdymą su mažesniu resursų naudojimu;
3. Siūlomas prieigos valdymo metodas paremtas OAuth 2.0 protokolu, mTLS metodu ir JWT žetonais;
4. Komunikacija tarp projekto mikropaslaugų architektūros servisų ir įrenginių vykdoma naudojant HTTP užklausas.

3. Prieigos valdymo metodo prototipas

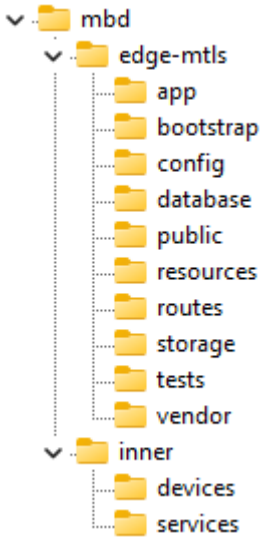
Šiame skyriuje pristatomas prieigos valdymo metodo mikropaslaugų architektūroje prototipas. Jis realizuotas vienoje mašinoje su „Linux“ „Ubuntu 20.04“ operacine sistema, kurioje veikia „Docker Engine“ konteinerizavimo programinė įranga. Technologijos, naudojamos prototipe, detalizuojamos 3.1. skyriuje.

3.1. Prieigos valdymo metodo prototipo technologijos

Prieigos valdymo metodo mikropaslaugų architektūroje prototipas realizuotas serveryje su „Nginx“ 1.80.0 versijos saityno serverio programine įranga ir „Docker“ konteineriuose su „Ubuntu 22.04“ operacine sistema bei „Nginx“ 1.24.0 versijos saityno serveriu. Prototipo loginėms operacijoms pasitelkta „PHP“ programavimo kalba su 8.2.13 versijos interpretatoriumi serveryje ir 8.3.4 – konteineriuose. Ši programavimo kalba pasirinkta dėl paprastos sintaksės, plačių funkcionalumų galimybių ir didelio bibliotekų bei programavimo karkasų pasirinkimo. Vartotojų duomenų saugojimui naudojama „MariaDB“ 10.3.38 versijos reliacinės duomenų bazės valdymo sistemos programinė įranga. Funkcionalumų įgyvendinimui panaudotos „Guzzle“ ir „PHP-JWT“ bibliotekos, skirtos „PHP“ programavimo kalbai. Pasirinktas technologijų rinkinys leidžia paprastai sukurti aplinką, kurioje gali būti diegiami mikropaslaugų architektūros servisai ir išbandomas darbe siūlomas prieigos valdymo metodas.

3.2. Prieigos valdymo metodo prototipo sandara

Prototipas skaidomas į dvi dalis – išorinį ir vidinį sluoksnius. Išoriniame sluoksnyje talpinamas API šliuzas, kuris detalizuojamas 3.2.1. skyriuje, o vidiniame – ūko ir galinių įrenginių sluoksniai, kurie detalizuojami atitinkamai 3.2.2. ir 3.2.3. skyriuose. 14 paveiksle pavaizduotas prototipo aplankų medis, kuriame galima matyti visą prototipo struktūrą.



14 pav. Prototipo aplankų medis (šaltinis: sukurta autoriaus)

3.2.1. API šliuzas

API šliuzas patalpintas serveryje, o jo realizavimui naudojamas „PHP“ programavimo kalbos karkasas „Laravel“. Prototipui pasirinkta programavimo metu buvusi naujausia karkaso versija – 11.3.1. „Laravel“ karkasas pasirinktas tam, kad būtų galima įgyvendinti OAuth 2.0 protokolo naudojimą prieigos valdymui. „Laravel“ karkase yra sukurtas „Laravel Passport“ modulis, kuris suteikia pilną OAuth 2.0 protokolo naudojimą šiuo karkasu paremtuose projektuose.

Vartotojai ir jiems priskirti vaidmenys saugomi reliacinėje duomenų bazėje, kurią API šliuzas gali pasiekti ir redaguoti. Klientas, norėdamas pasiekti API šliuzo paslaugas, turi prisijungti su jam skirtais prisijungimo duomenimis. Prisijungus prie API šliuzo, gaunami vartotojo vaidmenys, kurie yra naudojami prieigos valdymui vidiniuose sluoksniuose. Klientui pateikus užklausą, API šliuzas suformuoja JWT žetoną, kurį kartu su užklausa persiunčia į ūko sluoksnio atitinkamą servisą. 15 paveiksle pateiktas API šliuzo sugeneruotas JWT žetonas ūko sluoksnio servisui.

HEADER: ALGORITHM & TOKEN TYPE


```
{
  "typ": "JWT",
  "alg": "RS256"
}
```


PAYLOAD: DATA


```
{
  "aud": "4",
  "jti":
"7058d91df356766fb7eac9efdc6c29676a57ad426e16aad328c40b
93d21591f92f870c84874bcd30",
  "iat": 1713940494.171697,
  "nbf": 1713940494.171702,
  "exp": 1745476494.168152,
  "sub": "1",
  "scopes": [
    "owner"
  ],
  "iss": "54e1056f9be0"
}
```

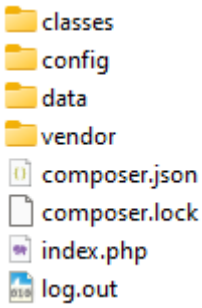

15 pav. API šliuzo sugeneruotas JWT žetonas (šaltinis: sukurta autoriaus)

API šliuzas žetonus pasirašo RSA algoritmo 256 bitų raktu, kaip nurodyta kintamajame ***alg***. Šių raktų poras turi API šliuzas ir trys ūko sluoksnio servisai. Ūko sluoksnio servisai tikrina gaunamus žetonus turimu raktu ir taip patvirtina, kad žetonas atkeliavo iš API šliuzo. Kintamasis ***jti*** saugo unikalią žetono reikšmę, kuri neleidžia pasinaudoti žetonu dar kartą. Taip pat žetonas turi laiko žymas kintamuosiuose ***iat***, ***nbf*** ir ***exp***, kuriuose atitinkamai nurodomi laikai, žymintys žetono išdavimą, priėmimo ribojimą ir galiojimą. Kintamasis ***sub*** saugo kliento, kuris pateikė užklausą, unikalų identifikavimo numerį, o ***scopes*** – kliento roles. Toliau žetone nurodomas servisas, išdavęs žetoną. Ši informacija talpinama kintamajame ***iss***, kuriame nurodomas serviso unikalus numeris.

Papildomam prieigos valdymui ir saugumui užtikrinti, tarp API šliuzo ir ūko sluoksnio servisų įgyvendintas mTLS prieigos valdymo metodas. Virtualūs saityno servisai turi jiems sugeneruotus sertifikatus, kuriais gali patvirtinti savo tapatybę persiunčiant užklausas vienas kitam.

3.2.2. Ūko sluoksnis

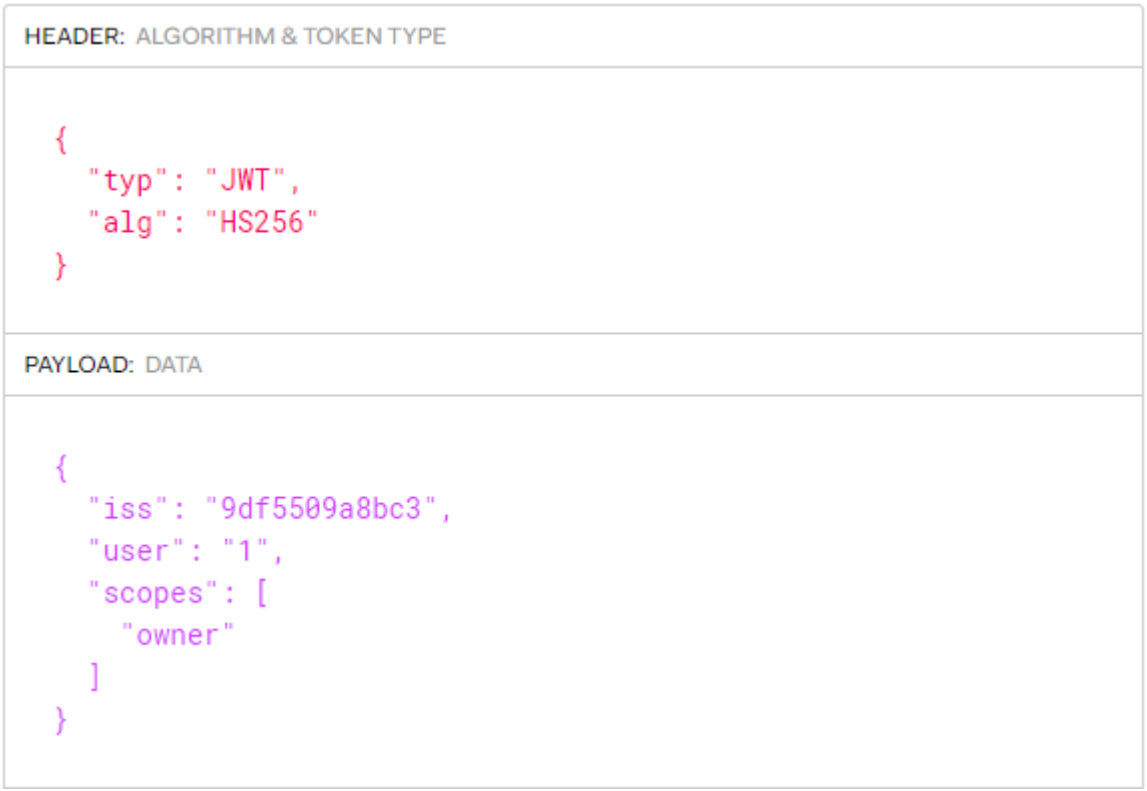
Ūko sluoksnyje suprogramuoti trys simuliaciniai servisai panaudojant „PHP“ programavimo kalbą. Servisai yra patalpinti „Docker“ konteineriuose, kuriuose ribojami resursai. Šie servisai gauna užklausas iš API šliuzo ir kreipiasi į reikiamus galinius įrenginius siekiant gauti arba perduoti duomenis. Kiekvienas servisas atlieka vaidmenimis grįstą prieigos valdymą pagal individualią konfigūraciją. 16 paveiksle vaizduojamos vieno ūko sluoksnio serviso rinkmenos ir aplankai.



16 pav. Ūko sluoksnio serviso rinkmenos ir aplankai (šaltinis: sukurta autoriaus)

Konfigūracijoje kiekvienam ūko sluoksnio servisui nustatomos prieigos teisės pagal kliento rolę. Taip pat nurodomos RSA ir ECDSA algoritmų raktų poros, kurios atitinkamai naudojamos komunikuojant su API šliuzu ir galiniais įrenginiais. Kartu sukonfigūruojamas sąrašas su servisų unikaliais numeriais, pagal kuriuos nustatoma, ar ūko sluoksnio servisas turi vykdyti gautą užklausą. Ūko sluoksnio servisai saugo duomenis apie savo unikalų numerį, naudojamą raktų algoritmą ir sąrašą galinių įrenginių, kuriuos gali pasiekti. Galinių įrenginių duomenis apima jų unikalūs numeriai, nuorodos, naudojami raktų algoritmai ir prieigos taškai.

Ūko sluoksnio servisai, remdamiesi turima informacija apie galinį įrenginį, parenka atitinkamą saugos algoritmą žetono pasirašymui. Jeigu galinis įrenginys palaiko ECDSA algoritmą, tuomet žetonas pasirašomas 256 bitų ECDSA raktu, priešingu atveju – naudojama maišos funkcija ir pasirašoma slaptu raktu. Tuomet žetonas siunčiamas galiniam įrenginiui. Ūko sluoksnio serviso sugeneruotas žetonas vaizduojamas 17 paveiksle.



17 pav. Ūko sluoksnio serviso sugeneruotas JWT žetonas (šaltinis: sukurta autoriaus)

Aukščiau pateiktas žetonas pasirašytas naudojant maišos funkciją ir slaptą raktą, kuris yra atsitiktiniu būdu sugeneruota tekstinė eilutė. Šį slaptą raktą turi ūko sluoksnio servisas ir galinis įrenginys. Žetone saugomas unikalus serviso, sugeneravusio žetoną, numeris, esantis kintamajame ***iss***. Šis kintamasis naudojamas galiniame įrenginyje tikrinant, ar ūko sluoksnio servisas turi prieigos teisę prie galinio įrenginio. Kintamasis ***user*** nurodo kliento, API šliuze pateikusio užklausą, unikalų identifikavimo numerį, o ***scopes*** – kliento roles.

3.2.3. Galinių įrenginių sluoksnis

Galinių įrenginių sluoksnį sudaro aštuoni simuliaciniai įrenginiai, suprogramuoti „PHP“ programavimo kalba ir patalpinti „Docker“ konteineriuose su ribotais resursais. Galiniai įrenginiai gauna užklausas iš ūko sluoksnio servisų ir remdamiesi konfigūracija, atlieka prieigos valdymą bei teikia duomenis servisui arba vykdo komandas. Įrenginio konfigūracijoje nurodomos prieigos teisės prie funkcijų pagal kliento rolę. Taip pat saugomas slaptas raktas, jeigu įrenginys neturi daug resursų arba ECDSA raktų pora, jeigu resursų kiekis yra didesnis. Konfigūracijoje nurodomas sąrašas ūko sluoksnio servisų unikalių numerių, kurie turi prieigos teisę prie galinio įrenginio. Jeigu besikreipiančio serviso šiame sąraše nėra, galinis įrenginys į užklausą atsako nesuteiktos prieigos žinute. Kiekvienas galinis įrenginys turi savo unikalų numerį, palaikomą saugos algoritmą ir tam tikrus duomenis, kuriuos gali pateikti ūko sluoksnio servisams.

3.3. Prieigos valdymo metodo prototipo apibendrinimas ir išvados

Darbe siūlomas prieigos valdymo metodas realizuotas prototipe pagal 2 skyriuje detalizuojamą projektą. Prototipui naudojama mašina su „Linux“ „Ubuntu 20.04“ operacine sistema, „Docker Engine“ konteinerizavimo programinė įranga, „Nginx“ saityno serveris, „PHP“ programavimo kalba, „MariaDB“ reliacinių duomenų bazių valdymo sistema ir „Guzzle“ bei „PHP-JWT“ bibliotekos, skirtos „PHP“ programavimo kalbai. Prieigos valdymo metodo prototipą sudaro du sluoksniai – išorinis ir vidinis, kuriuose realizuotas API šliuzas, 3 simuliaciniai ūko sluoksnio servisai ir 8 simulaciniai galiniai įrenginiai. Prototipe taikomas OAuth 2.0 protokolas, mTLS prieigos valdymo metodas ir JWT prieigos valdymo žetonai su trimis pasirašymo algoritmais.

Remiantis prototipo aprašymu ir apibendrinimu galima suformuoti išvadą, kad:

1. Darbe siūlomas prieigos valdymo metodas gali būti praktiškai realizuotas mikropaslaugų architektūra paremtoje infrastruktūroje;
2. Prototipas užtikrina prieigos valdymo saugą tarp visų sluoksnių ir juose esančių servisų arba įrenginių;
3. OAuth 2.0 protokolas ir mTLS metodas suteikia galimybę užtikrinti prieigos valdymą tarp kliento ir API šliuzo bei išorinio ir vidinio mikropaslaugų architektūros sluoksnių;
4. JWT žetonai leidžia supaprastinti prieigos valdymo realizavimą taupant galinių įrenginių resursus.

4. Prieigos valdymo metodo prototipo tyrimas

Realizuotas prieigos valdymo metodas mikropaslaugų architektūroje tiriamas pasinaudojant testais ir programine įranga, skirta analizuoti resursų naudojimą. Tiriant prototipą siekiama išsiaiškinti, ar pasirinkti prieigos valdymo būdai veikia tinkamai ir yra efektyvūs. Taip pat tyrimu norima įvertinti siūlomo prieigos valdymo metodo resursų naudojimą (procesoriaus, operatyviosios atminties) ir greitaveiką. Resursų naudojimas ir greitaveika bus lyginama su identiškos infrastruktūros prototipu be siūlomo prieigos valdymo metodo.

4.1. Prieigos valdymo metodo prototipo testavimas

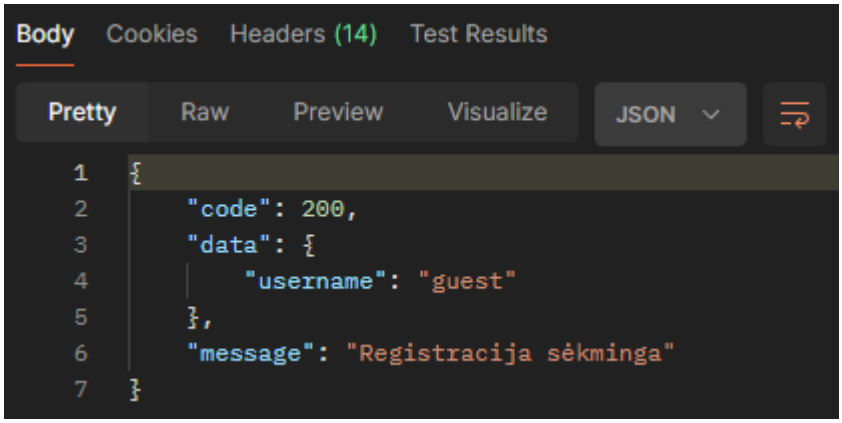
Prieigos valdymo metodo tyrimo testai buvo atliekami rankiniu būdu, užklausų siuntimui naudojant „Postman“ programinę įrangą. 2 lentelėje pateikiamas testų sąrašas su atliktų tikrinimų aprašymu, laukiamais rezultatais ir testo pabaigoje gautais rezultatais.

2 lentelė Prieigos valdymo metodo prototipo testai

Testas	Tikrinimas	Laukiamas rezultatas	Rezultatas
Vartotojo registracija	API šliuzui siunčiama užklausa dėl vartotojo registracijos į /register prieigos tašką	API šliuzas grąžina atsakymą su HTTP 200 kodu ir pateikia vartotojo vardą bei pranešimą apie sėkmingą registraciją	Gautas atsakymas su HTTP kodu 200, vartotojo vardu ir pranešimu apie sėkmingą registraciją (18 pav.)
Vartotojo prisijungimas	API šliuzui siunčiama vartotojo prisijungimo užklausa į /login prieigos tašką	API šliuzas grąžina atsakymą su HTTP 200 kodu ir pateikia vartotojo vardą, rolę bei OAuth 2.0 prieigos žetoną	Gautas atsakymas su HTTP kodu 200, vartotojo vardu, role, OAuth 2.0 protokolo prieigos žetonu ir pranešimu apie sėkmingą prisijungimą (19 pav.)
Vartotojo prieigos žetono generavimas	API šliuzas sugeneruoja OAuth 2.0 protokolo prieigos žetoną ir pateikia jį vartotojui	Sugeneruotas OAuth 2.0 prieigos žetonas su prototipo servisams reikalingais duomenimis	Gautas tinkamas OAuth 2.0 protokolo prieigos žetonas su visais reikalingais duomenimis (19 pav. ir 20 pav.)
Vartotojo vaidmens priskyrimas iš duomenų bazės	Į vartotojo prieigos žetoną įterpiamas vartotojo vaidmuo	Sugeneruotame OAuth 2.0 prieigos žetone įterptas vartotojo vaidmuo	OAuth 2.0 protokolo prieigos žetone įterptas teisingas vartotojo vaidmuo (20 pav.)
Vartotojo vaidmens leidimų tikrinimas servise	API šliuzui siunčiama užklausa dėl duomenų gavimo iš serviso. Prieš pateikiant atsakymą į užklausą, servisas patikrina, ar vartotojo vaidmuo gali gauti norimus duomenis pagal servise nustatytas prieigos valdymo taisykles	Siunčiant užklausą servisui dėl duomenų gavimo su role, kuri neturi teisės gauti duomenis, vartotojui turi būti grąžinamas atsakymas su HTTP 403 kodu ir pranešimu apie nesuteiktą prieigą	Siunčiant užklausą vartotojo vardu, kurio vaidmuo neturi prieigos teisių prie reikiamo serviso, gaunamas atsakymas su HTTP 403 kodu ir pranešimu, kad prieiga yra nesuteikta (21 pav.)

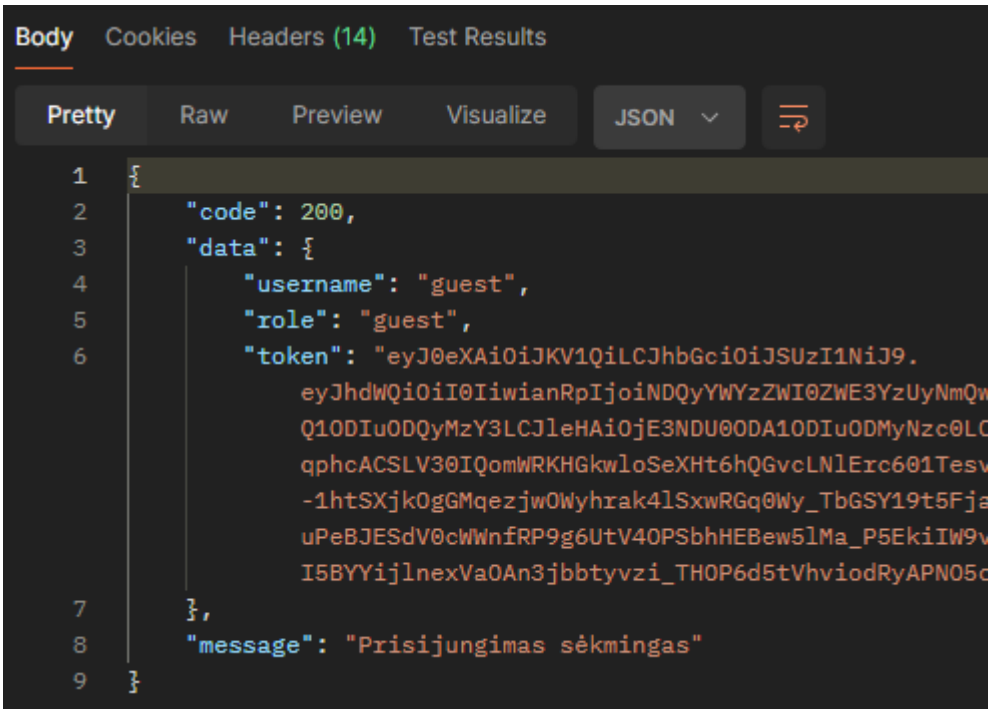
Duomenų iš vieno serviso gavimas su HMAC algoritmo žetonu	Iš ūko sluoksnio serviso siunčiama užklausa galiniam įrenginiui, kuris dėl savo resursų apribojimų naudoja HMAC algoritmą žetono tikrinimui	Ūko sluoksnio servisas, siųsdamas užklausą, žetoną pasirašo HMAC algoritmu, kurio parašą galinis įrenginys gali patikrinti naudodamas maišos funkciją ir slaptą žymą	Gautas atsakymas į išsiųstą užklausą, žetono algoritmas ir transakcijos duomenys atspausdinti galinio įrenginio žurnale (22 pav.)
Duomenų iš vieno serviso gavimas su ECDSA algoritmo žetonu	Iš ūko sluoksnio serviso siunčiama užklausa galiniam įrenginiui, kuris žetono tikrinimui naudoja ECDSA algoritmą	Ūko sluoksnio servisas, siųsdamas užklausą, žetoną pasirašo ECDSA algoritmu, kurio parašą galinis įrenginys gali patikrinti naudodamas viešą ūko sluoksnio serviso raktą	Gautas atsakymas į išsiųstą užklausą, žetono algoritmas ir transakcijos duomenys atspausdinti galinio įrenginio žurnale (23 pav.)
Duomenų iš skirtingų servisų gavimas su HMAC ir ECDSA algoritmo žetonais	API šliuzui siunčiama užklausa, į kurią turi būti atsakoma duomenimis iš skirtingų galinių įrenginių. API šliuzas kreipsis į du skirtingus ūko sluoksnio servisus, kurie toliau siųs užklausas į galinius įrenginius, naudojančius HMAC ir ECDSA algoritmų žetonus	API šliuzas grąžina atsakymą su HTTP 200 kodu ir pateikia sujungtus duomenis iš skirtingų galinių įrenginių	Gautas apjungtas užklausos atsakymas iš dviejų galinių įrenginių su skirtingais žetono algoritmais (24 pav., 25 pav. ir 26 pav.)
Komunikacijos kanalo sauga su mTLS metodu tarp kliento ir API šliuzo	API šliuzui siunčiama užklausa be kliento SSL sertifikato	Vartotojas informuojamas, kad komunikacijai reikalingas kliento SSL sertifikatas	Gautas „Nginx“ saityno serverio klaidos pranešimas apie nepateiktą kliento SSL sertifikatą (27 pav.)
Komunikacijos kanalo sauga su mTLS metodu tarp API šliuzo ir ūko sluoksnio servisų	API šliuzas siunčia užklausą ūko sluoksnio servisui be kliento SSL sertifikato	API šliuzas negali įvykdyti užklausos, nes nepateikia kliento SSL sertifikato	Gautas „cURL“ bibliotekos klaidos pranešimas apie trūkstamą SSL sertifikatą (28 pav.)

Vartotojo registracijos testui siunčiama užklausa į API šliuzo registracijos prieigos tašką ***register***. Užklausos parametrus sudaro vartotojo vardas, elektroninio pašto adresas ir slaptažodis. 18 paveiksle demonstruojamas atsakymas į registracijos užklausą. Jame nurodomas HTTP kodas (200), vartotojo vardas (guest) ir pranešimas apie sėkmingą registraciją.



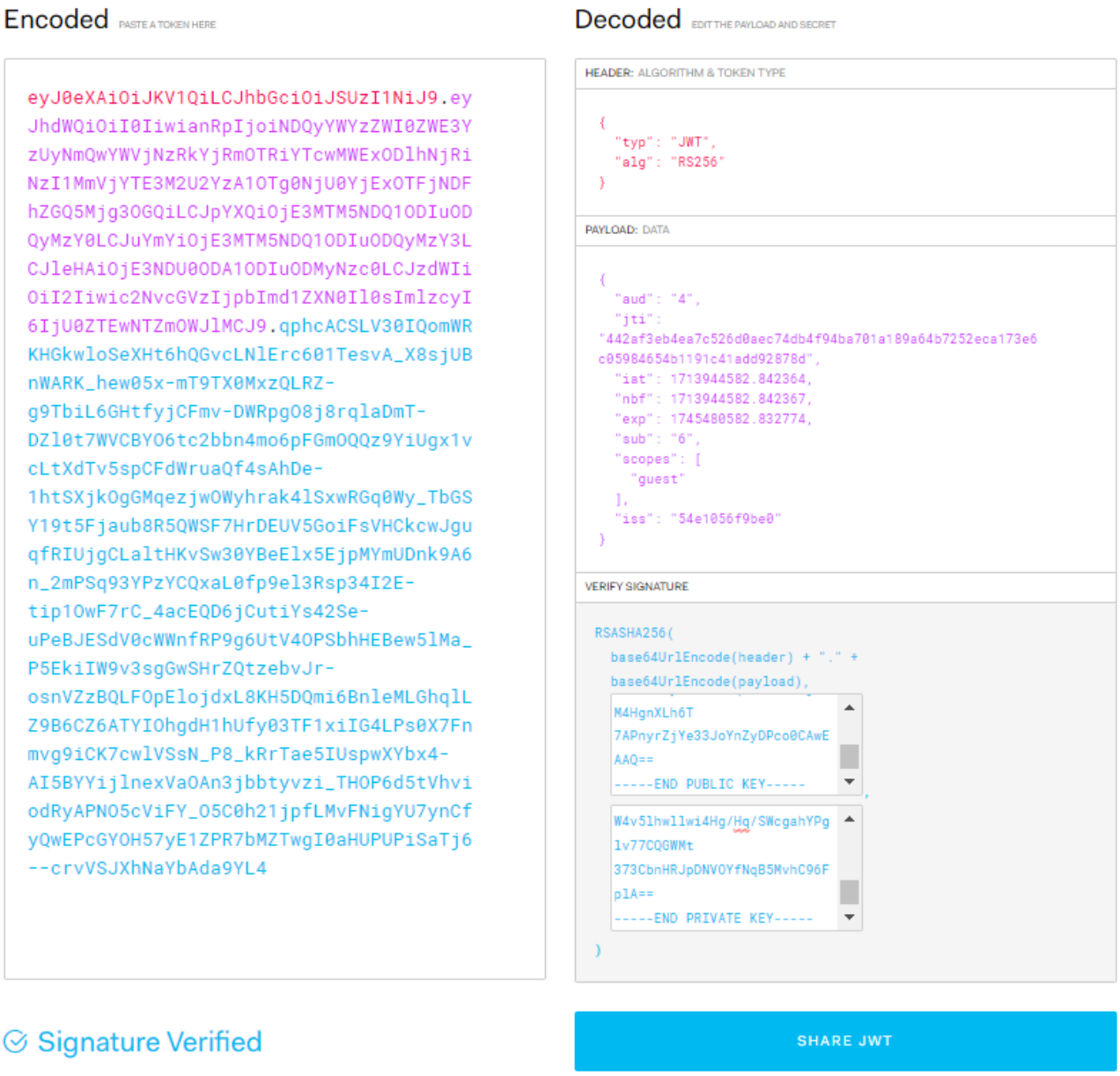
18 pav. Vartotojo registracijos testo užklausos atsakymas (šaltinis: sukurta autoriaus)

Toliau atliekamas vartotojo prisijungimo testas užklausą siunčiant API šliuzo prisijungimo prieigos taškui ***login***. Prisijungimo užklausos parametruose nurodomas vartotojo elektroninio pašto adresas ir slaptažodis. 19 paveiksle galima matyti atsakymą į užklausą, kuriame pateikiamas HTTP kodas (200), vartotojo vardas (guest), vartotojui priskirtas vaidmuo (guest), OAuth 2.0 protokolo prieigos žetonas ir pranešimas apie sėkmingą prisijungimą.



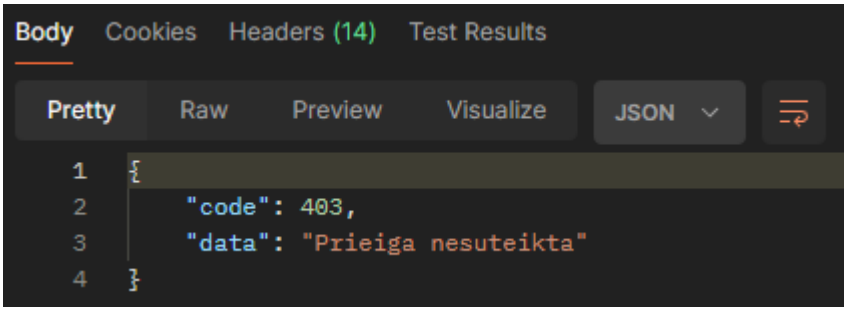
19 pav. Vartotojo prisijungimo testo užklausos atsakymas (šaltinis: sukurta autoriaus)

„Laravel“ karkaso modulis „Laravel Passport“ generuoja OAuth 2.0 protokolo prieigos žetonus su kintamaisiais, reikalingais saugumo užtikrinimui. Vidiniam prototipo sluoksniui reikalingi ***sub***, ***scopes*** ir ***iss*** kintamieji, kurie atitinkamai nurodo kliento identifikavimo numerį, roles ir, šiuo atveju, API šliuzo unikalų numerį. Kaip demonstruojama 20 paveiksle esančiame iškoduotame žetone, visi reikalingi kintamieji įterpti į prieigos žetoną ir gali būti naudojami vidiniame prototipo sluoksnyje. Taip pat galima matyti, kad žetonas pasirašytas RSA algoritmu su SHA-256 maišos funkcijos API šliuzo raktu ir pateikus raktus iškodavimo įrankiui, jis gali sėkmingai patvirtinti žetono parašą.



20 pav. Iškoduotas sugeneruotas prieigos valdymo žetonas (šaltinis: sukurta autoriaus)

Siekiant išbandyti vaidmenimis pagrįstą prieigos valdymą, įrenginių valdymo servisui buvo nustatytos taisyklės, kurios apriboja prieigą prie jo teikiamų funkciją vartotojams turintiems ***guest*** rolę. Tuomet buvo siunčiama užklausa į API šliuzo įrenginių valdymo serviso prieigos tašką ***device***, autorizacijos antraštėje įterpiant vartotojui pateiktą prieigos žetoną, kuris vaizduojamas 19 ir 20 paveiksluose. Kadangi prieiga vartotojo rolei apribota todėl, kaip matoma paveiksle 21, grąžintame atsakyme nurodomas HTTP kodas 403 ir pranešama, kad prieiga yra nesuteikta.



21 pav. Vartotojo leidimų testo užklausos atsakymas (šaltinis: sukurta autoriaus)

Visuose ūko sluoksnio servisuose saugoma informacija apie šiems servisams prieinamus galinius įrenginius ir jiems taikomus saugumo reikalavimus. Jeigu galinis įrenginys neturi pakankamai resursų arba jam nereikalingas didesnis saugumo lygis, tuomet ūko sluoksnio servisas siunčia užklausą su prieigos žetonus, kuris pasirašytas HMAC algoritmu. Galinis įrenginys gali patikrinti gautą prieigos žetoną apskaičiavęs jo maišos funkciją ir pritaikęs bendrą slaptą žymą. 22 paveiksle pateikiama ekrano iškarpa iš galinio įrenginio žurnalo. Jame galima matyti, kad įrenginys, kurio unikalus numeris yra 44086909dee0 (temperatūros jutiklis) gavo užklausą iš ūko sluoksnio serviso 9df5509a8bc3 (vidaus klimato servisas). Taip pat pateikiamas žetonas, naudotas algoritmas ir funkcija, į kurią buvo kreiptasi.

```
[2024-04-29 10:46:06]
Gavėjas: 44086909dee0
Siuntėjas: 9df5509a8bc3
Žetonas: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiI5ZGY1NTA5YThiYzMiLCJ1c2VyIj
oiMSIsInNjb3BlcyI6WyJvd25lciJdfQ.DfPVJOhgqyYWlEv6WaEz5EUuF9hdLBL5OqoVMDByBVU
Algoritmas: HMAC
Funkcija: temperature
```

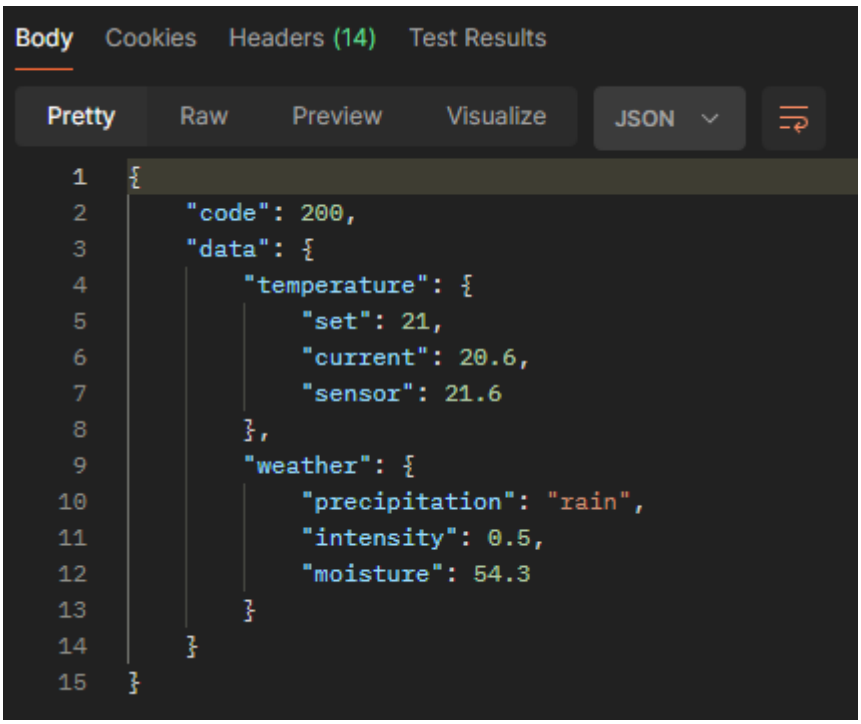
22 pav. HMAC algoritmo žetono testo žurnalo įrašas galiniame įrenginyje (šaltinis: sukurta autoriaus)

ECDSA algoritmo žetono testavimas buvo atliekamas analogiškai, kaip ir HMAC algoritmo atveju. Galinis įrenginys a1ba7bf3998b (oro kondicionierius) gavo užklausą iš ūko sluoksnio serviso 9df5509a8bc3 (vidaus klimato servisas) temperatūros funkcijai. Kaip matoma 23 paveiksle esančioje ekrano iškarpoje, užklausoje esantis prieigos žetonas buvo pasirašytas ECDSA algoritmu. Šis galinis įrenginys turi galimybę skirti daugiau resursų saugos funkcijoms todėl žetoną gali tikrinti naudodamas ūko sluoksnio serviso raktą.

```
[2024-04-29 10:47:38]
Gavėjas: a1ba7bf3998b
Siuntėjas: 9df5509a8bc3
Žetonas: eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NiJ9.eyJpc3MiOiI5ZGY1NTA5YThiYzMiLCJ1c2VyIjoiMSIsInNjb3BlcyI6WyJvd25lciJdfQ.PBn6a0YOT0wGCd4FL9qIcLjXZqo39-_vSlMuoN9lrydfUaKdhHaeq8nhrqg9FzNTWeAqE8HhixsoAT_YDv0EoA
Algoritmas: ECDSA
Funkcija: temperature
```

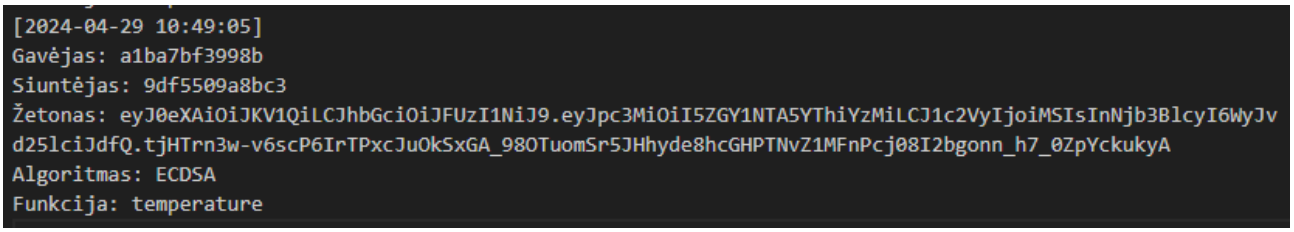
23 pav. ECDSA algoritmo žetono testo žurnalo įrašas galiniame įrenginyje (šaltinis: sukurta autoriaus)

API šliuzui numatytas funkcionalumas, leidžiantis apjungti užklausų atsakymus iš keleto galinių įrenginių. 24 paveiksle pateikiamas atsakymas į užklausą, kurios duomenis sudaro atsakymai iš skirtingų galinių įrenginių. Apjungtame atsakyme matomi vidaus temperatūros ir meteorologiniai duomenys iš vidaus ir išorės klimato servisų.



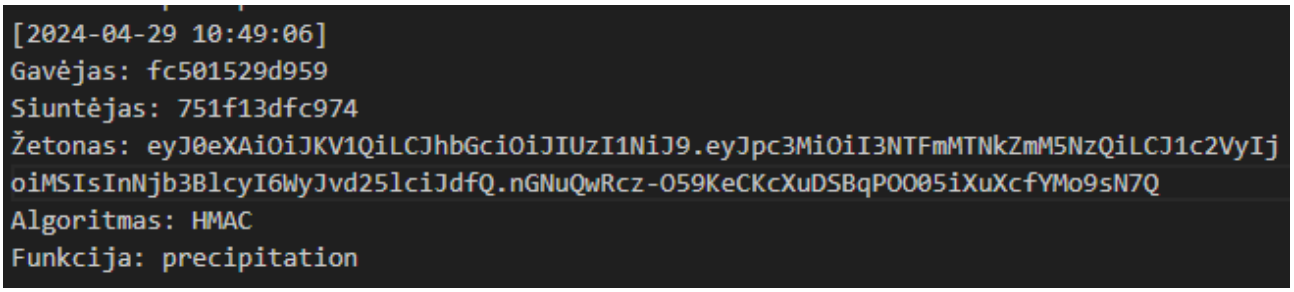
24 pav. Apjungto užklausos atsakymo testo rezultatas (šaltinis: sukurta autoriaus)

Aukščiau minėtas atsakymas buvo sudarytas siunčiant užklausas iš ūko sluoksnio servisų į keletą galinių įrenginių su skirtingais saugos algoritmais. Kaip demonstruojama 25 paveiksle, galinis įrenginys a1ba7bf3998b (oro kondicionierius) gavo užklausą iš ūko sluoksnio serviso 9df5509a8bc3 (vidaus klimato servisas). Gautas prieigos žetonas pasirašytas ECDSA algoritmu ir besikreipiantis servisas gavo prieigą prie temperatūros funkcijos.



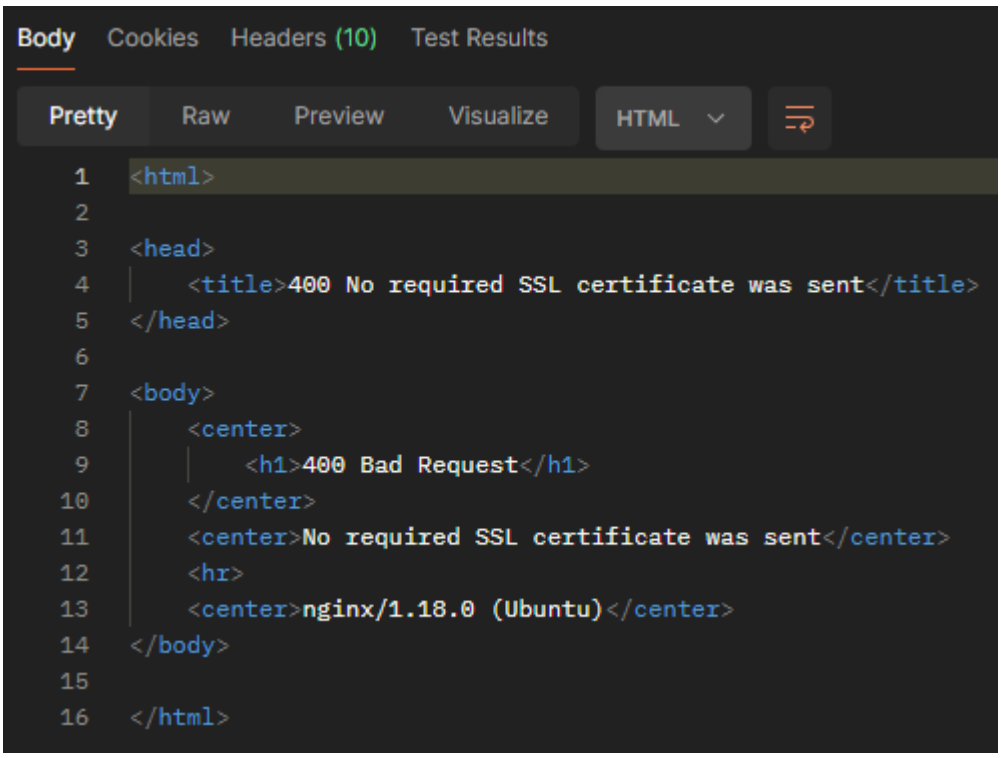
25 pav. Apjungto atsakymo testo galinio įrenginio su ECDSA algoritmu žurnalo įrašas (šaltinis: sukurta autoriaus)

26 paveiksle pateikta ekrano iškarpa su antrojo apjungto atsakymo galinio įrenginio žurnalo įrašu, kuriame matoma, kad ūko sluoksnio servisas 751f13dfc974 (išorės klimato servisas) siuntė užklausą galiniam įrenginiui fc501529d959 (kritulių jutiklis), kad būtų gauti duomenys apie kritulius. Šis įrenginys naudoja HMAC algoritmą todėl žetonas pasirašytas būtent šiuo algoritmu.



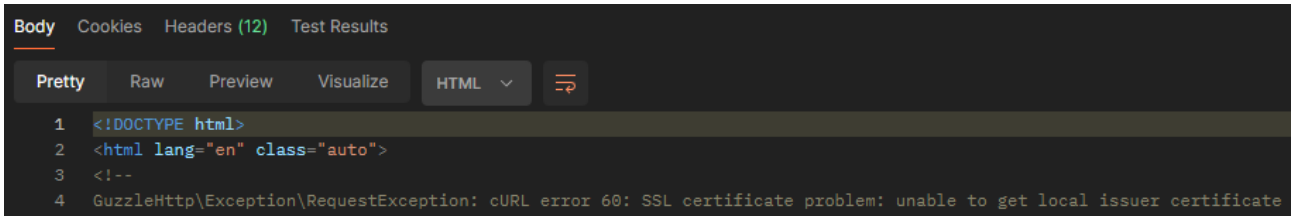
26 pav. Apjungto atsakymo testo galinio įrenginio su HMAC algoritmu žurnalo įrašas (šaltinis: sukurta autoriaus)

Komunikacijos kanalas tarp kliento ir API šliuzo bei API šliuzo ir ūko sluoksnio servisų apsaugotas mTLS prieigos valdymo metodu, kuomet abejoms komunikuojančioms pusėms reikalingi SSL sertifikatai. Šio metodo testavimui į API šliuzą buvo siunčiama užklausa nepateikus kliento SSL sertifikato. 27 paveiksle demonstruojamame gautame atsakyme galima matyti, kad saityno serveris pateikė pranešimą, jog nebuvo pateiktas reikalaujamas kliento SSL sertifikatas. Pateikus minėtą SSL sertifikatą kartu su užklausa, API šliuzas sėkmingai priima užklausas ir jas vykdo.



27 pav. mTLS metodo testo API šliuze užklausos atsakymas (šaltinis: sukurta autoriaus)

mTLS metodas tarp API šliuzo ir ūko sluoksnio servisų įgyvendinamas pasinaudojant „cURL“ biblioteka. Formuojant užklausą bibliotekos metodui nurodomi reikalingi SSL sertifikatai, kurie siunčiami ūko sluoksnio servisams. Atliekant mTLS metodo testą tarp API šliuzo ir ūko sluoksnio servisų, bibliotekos metodui SSL sertifikatai nebuvo nurodyti ir užklausa siunčiama servisams. Kaip matoma 28 paveiksle pateiktame užklausos atsakyme, biblioteka pateikia klaidos pranešimą apie trūkstamą SSL sertifikatą, kad užklausa būtų sėkmingai įvykdyta.



28 pav. mTLS metodo testo tarp API šliuzo ir ūko sluoksnio servisų užklausos atsakymas (šaltinis: sukurta autoriaus)

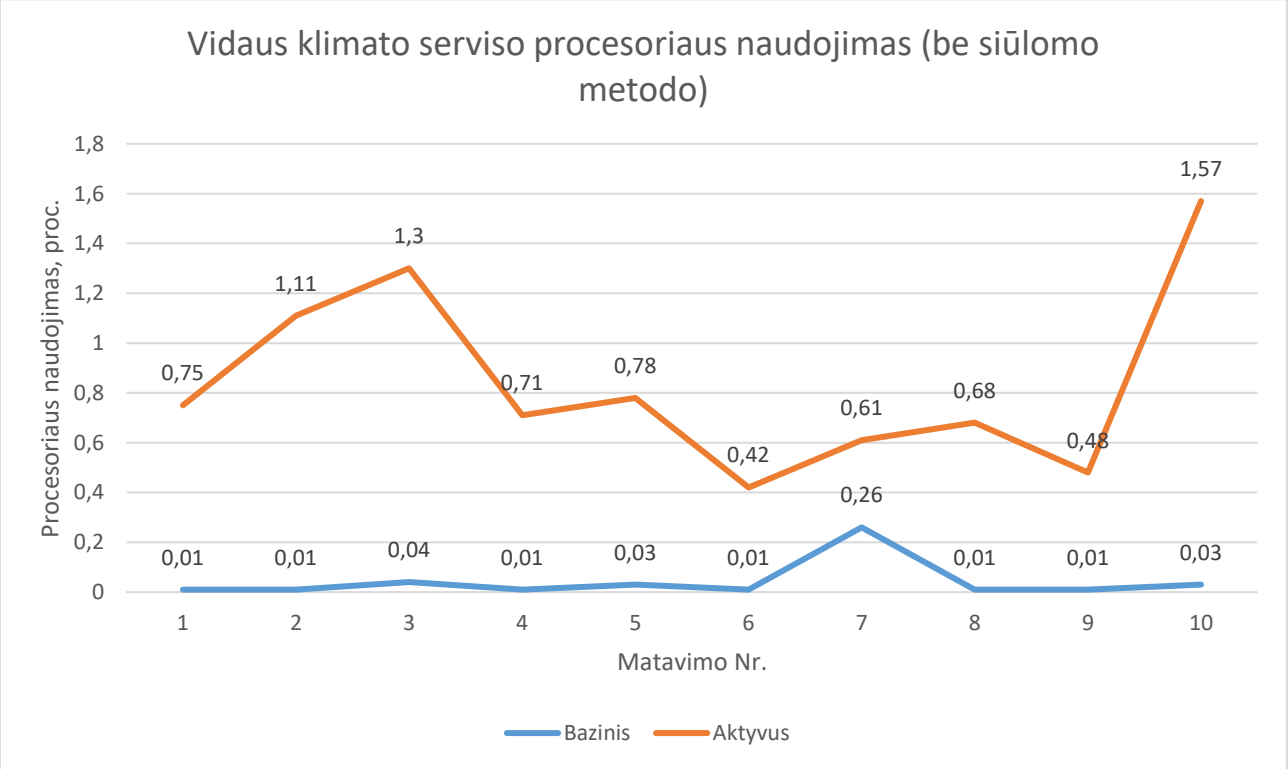
Atsižvelgiant į atlikto prieigos valdymo metodo mikropaslaugų architektūroje prototipo testavimo rezultatus galima teigti, kad projekte, aprašytame 2 skyriuje, numatyti funkcionalumai sėkmingai įgyvendinti prototipe ir veikia korektiškai.

4.2. Prieigos valdymo metodo prototipo tyrimas infrastruktūroje

Prieigos valdymo metodo mikropaslaugų architektūroje prototipo tyrimo infrastruktūroje metu buvo matuojamas procesoriaus, operatyviosios atminties naudojimas vidaus klimato serviso, oro kondicionieriaus ir temperatūros jutiklio konteineriuose. Taip pat užfiksuoti užklausų atsakymo laikai iš vidaus ir išorės klimato bei prietaisų valdymo servisų. Toliau šiame skyriuje pateikiami tyrimo rezultatai ir jų palyginimai.

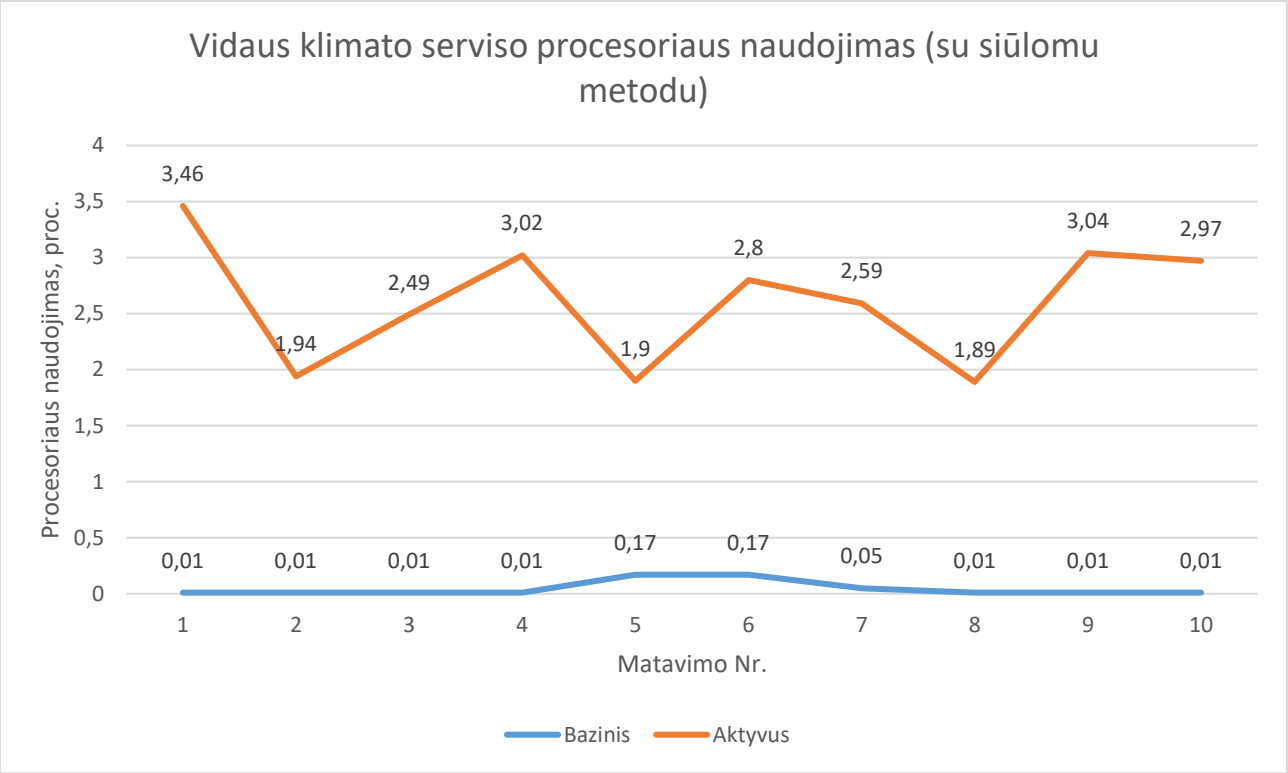
4.2.1. Prieigos valdymo metodo prototipo resursų naudojimas

Prieigos valdymo metodo prototipo resursų naudojimas buvo tiriamas siunčiant po vieną užklausą į vidaus klimato servisą maždaug kas vieną sekundę. Iš viso išsiųsta dešimt užklausų į serviso prieigos tašką ir atlikta tiek pat resursų naudojimo matavimų. Vidaus klimato servisas pasirinktas todėl, kad šis servisas kreipiasi į du galinius įrenginius, oro kondicionierių ir temperatūros jutiklį, kurie naudoja skirtingus žetonų algoritmus ir reikalauja skirtingo saugumo lygio. Tyrimas buvo vykdomas dviem scenarijais – be siūlomo prieigos valdymo metodo ir su juo. Matavimų metu buvo renkama informacija apie konteinerių procesoriaus ir operatyviosios atminties naudojimą matavimo laiko momentu. 29 paveiksle pateikiama diagrama, kurioje vaizduojamas vidaus klimato serviso be siūlomo prieigos valdymo metodo procesoriaus naudojimas.



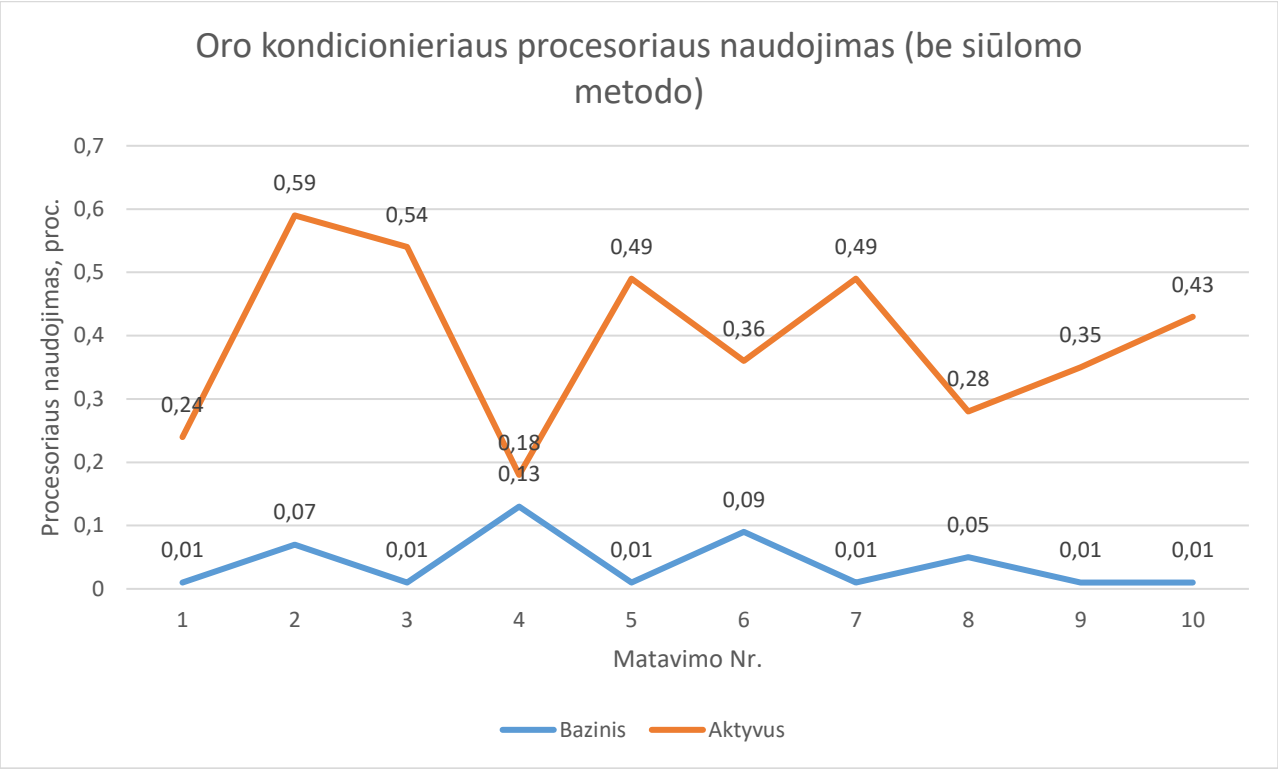
29 pav. Vidaus klimato serviso be siūlomo metodo procesoriaus naudojimo diagrama (šaltinis: sukurta autoriaus)

Bazinis procesoriaus naudojimas vaizduoja resurso sąnaudas be apkrovos arba kitaip – negaunant užklausų. Iš diagramos galima matyti, kad aktyvus procesoriaus naudojimas, kai vidaus klimato servisas aktyviai apdoroja užklausas, svyruoja aplink vieną procentą, vidutiniškai 0,84 procento. Tuo tarpu 30 paveiksle pateiktoje diagramoje galima matyti, kad vidaus klimato serviso su siūlomu prieigos valdymo metodu procesoriaus naudojimas yra šiek tiek didesnis, apie 2,5 procento (vidutiniškai – 2,61 procento).

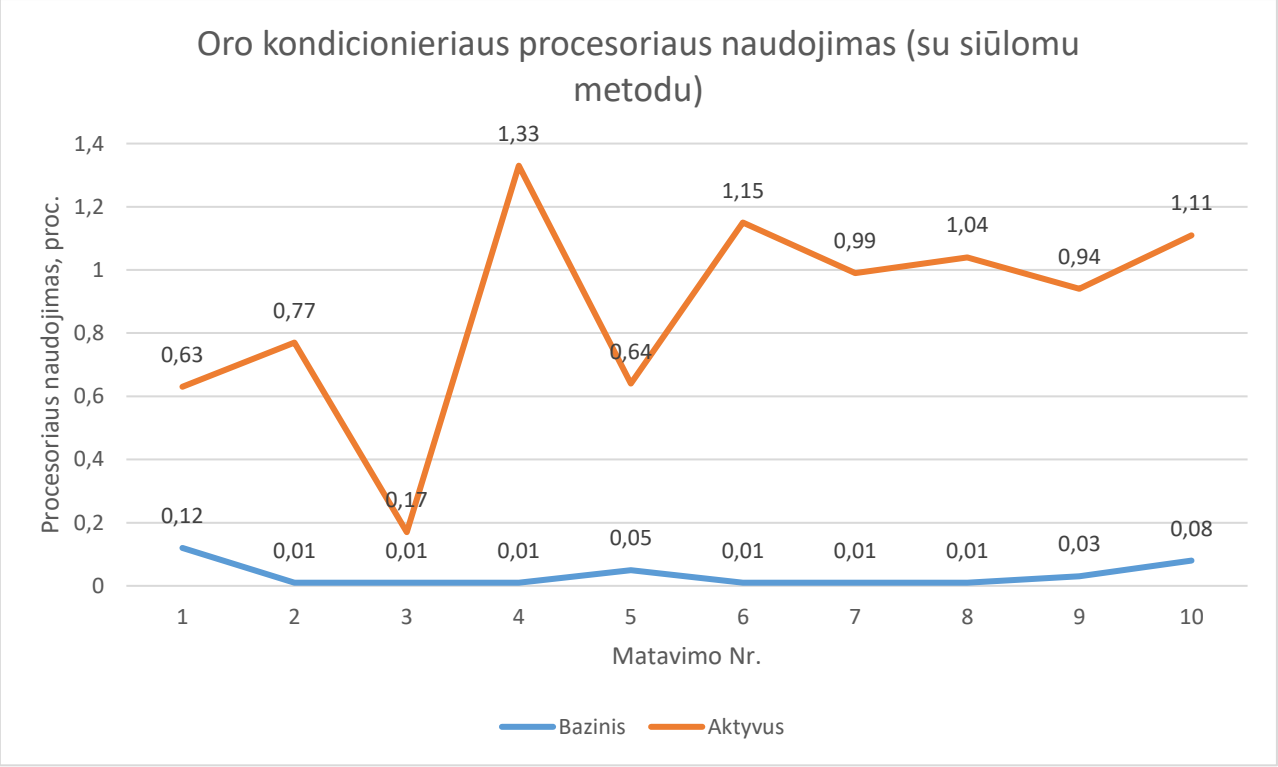


30 pav. Vidaus klimato serviso su siūlomu metodu procesoriaus naudojimo diagrama (šaltinis: sukurta autoriaus)

Oro kondicionieriaus prieigos valdymui prototipe naudojamas ECDSA algoritmo žetonas, kuris pasirašomas raktų pora. 31 paveiksle pateikiama diagrama, kurioje vaizduojamas procesoriaus naudojimas oro kondicionieriaus konteineryje, kuriame nėra siūlomo prieigos valdymo metodo. Diagramoje galima matyti, kad aktyvus procesoriaus naudojimas vidutiniškai siekia 0,40 procento. Konteineryje su darbe siūlomu prieigos valdymo metodu, kaip matoma 32 paveiksle pateiktoje diagramoje, aktyvus procesoriaus naudojimas svyruoja tarp 0,17 ir 1,33 procento (vidutiniškai – 0,88 procento). Lyginant oro kondicionieriaus konteinerio procesoriaus naudojimą tarp scenarijų, galima pastebėti, kad siūlomas metodas padidino procesoriaus resurso naudojimą maždaug dviem kartais, tačiau vidutinis procesoriaus naudojimas išlieka žemesnis nei vienas procentas.

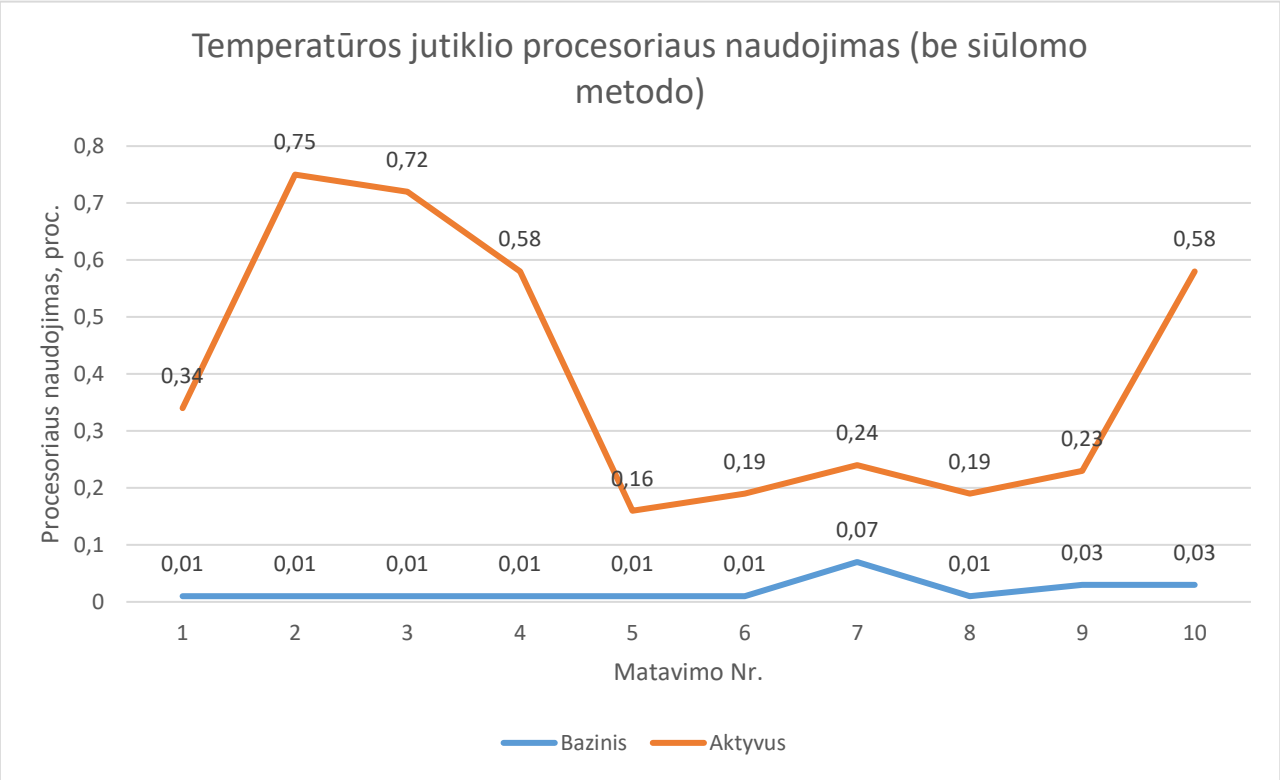


31 pav. Oro kondicionieriaus be siūlomo metodo procesoriaus naudojimo diagrama (šaltinis: sukurta autoriaus)



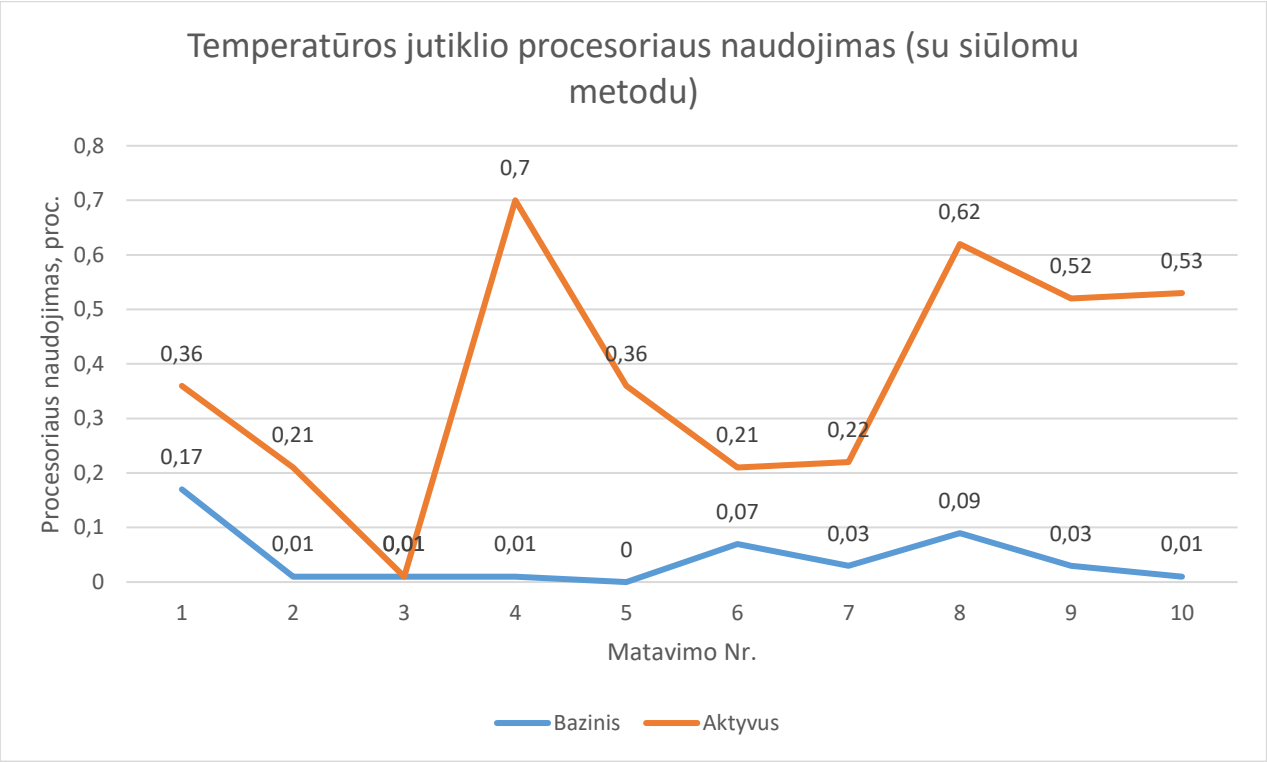
32 pav. Oro kondicionieriaus su siūlomu metodu procesoriaus naudojimo diagrama (šaltinis: sukurta autoriaus)

Prieigos valdymo metodo prototipe numatyta, kad temperatūros jutiklis yra įrenginys, turintis mažiau resursų ir reikalaujantis žemesnio saugumo lygio. Dėl šios priežasties tyrimui pasirinktas temperatūros jutiklio konteineris, kuriame prieigos valdymui naudojamas žetonas su HMAC saugos algoritmu ir slapta žyma. 33 paveiksle pateikiama diagrama su atliktų temperatūros jutiklio konteinerio procesoriaus naudojimo be siūlomo metodo matavimų rezultatais.



33 pav. Temperatūros jutiklio be siūlomo metodo procesoriaus naudojimo diagrama (šaltinis: sukurta autoriaus)

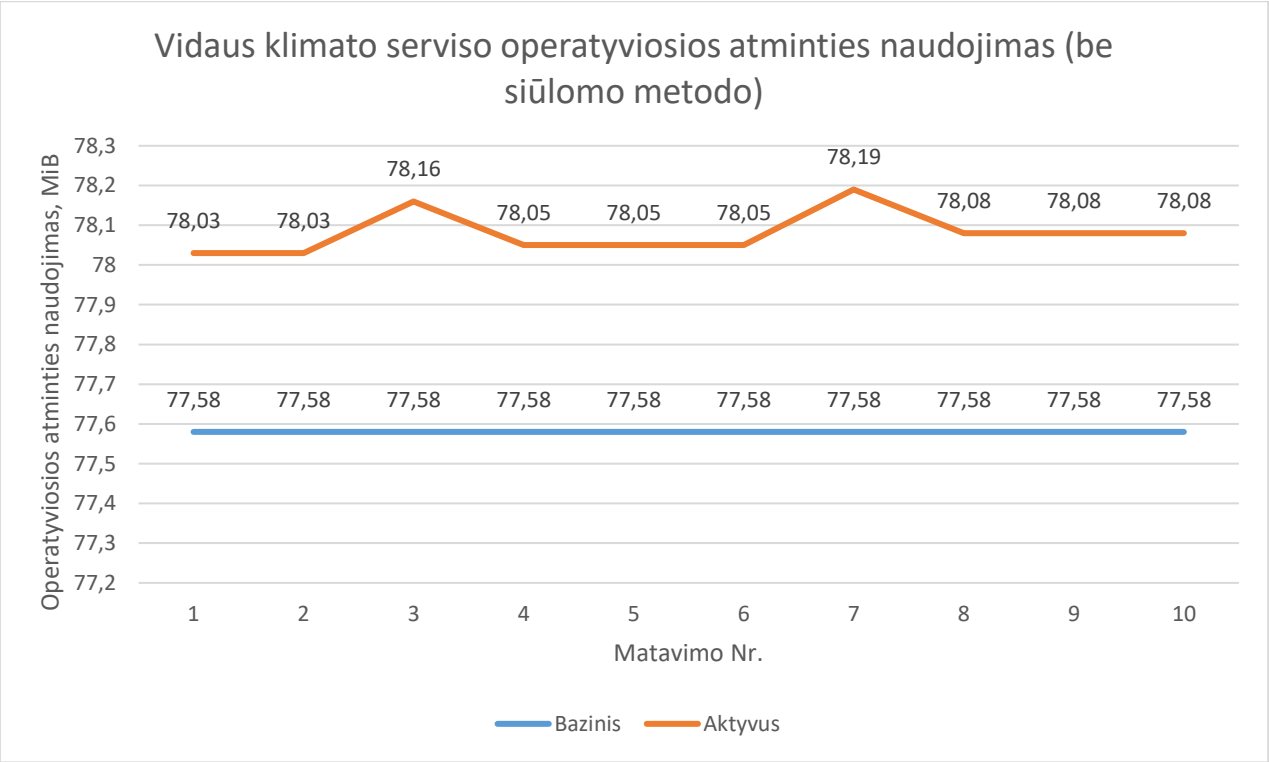
Kaip galima pastebėti iš diagramoje esančių duomenų, be darbe siūlomo prieigos valdymo metodo temperatūros jutiklio konteinerio procesoriaus naudojimas neviršija vieno procento ir svyruoja tarp 0,16 ir 0,75 procento (vidutiniškai – 0,40 procento). Palyginimui atlikti procesoriaus naudojimo matavimai su siūlomu metodu, kurių rezultatai atvaizduojami 34 paveiksle esančioje diagramoje. Iš abiejų diagramų duomenų galima pastebėti, kad lyginant du scenarijus, procesoriaus naudojimas yra labai panašus. Temperatūros jutiklio konteinerio procesoriaus naudojimas su siūlomu metodu taip pat neviršija vieno procento ir svyruoja tarp 0,01 bei 0,70 procento (vidutiniškai – 0,38 procento). Vidutinis procesoriaus naudojimas rodo, kad siūlomas metodas nenaudoja daugiau procesoriaus resursų nei įprastai, tačiau padidina įrenginio saugumo lygį įgyvendindamas prieigos valdymą žetonais su integralumo užtikrinimu.



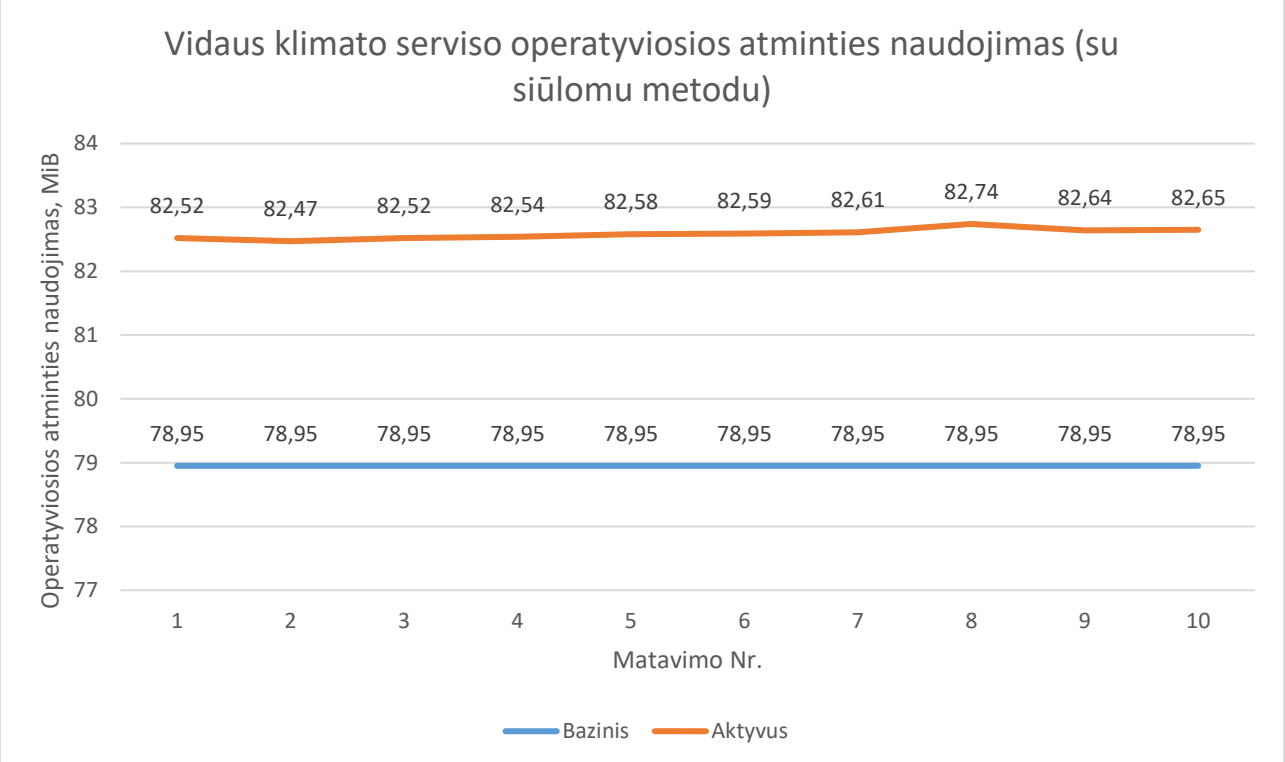
34 pav. Temperatūros jutiklio su siūlomu metodu procesoriaus naudojimo diagrama (šaltinis: sukurta autoriaus)

Įvertinus visų trijų įrenginių procesoriaus resurso naudojimo palyginimus galima daryti prielaidą, kad darbe siūlomas prieigos valdymo metodas reikšmingai nepadidinta procesoriaus naudojimo įrenginiuose, nepaisant jų resursų apribojimų. Taip pat galima pastebėti, kad lyginant ECDSA ir HMAC algoritmų naudojimą prieigos valdymo žetonuose, procesoriaus naudojimo skirtumas nėra didelis ir siekia tik 0,50 procento. Vadinasi, įrenginiams, turintiems mažiau resursų, tačiau galintiems atlikti kriptografijos operacijas, procesoriaus naudojimo požiūriu, būtų tikslinga naudoti ECDSA algoritmo žetonus, kurie turi saugos pranašumą prieš HMAC algoritmą, kadangi pasirašymui naudoja raktų porą, o ne maišos funkciją su slapta žyma.

Kartu su procesoriaus naudojimu, įrenginiuose buvo matuojamas ir operatyviosios atminties naudojimas. Matavimai atlikti baziniam operatyviosios atminties naudojimui, kai įrenginys negavo jokių užklausų ir aktyviam, kuomet įrenginiui buvo siunčiama po vieną užklausą maždaug kas vieną sekundę. Kaip ir procesoriaus naudojimo matavimų atveju, operatyvioji atmintis buvo matuojama dešimt kartų. 35 paveiksle pateikiama diagrama su vidaus klimato serviso be darbe siūlomo prieigos valdymo metodo operatyviosios atminties naudojimo matavimų rezultatais. Bazinis operatyviosios atminties naudojimas, kaip ir kitų įrenginių atvejais, siekia apie 78 megabitus, kurį sudaro resursai, skirti operacinei sistemai. Vidaus klimato serviso aktyvus operatyviosios atminties naudojimas svyruoja tarp 78,03 ir 78,16 megabito (vidutiniškai – 78,08 megabito). Skirtumas tarp vidutinio aktyvaus ir bazinio operatyviosios atminties naudojimo yra 0,50 megabito.



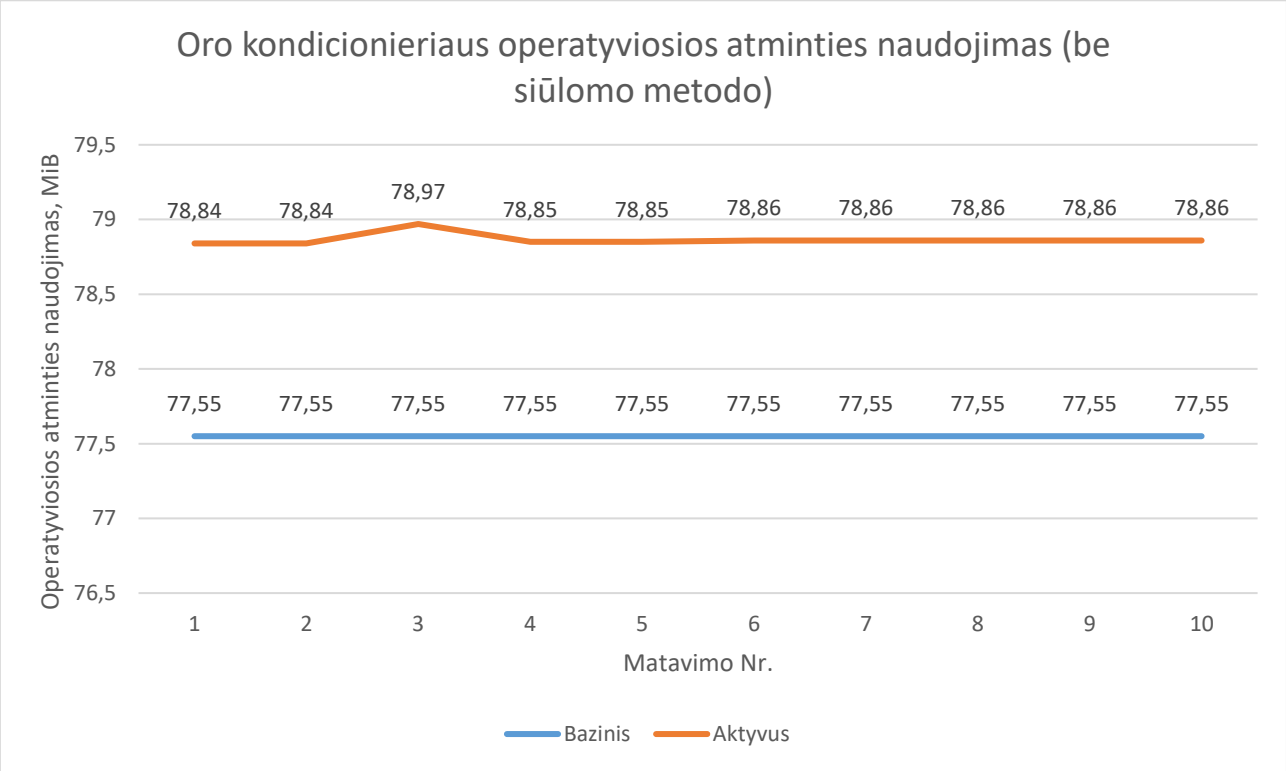
35 pav. Vidaus klimato serviso be siūlomo metodo operatyviosios atminties naudojimo diagrama (šaltinis: sukurta autoriaus)



36 pav. Vidaus klimato serviso su siūlomu metodu operatyviosios atminties naudojimo diagrama (šaltinis: sukurta autoriaus)

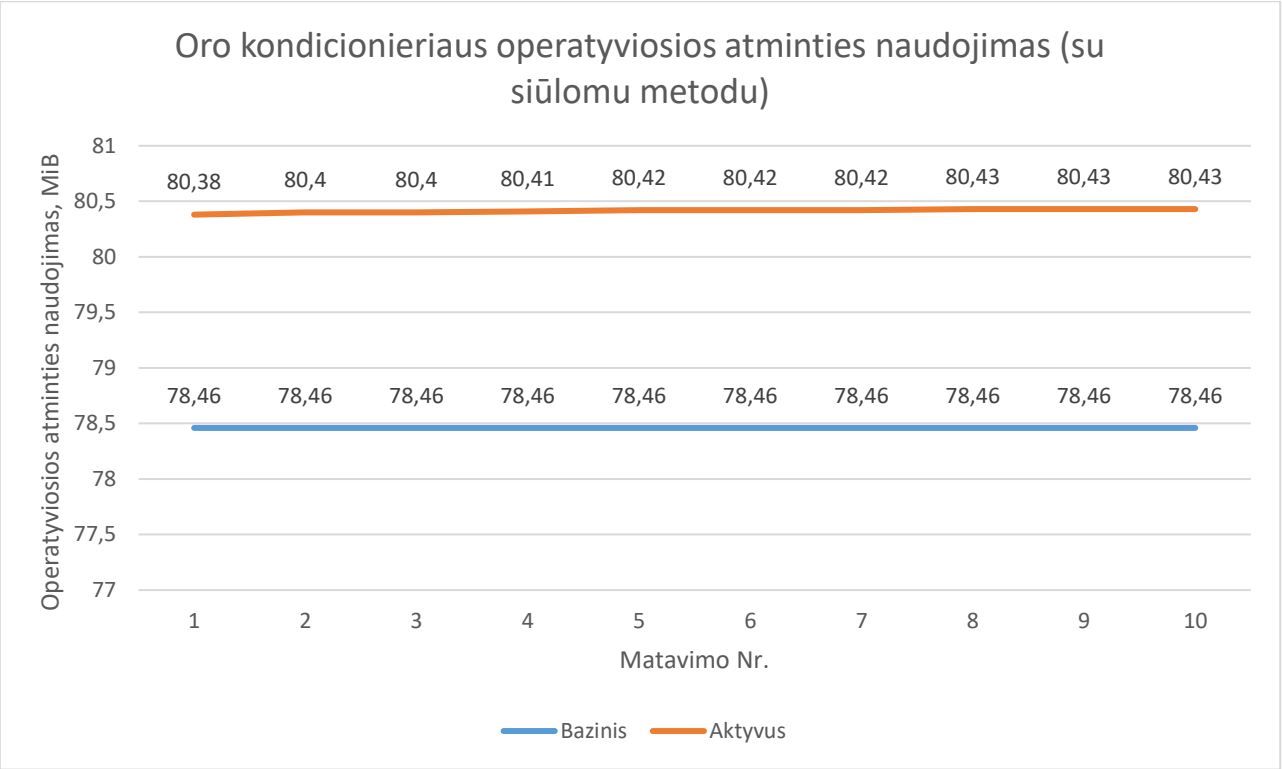
Vidaus klimato serviso, kuriame įdiegtas darbe siūlomas prieigos valdymo metodas, konteinerio operatyviosios atminties naudojimo matavimų duomenys matomi 36 paveiksle esančioje diagramoje. Antrojo scenarijaus atveju vidaus klimato serviso operatyviosios atminties naudojimas svyruoja tarp 82,47 ir 82,74 megabito (vidutiniškai – 82,59 megabito). Vidutinio aktyvaus ir bazinio operatyviosios atminties naudojimo skirtumas siekia 3,64 megabito. Lyginant abu scenarijus, vidutinis aktyvus vidaus klimato serviso operatyviosios atminties naudojimas skiriasi 4,51 megabito. Įvertinant simuliacinio įrenginio apribojimus, šis skirtumas nėra reikšmingas, kadangi numatyta, kad vidaus klimato servisas turi pakankamai resursų atlikti visoms siūlomo prieigos valdymo metodo operacijoms atlikti.

37 paveiksle pateikiama diagrama, kurioje pavaizduoti oro kondicionieriaus konteinerio be siūlomo prieigos valdymo metodo operatyviosios atminties naudojimo matavimų rezultatai. Kaip galima matyti iš pateiktų duomenų, aktyvus operatyviosios atminties naudojimas svyruoja tarp 78,84 ir 78,97 megabito (vidutiniškai – 78,87 megabito). Vidutinis aktyvus operatyviosios atminties naudojimas nuo bazinio skiriasi 1,32 megabito.



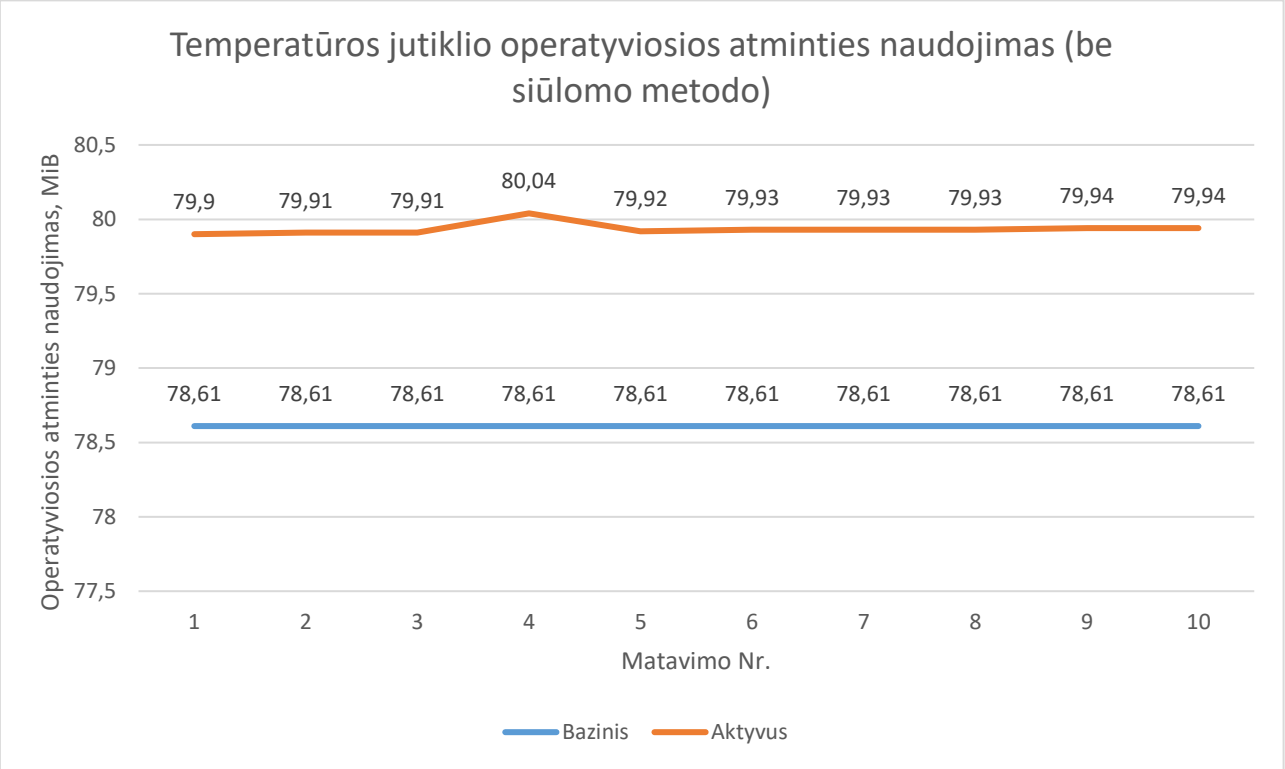
37 pav. Oro kondicionieriaus be siūlomo metodo operatyviosios atminties naudojimo diagrama (šaltinis: sukurta autoriaus)

Anksčiau darbe paminėta, kad oro kondicionierius su siūlomu prieigos valdymo metodu naudoja ECDSA algoritmo raktų pora pasirašytus prieigos žetonus. 38 paveiksle esančioje diagramoje vaizduojamas oro kondicionieriaus su siūlomu metodu operatyviosios atminties naudojimas. Galima matyti, kad aktyvus operatyviosios atminties naudojimas yra pakankamai stabilus ir kinta tarp 80,38 ir 80,43 megabito (vidutiniškai – 80,41 megabito). Vidutinis operatyviosios atminties naudojimas nuo bazinio skiriasi 1,95 megabito. Lyginant vidutinius aktyvaus operatyviosios atminties naudojimo duomenis tarp scenarijų, siūlomas metodas padidina vidutinį naudojimą 1,54 megabito. Šis padidėjimas gali būti laikomas nežymiu ir neturinčiu didelės įtakos įrenginiui.



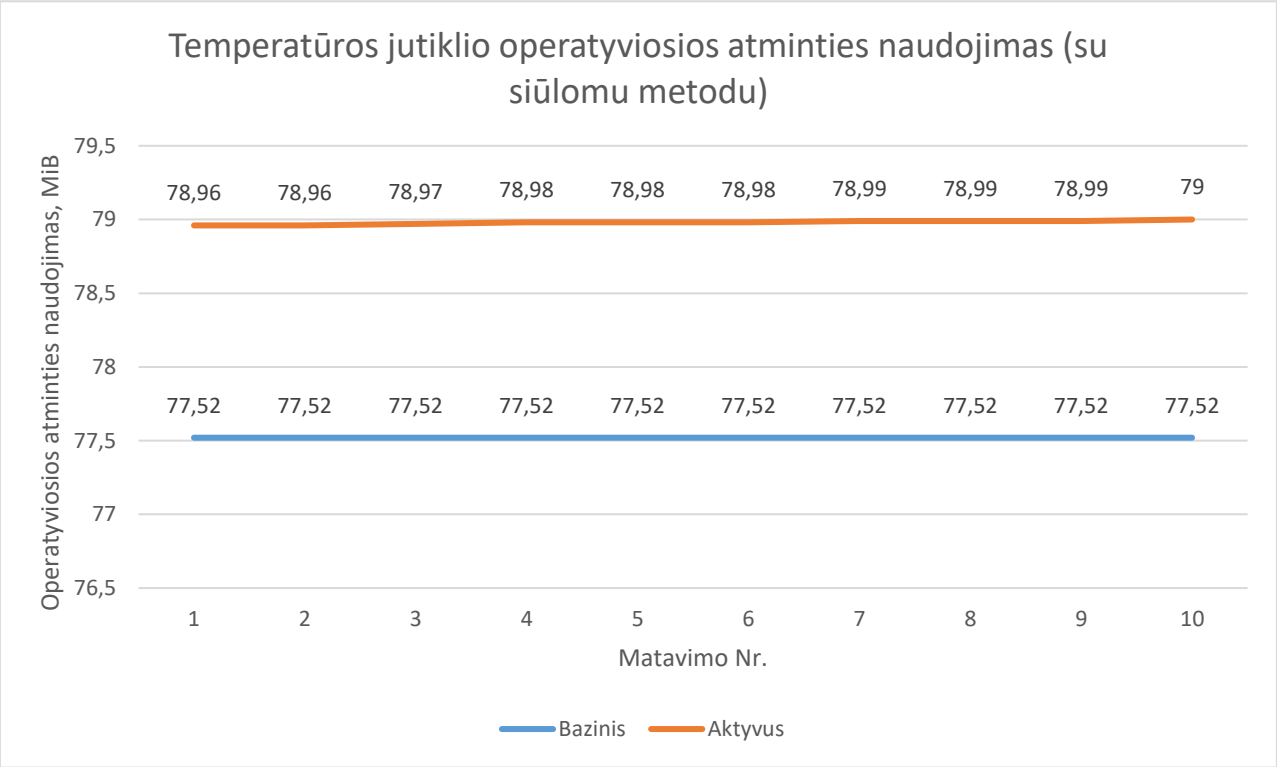
38 pav. Oro kondicionieriaus su siūlomu metodu operatyviosios atminties naudojimo diagrama (šaltinis: sukurta autoriaus)

39 paveiksle esančioje diagramoje pateikti rezultatai iš temperatūros jutiklio konteinerio be siūlomo metodo operatyviosios atminties naudojimo matavimų.



39 pav. Temperatūros jutiklio be siūlomo metodo operatyviosios atminties naudojimo diagrama (šaltinis: sukurta autoriaus)

Galima matyti, kad temperatūros jutiklio konteinerio be siūlomo metodo aktyvus operatyviosios atminties naudojimas yra tarp 79,90 ir 80,04 megabito (vidutiniškai – 79,94 megabito). Skirtumas tarp bazinio ir vidutinio aktyvaus operatyviosios atminties naudojimo siekia 1,33 megabito. Tuo tarpu temperatūros jutiklis, kuriame įdiegtas darbe siūlomas prieigos valdymo metodas, kaip matoma 40 paveiksle esančioje diagramoje, sunaudoja tarp 78,96 ir 79 megabitų operatyviosios atminties (vidutiniškai – 78,98 megabito). Vidutinio aktyvaus ir bazinio operatyviosios atminties naudojimo temperatūros jutiklyje su siūlomu metodu skirtumas yra 1,46 megabito.



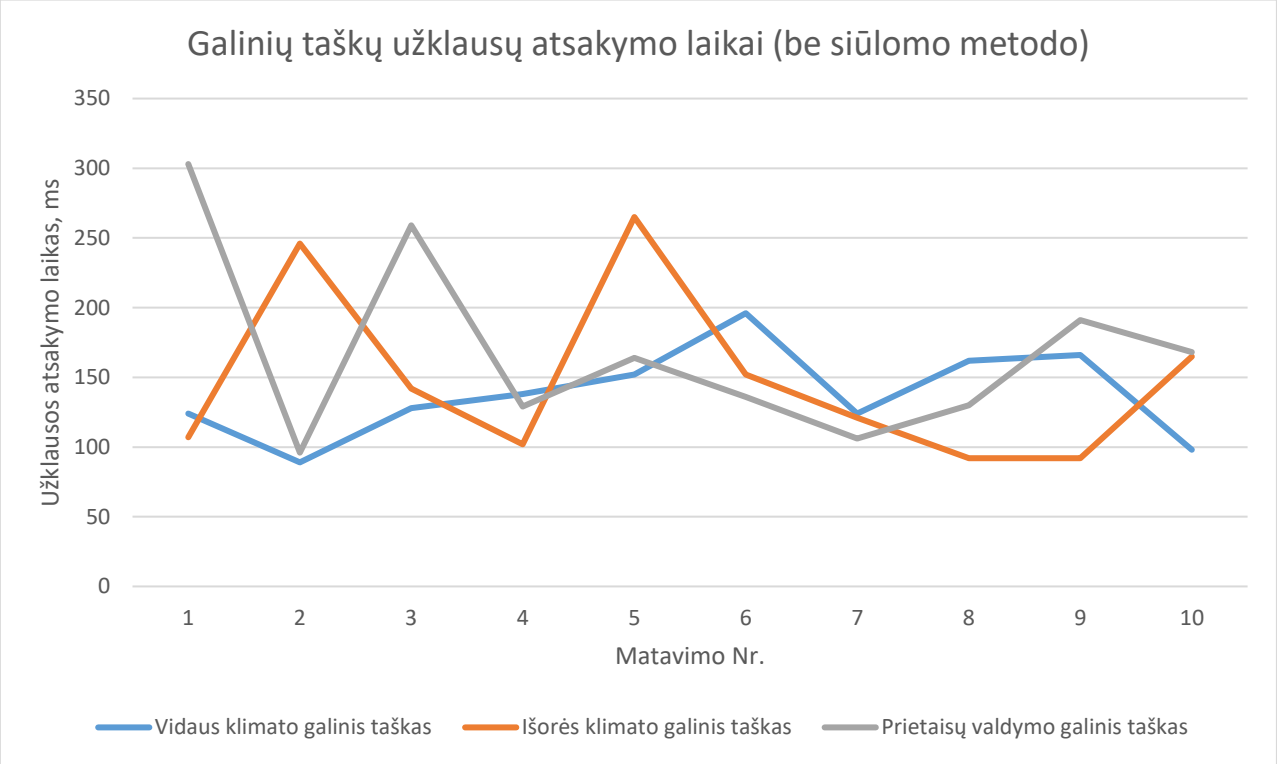
40 pav. Temperatūros jutiklio su siūlomu metodu operatyviosios atminties naudojimo diagrama (šaltinis: sukurta autoriaus)

Lyginant vidaus klimato serviso, oro kondicionieriaus ir temperatūros jutiklio operatyviosios atminties naudojimą tarp scenarijų, galima pastebėti, kad operatyviosios atminties suvartojimo padidėjimas įrenginiuose, kuriuose įdiegtas siūlomas prieigos valdymo metodas, neviršija 2 megabitų. Vertinant naudas, kurias siūlomas metodas suteikia saugumo lygiui, tokio dydžio operatyviosios atminties suvartojimas gali būti toleruojamas. Taip pat, palyginus skirtingų saugos algoritmų (ECDSA ir HMAC) operatyviosios atminties naudojimą atitinkamai tarp oro kondicionieriaus ir temperatūros jutiklio, vidutinio aktyvaus operatyviosios atminties naudojimo skirtumas yra 0,49 megabito. Įvertinus skirtumą, galima daryti prielaidą, kad esant techninėms galimybėms įrenginiuose, vietoje HMAC saugos algoritmo turėtų būti naudojamas ECDSA.

4.2.2. Prieigos valdymo metodo prototipo greitaveika

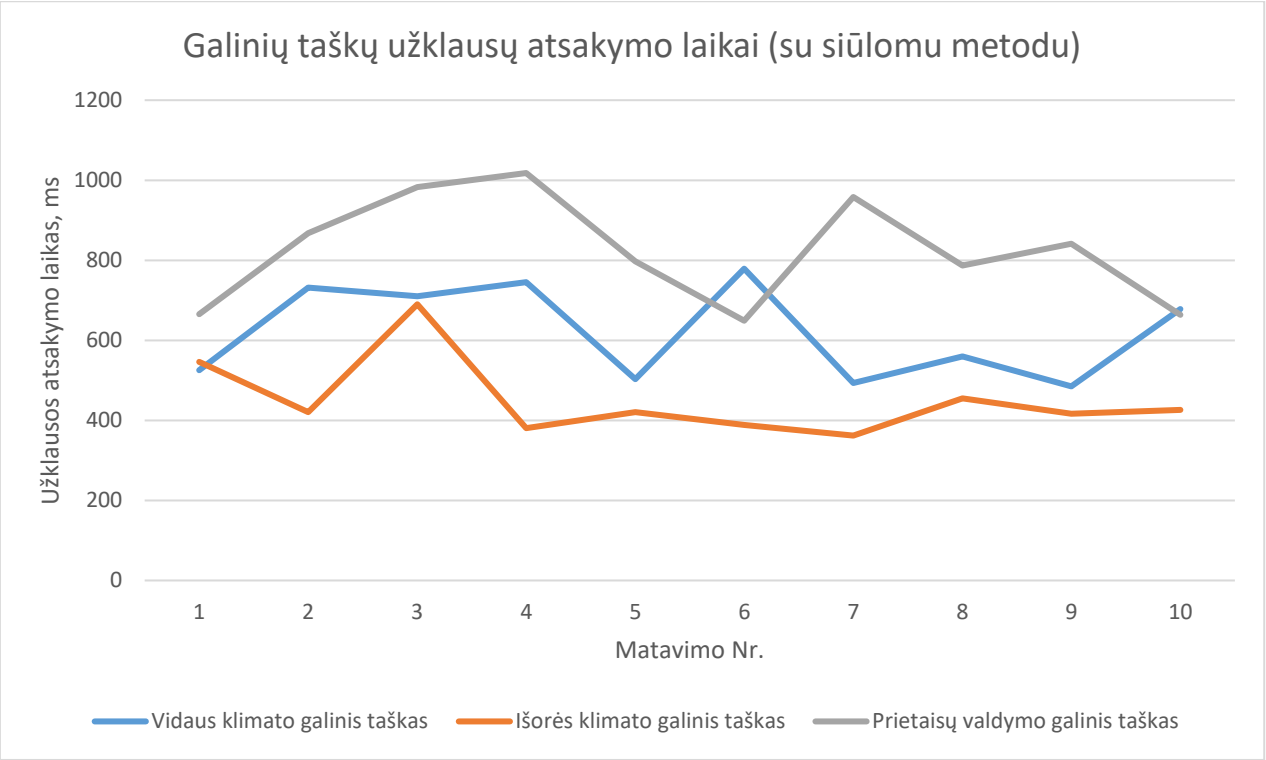
Tyrimo metu taip pat įvertinta prieigos valdymo metodo prototipo greitaveika infrastruktūroje. Matavimai atlikti siunčiant užklausas API šliuzui į tris galinius taškus: vidaus klimato, išorės klimato ir prietaisų valdymo. Iš viso išsiųsta trisdešimt užklausų, po dešimt kiekvienam galiniam taškui. Matavimai atlikti dviem scenarijais – be siūlomo metodo ir su darbe siūlomu metodu.

41 paveiksle pateikiamoje diagramoje matomi rezultatai iš trijų galinių taškų. Šie duomenys gauti atlikus matavimus infrastruktūroje, kurios įrenginiuose nėra naudojamas darbe siūlomas prieigos valdymo metodas. Vidutiniškai greičiausiai atsakymą į užklausą pateikė vidaus klimato galinis taškas (138 milisekundės), tuomet išorės klimato galinis taškas (148 milisekundės) ir prietaisų valdymo galinis taškas (168 milisekundės).



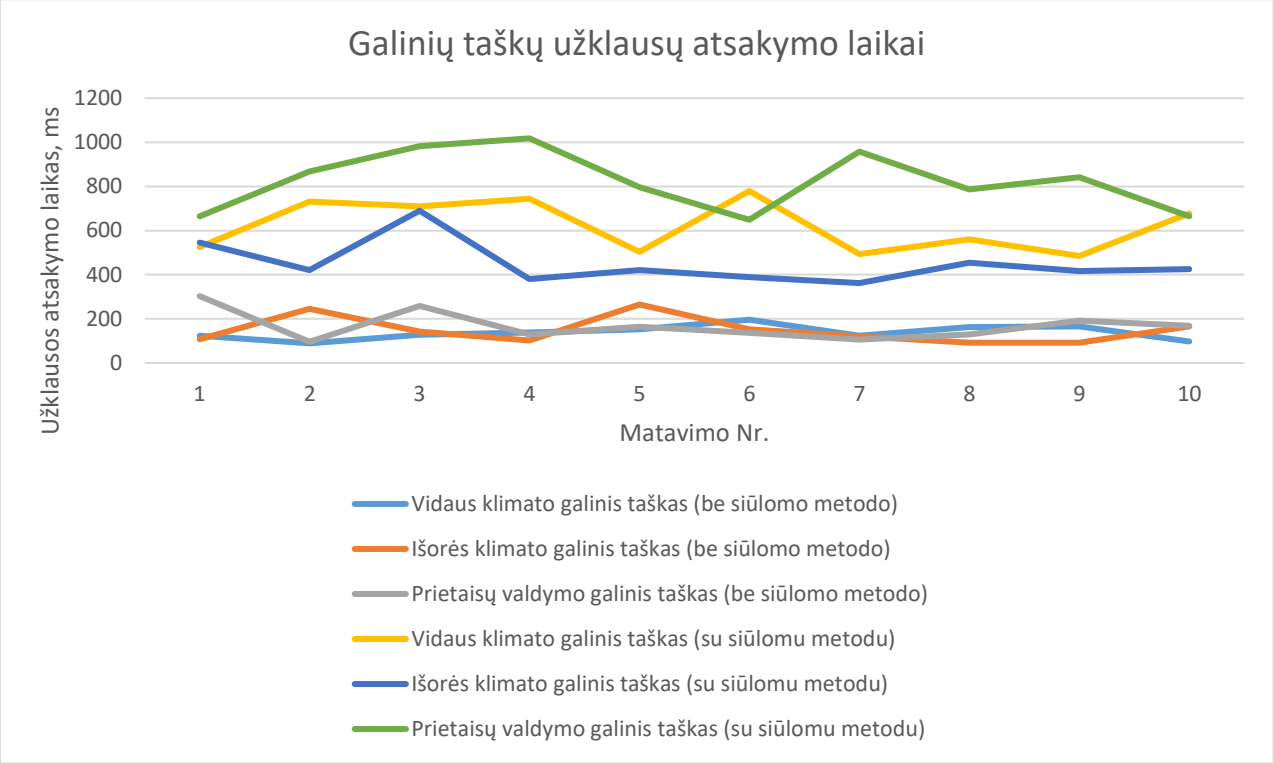
41 pav. Galinių taškų be siūlomo metodo atsakymo laikų diagrama (šaltinis: sukurta autoriaus)

Atliekant matavimus infrastruktūroje, kurios įrenginiuose naudojamas darbe siūlomas prieigos valdymo metodas, gauti skirtingi duomenys. Kaip matoma 42 paveiksle esančioje diagramoje, įdiegus siūlomą metodą, atsakymo į užklausas laikai ženkliai išaugo. Vidaus klimato galinio taško vidutinis atsakymo laikas padidėjo iki 621 milisekundės, išorės klimato galinio taško – 451 milisekundė ir prietaisų valdymo galinio taško – 823 milisekundės. Visų galinių taškų atsakymo laikas padidėjo daugiau kaip tris kartus. Abiejų scenarijų įrenginių konteineriai yra identiški, skiriasi tik jų programinis kodas. Šis ženklus atsakymo į užklausas laikų padidėjimas galėjo būti nulemtas prototipo programinio kodo ir vykdomų operacijų, kurios turėjo būti apdorojamos ribotų resursų aplinkoje.



42 pav. Galinių taškų su siūlomu metodu atsakymo laikų diagrama (šaltinis: sukurta autoriaus)

Apibendrinant greitaveikos tyrimą ir įvertinant duomenis, vaizduojamus 43 paveiksle esančioje diagramoje, galima teigti, kad siūlomas prieigos valdymo metodas prailgina užklausų atsakymo laiką mažiausiai trimis kartais. Greitaveikos sumažėjimui įtakos gali turėti prototipe naudojami metodai prieigos valdymui ir saugumo užtikrinimui.



43 pav. Galinių taškų atsakymo laikų palyginimo diagrama (šaltinis: sukurta autoriaus)

4.3. Prieigos valdymo metodo prototipo tyrimo apibendrinimas ir išvados

Darbe siūlomas prieigos valdymo metodas mikropaslaugų architektūroje buvo ištestuotas ir ištirtas infrastruktūroje, kuri yra detalizuojama 3 skyriuje. Atlikus prototipo komponentų ir funkcionalumų testavimą paaiškėjo, kad 2 skyriuje aprašytas prototipas įgyvendintas sėkmingai ir veikia korektiškai. Taip pat atliktas tyrimas, kuris leido palyginti siūlomo metodo prototipą su identiška infrastruktūra be metodo ir įvertinti pasirinktų metodų tinkamumą, efektyvumą, resursų suvartojimą bei greitaveiką.

Testavimo ir tyrimo duomenys leidžia teigti, kad:

1. Prototipas įgyvendintas pagal 2 skyriuje numatytus reikalavimus ir veikia korektiškai;
2. Siūlomo metodo procesoriaus naudojimo padidėjimas, lyginant bazinius matavimus, yra nežymus ir metodas gali būti naudojamas įrenginiuose su ribotais resursais;
3. Siūlomo metodo operatyviosios atminties naudojimo padidėjimas nėra ženklus, o tai suteikia galimybę metodą naudoti ribotų resursų įrenginiuose;
4. Prieigos valdymo metodo su ECDSA algoritmo žetonais resursų suvartojimas nežymiai skiriasi nuo HMAC saugos algoritmo žetonų todėl tikslinga pasirinkti ECDSA algoritmą, jeigu įrenginiuose yra galimybė naudoti kriptografines operacijas;
5. Siūlomas prieigos valdymo metodas ženkliai padidina užklausų atsakymo laiką, tačiau metodo optimizavimas gali padėti sumažinti įtaką greitaveikai.

Išvados

1. Analizuojant literatūros šaltinius pastebėta, kad vienos iš pagrindinių mikropaslaugų architektūros problemų yra prieigos valdymas servisuose, komunikacijos kanalo saugos tarp skirtingų servisų užtikrinimas, būtinybė vertinti ir projektuoti saugą kiekvienam servisui individualiai bei riboti resursai galiniuose įrenginiuose;
2. Esamų prieigos valdymo metodų analizė parodė, kad mikropaslaugų architektūroje yra naudojami plačiai paplitę, įvairioms architektūroms ir sistemoms taikomi prieigos valdymo metodai. Išnagrinėjus metodų taikymą mikropaslaugų architektūroje, galima daryti išvadą, kad saugaus prieigos valdymo įgyvendinimas gali būti pasiektas jau turimomis priemonėmis;
3. Atlikus esamų prieigos valdymų metodų mikropaslaugų architektūroje analizę ir palyginimą pastebėta, kad norint pasiekti didesnį saugumo lygį ir išplėsti sistemos funkcines galimybes, tikslinga naudoti keletą skirtingų prieigos valdymo metodų. Toks pasirinkimas leidžia apjungti stipriąsias metodų savybes ir išspręsti problemas, kurios kyla metodus naudojant atskirai;
4. Darbe siūlomas metodas, naudojant OAuth 2.0 protokolą, mTLS metodą ir JWT žetonus, sprendžia prieigos valdymo problemą mikropaslaugų architektūra paremtos daiktų interneto infrastruktūros įrenginiuose, kuriuose yra riboti resursai, tokie kaip procesorius, operatyvioji atmintis;
5. Ištyrus realizuotą prieigos valdymo metodo mikropaslaugų architektūroje prototipą nustatyta, kad pasiūlyto metodo resursų naudojimo padidėjimas nėra didelis, lyginant su infrastruktūra be metodo. Taip pat nustatyta, kad siūlomas metodas sumažina greitaveiką, tačiau keliama hipotezė, kad optimizavus programinį kodą ir naudojamas bibliotekas galima pasiekti didesnę greitaveiką. Apibendrinant galima daryti išvadą, kad siūlomas metodas sprendžia darbe keliamą problemą.

Literatūros sąrašas

1. VENČKAUSKAS, A. - MORKEVIČIUS, N. - JUKAVIČIUS, V. - DAMAŠEVIČIUS, R. - TOLDINAS, J. - GRIGALIŪNAS, Š. An Edge-Fog Secure Self-Authenticable Data Transfer Protocol. *Sensors* [interaktyvus]. 2019. Vol. 19, no. 16, p. 3612. [žiūrėta 2023-01-02]. Prieiga per: doi: <https://doi.org/10.3390/s19163612>
2. BRACHMANN, M. - GARCIA-MORCHON, O. - KIRSCHE, M. Security for Practical CoAP Applications: Issues and Solution Approaches [interaktyvus]. 2015. [žiūrėta 2023-01-02]. Prieiga per: <https://www.researchgate.net/publication/265973615_Security_for_Practical_CoAP_Applications_Issues_and_Solution_Approaches>
3. DE ALMEIDA, M.G. - CANEDO, E.D. Authentication and authorization in microservices architecture: A systematic literature review. *Applied Sciences* [interaktyvus]. 2022. Vol 12, no 6, p. 3023. [žiūrėta 2023-01-02]. Prieiga per: doi: <https://doi.org/10.3390/app12063023>
4. VIGGIATO, M. - TERRA, R. - ROCHA, H. - VALENTE, M.T. - FIGUEIREDO, E. Microservices in Practice: A Survey Study. [interaktyvus]. 2018. [žiūrėta 2023-01-02]. Prieiga per: doi: <https://doi.org/10.48550/arXiv.1808.04836>
5. RICHARDSON, C. In *What are microservices?* [interaktyvus]. [žiūrėta 2023-01-02]. Prieiga per internetą: <https://microservices.io/>
6. RICHARDSON, C. *Microservices patterns: With examples in Java* [interaktyvus]. Simon and Schuster, 2018. [žiūrėta 2023-01-02]. Prieiga per internetą: < https://books.google.lt/books?id=QTgzEAAAQBAJ&vq=microservices&dq=microservices&lr=&hl=lt>
7. VAILSHERY, L.S. Microservices use in organizations worldwide 2021. *Statista* [interaktyvus]. 2022. [žiūrėta 2023-01-02]. Prieiga per internetą: <https://www.statista.com/statistics/1236823/microservices-usage-per-organization-size/>
8. Service-to-service communication. *Microsoft Learn* [interaktyvus]. 2022. [žiūrėta 2023-01-02]. Prieiga per internetą: <https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/service-to-service-communication>
9. BARABANOV, A. - MAKRUSHIN, D. Authentication and authorization in Microservice-based systems: Survey of Architecture Patterns. *Voprosy kiberbezopasnosti* [interaktyvus]. 2020. no. 4(38), p. 32–43. [žiūrėta 2023-01-02]. Prieiga per: doi: <http://dx.doi.org/10.21681/2311-3456-2020-04-32-43>
10. What is mutual TLS (mTLS)? *What is MTLS? | Mutual TLS | Cloudflare* [interaktyvus]. [žiūrėta 2023-01-02]. Prieiga per internetą: <https://www.cloudflare.com/learning/access-management/what-is-mutual-tls/>
11. LEINES-VITE, L. - PÉREZ-ARRIAGA, J.C. - LIMÓN, X. Information and Communication Security Mechanisms for microservices-based systems. *International Journal of Network Security & Its Applications* [interaktyvus]. 2021. Vol. 13, no. 6, p. 85–103. [žiūrėta 2023-01-02]. Prieiga per: doi: <https://doi.org/10.5121/ijnsa.2021.13607>
12. Introduction to JSON Web Tokens. *JSON Web Token Introduction - jwt.io* [interaktyvus]. [žiūrėta 2023-01-02]. Prieiga per internetą: <https://jwt.io/introduction>
13. MATEUS-COELHO, N. - CRUZ-CUNHA, M. - FERREIRA, L.G. Security in microservices architectures. *Procedia Computer Science* [interaktyvus]. 2021. Vol. 181, p. 1225–1236. [žiūrėta 2023-01-02]. Prieiga per: doi: <https://doi.org/10.1016/j.procs.2021.01.320>

14. HE, X. - YANG, X. Authentication and authorization of end user in Microservice architecture. *Journal of Physics: Conference Series* [interaktyvus]. 2017. Vol. 910, no. 1, p. 012060. [žiūrėta 2023-01-06]. Prieiga per: doi: <https://doi.org/10.1088/1742-6596/910/1/012060>
15. REZAEI NASAB, A. - SHAHIN, M. - HOSEYNI RAVIZ, S.A. - LIANG, P. - MASHMOOL, A. - LENARDUZZI, V. An empirical study of security practices for microservices systems. *Journal of Systems and Software* [interaktyvus]. 2023. Vol. 198, p. 111563. [žiūrėta 2023-01-06]. Prieiga per: doi: <https://doi.org/10.1016/j.jss.2022.111563>
16. What is OAuth 2.0? *What is OAuth 2.0 and what does it do for you?* [interaktyvus]. [žiūrėta 2023-01-06]. Prieiga per internetą: <https://auth0.com/intro-to-iam/what-is-oauth-2>
17. HARDT, D. RFC 6749. *The OAuth 2.0 Authorization Framework* [interaktyvus]. 2012. [žiūrėta 2023-01-06]. Prieiga per: doi: <https://doi.org/10.17487/RFC6749>
18. ANICAS, M. An Introduction to OAuth 2. *An Introduction to OAuth 2 | DigitalOcean* [interaktyvus]. 2021. [žiūrėta 2023-01-06]. Prieiga per internetą: <https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2>
19. CHANDRAMOULI, R. Security Strategies for Microservices-based Application Systems. *NIST Special Publication* [interaktyvus]. 2019, 800.204:800-204 [žiūrėta 2023-01-06]. Prieiga per: doi: <https://doi.org/10.6028/NIST.SP.800-204>
20. TAIBI, D. - LENARDUZZI, V. - PAHL, C. Architectural Patterns for Microservices: A Systematic Mapping Study. *Proceedings of the 8th International Conference on Cloud Computing and Services Science* [interaktyvus]. 2018. [žiūrėta 2023-01-06]. Prieiga per: doi: <http://dx.doi.org/10.5220/0006798302210232>