KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Giedrius Banaitis

MEDICINOS INTERNETINIŲ PASLAUGŲ, REALIZUOTŲ ASP.NET 2.0 TERPĖJE, SAUGUMO ANALIZĖ

Magistro darbas

Vadovas
dr. doc. V.Pilkauskas

KAUNAS, 2005

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

TVIRTINU
Katedros vedėjas
prof. habil. dr. H. Pranevičius

MEDICINOS INTERNETINIŲ PASLAUGŲ, REALIZUOTŲ ASP.NET 2.0 TERPĖJE, SAUGUMO ANALIZĖ

Informatikos magistro baigiamasis darbas

Kalbos konsultantė
Lietuviu kalbos katedros lektorė
dr. J. Mikelionienė

Vadovas
doc. dr. V. Pilkauskas

Recenzentas

Atliko
IFM 9/1 gr. stud. G. Banaitis

KAUNAS, 2005

KVALIFIKACINĖ KOMISIJA

Pirmininkas:	Laimutis Telksnys, akademikas
Sekretorius:	Stasys Maciulevičius, docentas
Nariai:	Rimantas Barauskas, profesorius
	Raimundas Jasinevičius, profesorius
	Jonas Kazimieras Matickas, docentas
	Jonas Mockus, akademikas
	Rimantas Plėštys, docentas
	Henrikas Pranevičius, profesorius

SUMMARY

Today's healthcare computer systems rely upon a disparate collection of legacy services, all developed separately and independently for patient records, radiological images, scheduling, billing, and administration. The Web is based on a client-server model. It consists of a set of servers, known as Web servers, which receive one request at a time and respond to that request without preserving state information, and a set of clients, known as Web browsers, which make requests based on user input and present results. Several efforts have been undertaken to address security in the Web, although the primary focus has been at the application level. These efforts have addressed the issue of protecting the privacy, accuracy and authenticity of transactions conducted over the Internet. In this work we will discuss cutting edge security tools in Microsoft's newest .NET framework version, how to use and witch to choose, how to integrate them in medical environment, will see how they meet today's medical web services requirements and will propose some methods to make securing of web services easier and more reliable.

.

TURINYS

Lentelių sąrašas

Paveikslėlių sąrašas

ĮVADAS

Judėjimas link standartais pagrįstų, silpnai tarpusavyje susietų technologijų, į paslaugas orientuotos architektūros, lemia galybę pasikeitimų šiandienos informacinių technologijų sferoje. XML tampa *lingue franca* informacijai, esančiai internete, o internetinės paslaugos, pagrįstos XML veda link naujo požiūrio į sistemų tarpusavio integraciją, ko pasėkoje XML duomenų srautų apimtys internete auga eksponentiškai. Su šiuo milžinišku kiekiu informacijos XML formate atsiranda ir naujos saugumo rizikos bei problemos. Šiandienos IT sferai svarbu suvokti, kad XML ir internetinių paslaugų privalumai gali būti sėkmingai pritaikomi tik tuo atveju, jeigu yra įvertinamos šių naujovių rizikos ir problemos.

Į paslaugas orientuota architektūra gali būti itin sėkmingai taikoma ir medicinos srityje, integruojant šiuolaikinę medicininę aparatūrą su esančiomis pacientų duomenų bazėmis, diagnostikos algoritmais, išnaudojant nuotolinio gydymo ir diagnostikos privalumus. Šiandienos didžioji medicinos sferos informacinės sistemos yra pagrįsta tarpusavyje nesuderinama, pasenusia programine įranga, kuri buvo kurta kiekvienai užduočiai atskirai bei nepriklausomai – pacientams registruoti, medicininiams tyrimams atlikioti, finansinei apskaitai ir t.t. Dėl to saugumo užtikrinimo užduotis tampa dar sudėtingesne dėl kelių priežasčių – pirma, tai remiantis esmine saugumo teorijos aksioma (sistemos saugumas nusakomas silpniausios sistemos vietos saugumu) pasenusios sistemos yra potencialūs sistemos silpni taškai, antra – dėl pasenusių sistemų specifikos tenka dažnai rinktis ne saugiausius metodus sistemų tarpusavio integracijai.

Šiandien yra aibė įrankių, sistemų, skirtų palengvinti programuotojų darbą kuriant internetines paslaugas, viena labiausiai paplitusių sistemų, skirtų kurti internetinėms paslaugoms, taipogi ir viena pažangiausių yra Microsoft kompanijos sukurtas karkasas .NET bei speciali kalba ASP. Microsoft kompanija aktyviai kuria naujas technologijas bei standartus, todėl per savo produktus gali suteikti programuotojams galimybę naudotis pačiomis pažangiausiomis technologijomis kuriant šiuolaikiškas internetines paslaugas.

Microsoft .NET technologiją naudojome ir savo programinei įrangai kurti. Brokerio dalis buvo sukurta naudojant Microsoft Visual Studio.NET 2003, MS SQL Server 2000, tuo tarpu kliento dalis buvo kuriama pasitelkiant dar oficialiai

nepasirodžiusio Microsoft naujausio produkto bandomąją versiją, kurioje itin daug pažangiausių technologijų, tarp jų ir ASP .NET 2.0 kalba, kuri po oficialaus pristatymo be jokios abejonės sukels perversmą internetinių paslaugų projektavimo bei realizavimo srityse. Bandydami susieti šias dvi sistemas, sukurtas skirtingų kartų programavimo įrankiais, turėjome progą išsamiau susipažinti su saugumo tendencijomis, su naujausių technologijų taikymu medicinos kontekste.

Pagrindinis darbo tikslai yra susipažinti su medicininių paslaugų apsaugojimo specifika, ištirti naujausios kartos .NET priemones interneto paslaugoms realizuoti, išanalizavus dažniausiai naudojamus atakų tipus paruošti metodiką medicinos internetinių paslaugų saugumo lygiui didinanti, palyginti naujajame .NET karkase esančias priemones su kitomis esančiomis rinkoje bei aptarti galimus esančios saugumo išplėtimus.

Viena svarbiausių ir opiausių problemų, kuri kamuoja šiandieninius internetinių paslaugų tiekėjus yra DoS (*Denial of Service*) tipo atakos, kurios išveda internetines paslaugas teikiančią programinę įrangą iš rikiuotės, tokiu būdu kiti vartotojai negali gauti atsakymų į savo užklausas. Pavojingiausias šių atakų bruožas yra jų aptikimo sudėtingumas, nes jos dažnai slepiasi po užklausomis, suformuotomis sistemoje galiojančių taisyklių. Šio tipo atakos yra ypač svarbios medicinos srityje, kur dažnai delsti negalima, arba uždelsimas turi būti minimalus. Turint galvoje, kad vis didesnė medicinos apsaugos sistemos dalis kompiuterizuojasi ir ne mažą dalį savo infrastruktūros apjungia naudojant XML srautus, kiltų didelis pavojus visuminei, jeigu būtų sutrikdytas sistemos darbas šio tipo atakomis.

Ne mažos dalies nagrinėtų šaltinių autorių užsimena apie šia problemą, tačiau daugelis ir palieka ją išsamiau nepaliestą, apsiribodami nuorodomis į kuriamus standartus, nesukonkretindami konkrečios realizacijos.

Internetinių paslaugų saugumo tema yra aktuali šiandien, todėl yra nemažai diskutuojama tarp šios srities profesionalų, tikintis surasti universalesnius, paprasčiau realizuojamus, našesnius internetinių paslaugų saugumo užtikrinimo būdus.

1. Medicinos internetinių paslaugų saugumo reikalavimų analizė

1.1. Saugumo reikalavimai medicinos internetinėms paslaugoms

Medicinos internetinės paslaugos turi tenkinti šešis esminius saugumo reikalavimus:

Identifikavimas – gavėjas turi turėti galimybę patikrinti siuntėjo tapatybę, šis reikalavimas yra reikalingas daugeliui iš toliau sekančių reikalavimų.

Autorizavimas – identifikavus siuntėją, galima spręsti, ar jam leidžiama vykdyti vieną ar kitą funkciją/operaciją, pasiekti tam tikrus duomenis. Tai vienas svarbiausių reikalavimų, taipogi vienas sudėtingesnių, ypač situacijose, kai yra daugelio lygių heterogeniškų sistemos yra susiejamos, kurių kiekviena arba didžioji dalis turi savo saugumo nuostatas ir taisykles. Šiai problemai spręsti buvo sukurtas WS-Security standartas. WS-Security aprašo abstraktų sluoksnį, kuris yra aukščiau konkrečios sistemos taikomosios programos saugumo technologijos (PKI, Kerberos ir kt.), tokiu būdu tokios skirtingos infrastruktūros gali buti sujungtos bendru patikimu ryšiu.

Konfidencialumas – tai reikalavimas, kad neautorizuotas veikėjas negalėtų perskaityti ar įsiterpti tarp dviejų komunikuojančių pusių. Infrastruktūros, kaip kad PKI (*Public Key Infrastructure*) ar Kerberos naudoja duomenų kodavimą tam, kad duomenų apsikeitimas išliktų konfidencialus. PKI gali naudoti kodavimą konfidencialumui užtikrinti tiek duomenų tranzite, tiek saugojime. VPN (*Virtual Private Networks* – virtualūs privatūs tinklai) bei SSL (*Secure Sockets Layer* – saugių jungčių sluoksnis) gali apsaugoti duomenų saugumą perduodant juos tarp dviejų taškų, bet neužtikrina duomenų saugumo juos išsaugojant po parsiuntimo, nei tarpiniuose taškuose. Be to, naudojant SSL, duomenys tarpiniuose taškuose yra iškoduojami, tokiu būdu atsiranda papildoma, saugumo atžvilgiu silpna, vieta.

Duomenų nepažeidžiamumas (*integrity*). Kitaip nei konfidencialumas, duomenų nepažeidžiamumas turi užtikrinti du dalykus: pirma, gauti duomenys turi būti identiški išsiųstiems, kitais žodžiais tariant, duomenų nepažeidžiamumo sistema turi užtikrinti, kad duomenys nepasikeis tranzito metu, net jeigu tai būtų persiuntimo klaida ar tyčinis įsikišimas. Antrasis reikalavimas duomenų nepažeidžiamumui, kad bet kuriuo momentu

ateityje gavėjas turėtų galimybę patikrinti, ar skirtingos to paties dokumento kopijos yra identiškos savo turiniu.

Nepaneigiamumas. Siuntėjas ir gavėjas turi galėti esant reikalui patvirtinti, kad siuntėjas išsiuntė, o gavėjas gavo, įskaitant laiką, kada duomenys buvo išsiusti ir faktą, kad gavėjas gavo tik vieną kopiją.

Operatyvi gynyba (*Operational defence*). Sistema turi sugebėti aptikti ir apsisaugoti nuo klaidingų XML duomenų srautų, kaip kad XdoS (*XML Denial of Service*) ar XML virusų, taip pat turi būti operatyviai išplečiama esančiam personalui ir infrastruktūrai.

1.2. Saugumo įvertinimas

Saugumo įvertinimo tikslas yra grėsmių ir silpnų vietų įvertinimas sistemos kritinių vertybių (duomenų, reputacijos, nenutrūkstamo veikimo) atžvilgiu ir sertifikuoti visas sistemoje realizuotas saugumo sistemas kaip adekvačias, pilnai užtikrinančias saugumą ar atitinkančias rizikos lygį.

Rizika yra apibrėžiama kaip tikimybė sugadinti ar prarasti kurį nors informacinės sistemos resursą. Dažniausiai yra klasifikuojama nuo konkrečių komponentų iki abstrakčių žymių ar resursų. Reputacija, programinė įranga, aparatūra, duomenys ar netgi personalas gali būti įvardijami kaip resursai, kuriems įvertinama rizika.

Turtas (*asset*) – tai bet kurios sistemos esybės vertės elementas. Turtai sistemoje gali būti apibūdinami įvairiais stambumo lygiais: slaptažodis, saugomas užšifruotoje byloje, fizinė tarnybinė stotis ar visas tinklas. Turtas visada priklauso kitoms esybėms. Savininkas įvertina turto vertę ir maksimalias išlaidas, skiriamas jam apsaugoti.

Priimtina pažeidžiamumo rizika yra tada, kai turto saugumo nulaužimo išlaidos yra didesnės už turto vertę.

Grėsmė yra bet kokia piktavališka veikla ar atsitiktinumas, kurie gali sukompromituoti sistemą.

Silpna vieta – tai sistemos projektavimo, realizavimo trūkumas, kuri didina tikimybę (riziką) prarasti turtą.

Saugumo įvertinimas nėra analizė aktyviu saugumo įrankiu ar sistemos atakavimu pasamdytų saugumo ekspertų, tai yra atskiras žingsnis programinės įrangos projektavime, norint pagerinti kuriamos programinės įrangos kokybę.

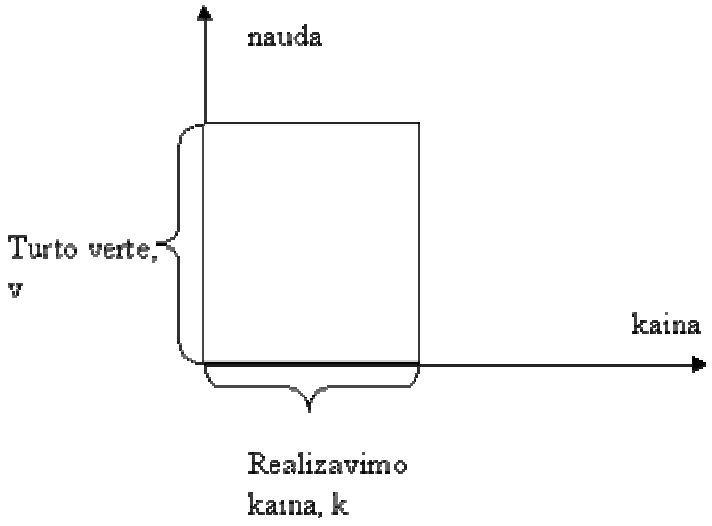
Įvertinant saugumą, į pagalbą dažnai pasitelkiamos grėsmės-sprendimo lentelės, kuriose yra surašomos numatomos grėsmės, šių grėsmių veikiamo turto vertės įvertinimas, rizika, sprendimas bei sprendimo kaina.

1 lentelė. Į paslaugas orientuotoje architektūroje naudojami standartai

Grėsmė	Turto vertė	Tikimybė	Sprendimas	Sprendimo realizavimo kaina
Ugniasienės išorėje esanti interneto tarnybinė stotis gali būti sukompromituota.	Didelė	Didelė	Įdiegti įsilaužimo sensorių, naujausius saugumo atnaujinimus, patikrinti vartotojų teisių politiką visam dokumentų medžiui	Vidutinė
Pasenusios(*legacy*) programinės įrangos CORBA ryšys su tarnybine stotimi yra nesaugiame WAN.	Didelė	Vidutinė	Realizuoti taškas-taškas(*p2p*) SSL ryšį.	Didelė
Silpni vartotojų slaptažodžiai	Didelė	Didelė	Neleisti vartotojams naudoti trumpų slaptažodžių, nustatyti slaptažodžių galiojimo laiką, neleisti vartotojams naudoti trijų paskutinių savo slaptažodžių	Maža

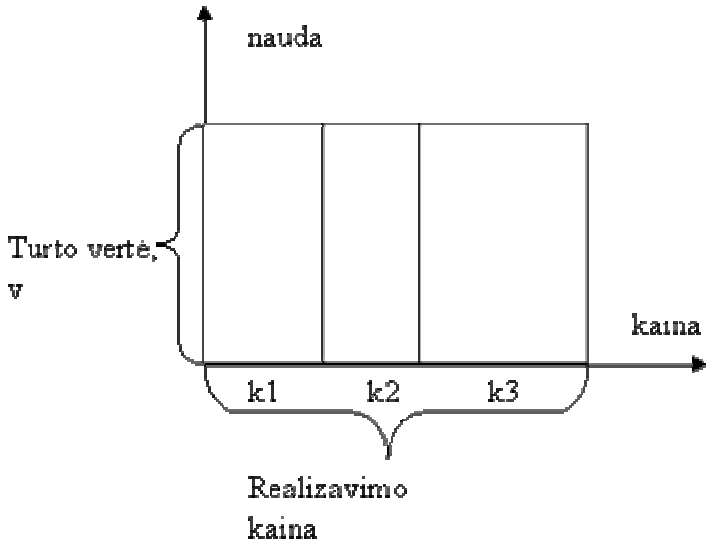
Žvelgiant abstrakčiai, saugumo įvertinimo ir kainos-naudos proceso analizė susideda iš aibės sprendimų. Kiekvienas sprendimas apsaugo tam tikros vertės turtą

realizuojant saugumo komponentą, turintį tam tikrą realizacijos kainą. Paprasčiausias kainos-naudos variantas grafiškai atrodytų taip:



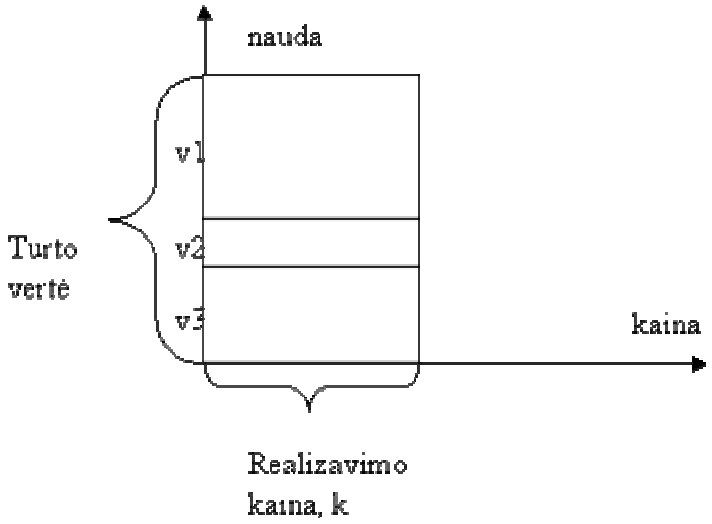
1 pav. Paprasčiausias kainos-naudos variantas

Tačiau realiuose projektuose, turto apsaugojimui gali reikėti realizuoti kelis saugumo komponentus:



2 pav. Kainos-naudos variantas, kai reikalingi keli komponentai vienam turtui apsaugoti

Taip pat galimas atvejis, kai keli atskiri turtai (*assets*) gali būti apsaugojami vienu saugumo komponentu:



3 pav. Kainos-naudos variantas, kai apsaugomi keli turtai vienu komponentu

Programinės įrangos saugumo įgyvendinimas gali būti įsivaizduojamas, kaip kainos-naudos blokų poaibio parinkimas iš visų galimų variantų taip, kad būtų apsaugojama maksimali turto vertė panaudojant apibrėžtą biudžetą. Realiose sistemose blokai nėra idealūs, gali persidengti keliose vietose, būti ne stačiakampio formos ir pan. Paprastumo dėlei ignoruosime šiuos atvejus. Iš uždavinio formuluotės matyti, kad turimas standartinis optimizacijos uždavinys. Šis uždavinys sprendžiamas daug paprasčiau, jeigu kiekvienam turtui nustatomi išankstiniai prioritetai, kuriuos galima paimti, pavyzdžiui, iš grėsmės-sprendimo lentelės, prilyginant prioritetą rizikai.

1.3. Atakų metodika

Šiame skyriuje analizuojame, kaip potencialūs įsilaužėliai renka informaciją, reikalingą įsilaužimui atlikti bei populiariausius atakų metodus, kuriais įsilaužėlis gali perimti sistemos valdymą į savo rankas arba apkrauti ją taip, kad negalėtų naudotis kiti.

1.3.1. WSDL dokumentų naudojimas informacijos rinkimui

WSDL dokumentai yra esminė šaltinis, kuris aprašo prieinamas iš išorės funkcijas kartu su laukiamais rezultatais. Kadangi dažniausiai WSDL dokumentai yra viešai prieinami vartotojams, jie nėra apsaugoti nuo neautorizuoto peržiūrėjimo. Net jeigu ir yra

šie dokumentai apsaugoti, jie bus vienas pirmųjų įsilaužėlio taikinių ir pakankamai lengvai pasiekiamų. WSDL dokumentas suteikia aiškią informaciją apie taikomosios programos paskirtį, funkcinį pažeidžiamumą, įeities taškus bei žinučių tipus. Kadangi WSDL dokumente yra saugoma XML schema, sekantis įsilaužėlio žingsnis greičiausiai bus dirbtinių užklausų sugeneravimas remiantis WSDL dokumente rasta XML schema. Pasirenkant operacijų aibę bei suformuojant užklausą remiantis XML schema nėra sudėtinga užmegzti ryšį su internetine paslauga. Šiame taške įsilaužėlio tikslas į teisingai suformuluotą užklausą įterpti klaidinančius duomenis. Tokiu būdu galima įsitikinti, ar yra naudojamas XML schemos tikrinimas, jeigu jis yra įjungtas, kokius klaidų pranešimus jis generuoja. Kadangi paprastai programuotojai klaidų pranešimuose nori suteikti kiek galima daugiau informacijos klientui apie klaidos pobūdį ir priežastį, klaidų pranešimai taip pat yra vienas geriausių šaltinių įsilaužėliui apie saugumo silpnas vietas.

Pavyzdžiui:

Žinodamas, kad schemos griežtas patikrinimas yra atliekamas ne visose operacijose, atakuojantysis taikysis į schemos netikrinančias operacijas, bandydamas jose įterpti klaidinančius duomenis. Tokiu būda atakuojantysis gali lengvai surasti rimtą saugumo klaidą ar vietą, per kurią būtų galima atlikti DOS tipo ataką.

Žinant, kaip sistema reaguoja į klaidingus duomenis, galima susidaryti įspūdį apie sistemos vidinę technologiją, pavyzdžiui apie XML gramatikos nagrinėjimo ir pripažinimo (*validation*) posistemes.

Reakcija į neautorizuotą bandymą įvykdyti vieną ar kitą operaciją gali suteikti naudingos įsilaužėliui informacijos, kaip apeiti autorizavimo reikalavimus.

Šioje stadijoje įsilaužėlis renka informaciją, kurios vietos yra silpniausios ir kaip būtų galima jomis pasinaudoti.

1.3.2. Potencialių pažeidžiamų vietų tyrimas

Po informacijos rinkimo iš WSDL dokumentų fazės, seka atakų bandymo fazė. Įsilaužėlis testuoja sistemą naudodamas paprastas atakas silpnoms vietoms tikrinti. Šios atakos yra charakterizuojamos laužimais, naudojant „grubią jėgą“ – t.y. naudojant raidžių kombinacijas, skaičius bei specialius simbolius bandant nustatyti, kurios suformuotos užklausos pažeidžia sistemą. Pavyzdžiai būtų:

WSDL skanavimas. Kadangi WSDL yra aprašomos visos operacijos prieinamos vartotojui, įsilaužėlis tiesiog iš eilės bando visoms operacijoms pateikti užklausų šablonus kol randama pažeidžiama vieta. Šis „beldimo į visas duris, kol vienos iš jų atsidarys“ metodas yra efektyvus, kai programinę įrangą rašė nepatyrę programuotojai ar paprasčiausiai per aplaidumą WSDL dokumente buvo paliktos funkcijos, kurios neturėjo matytis išorėje.

Parametrų klastojimas. Kadangi operacijų parametrai yra aprašyti WSDL dokumente, įsilaužėlis gali mėginti įvairius parametrų šablonus, bandydamas pasiekti informaciją, prieinamą tik autorizuotam vartotojui. Pavyzdžiui, yra bandoma parametrų reikšmėse naudoti specialius simbolius (ar kitą netikėtą turinį) taip bandant nutraukti internetinės paslaugos veikimą dėl vidinės klaidos.

1.3.3. Priverstinis gramatikos nagrinėjimas (*Coersive parsing*)

Jeigu XML yra daugiareikšmis(*verbose*) savo aprašyme, tai SOAP yra ypač daugiareikšmis. Pirmas žingsnis, kurį atlieka internetinės paslaugos tiekėjas, yra elementų išskyrimas iš SOAP žinutės. Šio žingsnio tikslas yra atpažinti parametrus, nustatyti, kurią operaciją iš prieinamų vartotojui parinkti atlikimui. Šis paprastas žingsnis yra lengvas taikinys įsilaužėliui, norinčiam atlikti DOS tipo ataką ar piktavališkai apkrauti tarnybinę stotį. Kiti pavyzdžiai, kaip gali būti naudojamas priverstinis gramatikos nagrinėjimas:

Rekursinė apkrova. Šis atakos metodas yra vykdomas pasitelkiant idealiai teisingą SOAP dokumentą ir padarant mažą pakeitimą, norint apgauti XML Schemos pripažinimą. Įsilaužėlis pasirenka SOAP žinutės šaką ir pakartoja ją, kad būtų sukurta gausiai šakota struktūra. Pavyzdžiui, elementas su kardiogramos duomenimis yra pakartojamas daug kartų, tokiu būdu priverčiant programinę įrangą stipriai apkrauti procesorių apdorojant ir pripažinant kiekvieną elementą.

Dydžio sąlygojamas perkrovimas (*oversize payload*). XML gramatikos nagrinėjimas tiesiogiai priklauso nuo SOAP žinutės dydžio. Rezultate dideli kiekiai procesoriais taktų yra sunaudojami dideliems dokumentams apdoroti. Įsilaužėlis gali pasiųsti ypač dideles užklausas, kad galėtų išvesti sistemą iš rikiuotės.

Atsakymo ataka. Įsilaužėlis paprasčiausiai siunčia pasikartojančias SOAP užklausas, kad galėtų perkrauti tarnybinę stotį. Šis atakos tipas sunkiai aptinkamas, nes yra siunčiamas tvarkingas tinklo paketas, su tvarkinga HTTP užklausa.

1.3.4. Išorinių nuorodų atakos

Prasta ar neteisinga vidinių resursų koordinacija gali būti išnaudota įsilaužėlių Dos atakoms atlikti ar duomenims pavogti.

Išorinių esybių atakos. Nuorodos į išorines esybes leidžia SOAP dokumentams susirinkti duomenis patiems iš trečiai šaliai priklausiančių vietų, todėl įtraukus didelį kiekį išsibarsčiusių šaltinių galima kompromituoti sistemą DoS tipo atakomis.

Schemos pakeitimas. XML schemos nurodo formatavimo instrukcijas gramatikos nagrinėjimo posistemėms, nagrinėjančioms XML dokumentus. Kadangi XML schemos aprašo reikalingas išankstinio skaičiavimo instrukcijas, todėl tampa įsilaužėlių taikiniu – jis tiesiog pakeičia XML schemos elementus klaidinančia faksimile, toliau gali įrašyti datas vietoj skaičių, su kuriais atliekamos aritmetinės operacijos.

Maršruto nukreipimas (*Routing Detour*). Internetinės paslaugos adresavimo specifikacija nusako būdą, kaip nukreipti tiesioginį SOAP srautą per tarpinius taškus, naudojant XML žymes, kurios priskiria maršrutizavimo instrukcijas. Jeigu atakuojantysis perima vieną iš šių tarpinių taškų, jis gali įterpti klaidinančias maršrutizavimo instrukcijas, kurios nukreips į jį ateinančią informaciją į neautorizuotą tašką, kur kritine informacija galės pasinaudoti neautorizuoti asmenys. Ši technologija taip pat gali būti panaudota DoS atakos atlikimui maršrute nurodžius neegzistuojantį tašką.

1.3.5. Piktybinis turinys (*Malicious Content*)

Virusai, „Trojos arklių" programos, kurios gali būti perduotos su visais atžvilgiais teisinga XML užklausa, gali lengvai pažeisti internetines aplikacijas. Dvejetainiai duomenys, tokie kaip kardiogramos duomenys, sukompiliuotas programos kodas bei taikomosios programinės įrangos specifiniai dokumentai – visa tai gali būti modifikuota, norint sukelti nestandartines situacijas internetines paslaugas aptarnaujančioje programinėje įrangoje.

1.3.6. SQL injekcijos

Struktūrizuotos užklausų kalbos (*SQL – structured query language*) sakiniai yra pagrindas duomenų bazių įrašų manipuliacijai. Tarnybinės stotys konstruoja SQL instrukcijas remdamiesi užklausomis, gaunamomis iš kliento. Šios užklausos gali būti sudarytos iš SQL sakinių, o taipogi ir iš specifinių duomenų laukų, kurie yra suformatuoti taip, kad būtų galima neteisėtai parsiųsti, atnaujinti ar įterpti informaciją. Tokio pobūdžio sakinių neturėtų būti SOAP žinutėse ir bet kokie bandymai į SOAP žinutė įterpti tokius sakinius turėtų būti aptikti ir neutralizuoti.

1.4. Šiandienos saugumo iššūkiai

Nepriklausomas nuo transportavimo saugumas. Nors SOAP žinutės dažnai siunčiamos naudojant HTTP protokolą, kiti transporto protokolai taip pat gali būti naudojami, nes SOAP yra nepriklausoma nuo komunikacijos protokolų. SOAP žinutės gali būti siunčiamos ir naudojant SMTP, FTP ar žinučių eiles (*Message Queues*). Kadangi HTTP yra protokolas, kuris neturi būsenos ir ne visiškai patikimas, medicininei kritinei informacijai būtų geriau naudoti kitą protokolą.

Suderintas identifikavimas (*Blended Authentification, Granular Message-Level Access Control*). SSL sertifikatai ir HTTP vartotojo vardas/slaptažodis yra pateikiami, norint užsitikrinti priėjimą prie tam tikros informacijos, visa tai dažniausiai yra apibrėžta vaidmenimis (*roles*), atsakomybe ir privilegijomis. Šio tipo programinės įrangos lygio autorizacija yra orientuota į sesijas ir negali būti naudojama suderintam identifikavimui. Dėl to individualios SOAP/XML žinutės, taip pat ir WSDL dokumentai nėra tikrinami, ar juos nėra neleistino turinio, galinčio pažeisti sistemą.

Jungtinis saugumas (*Federated security*). Viena ir ta pati SOAP žinutė gali pereiti per aibę transporto protokolų (pvz. HTTP ir SMTP) vienos transakcijos kontekste. SSL suteikia adekvatų konfidencialumą ir identifikavimą vienai taškas iš taško sesijai, tačiau negarantuoja reikiamo saugumo esant daugeliui tarpinių taškų.

Orientuotas į dokumentą identifikavimas. Internetinės paslaugos pagal prigimtį yra pagrįstos žinučių pasikeitimu tarp paslaugos tiekėjo bei vartotojo ir neturi garantijos, kad ryšys vyksta tiesiogiai tarp šių dviejų objektų. Iš čia seka, kad nėra garantuojamas konfidencialumas tarp kliento ir paslaugos tiekėjo naudojant tradicinius protokolus.

Auditas. Tradicinės saugumo technologijos neatlieka audito visiems susijusiems dokumentams sesijos metu.

1.5. Į paslaugas orientuotoje architektūroje naudojami standartai

Į paslaugas orientuotos architektūros stiprybė yra standartizuotose sąsajose, todėl šiame skyriuje trumpai apžvelgsime esančius standartus, susijusius su internetinių paslaugų teikimu bei pabandysime apžvelgti tendencijas, atkreipti dėmesį į vis didesnį populiarumą įgyjančius standartus.

2 lentelė. Į paslaugas orientuotoje architektūroje naudojami standartai

Standartas	Standarto aprašymas	Paplitimas	
		Šiandien	Tendencijos
HTTPS	HTTP ryšys tarp paslaugos tiekėjo ir kliento naudojant SSL/TLS – saugų tunelį, kuris užtikrina siunčiamos informacijos konfidencialumą persiuntimui naudojant viešus interneto kanalus.	Labai didelis	Stabilios
XML	Žymėjimo kalba, skirta struktūrizuotų duomenų mainams.	Labai didelis	Stabilios
XML schema	Sutartinė kalba, kuria apibrėžiamos duomenų struktūros bei ryšiai tarp jų.	Didelis	Augančios
SOAP	Standartas, apibrėžiantis žinučių apsikeitimą tarp programinės įrangos.	Vidutinis	Augančios
WSDL	WSDL – struktūrizuotas elektroninės paslaugos aprašymas – kaip pasiekti paslaugą ir kokias operacijas galima atlikti.	Vidutinis	Augančios
WS-Security	Mechanizmas slaptos informacijos patalpinimui SOAP žinutėse	Vidutinis	Augančios
XML-Encryption	WS-Security standarto poaibis, aprašantis XML duomenų užkodavimo/iškodavimo procesą.	Vidutinis	Augančios
XML Signature	Mechanizmas, skirtas XML dokumento kilmės šaltinio patikrinimui bei duomenų vientisumo užtikrinimui, WS-Security standarto poaibis.	Didelis	Augančios
SAML	Karkasas, aprašantis autentifikacijos ir autorizacijos duomenų apsikeitimo procesą.	Vidutinis	Augančios
WS-Trust	Standartas jungtinio (*federate*) saugumo tinklams kurti.	Mažas	Augančios

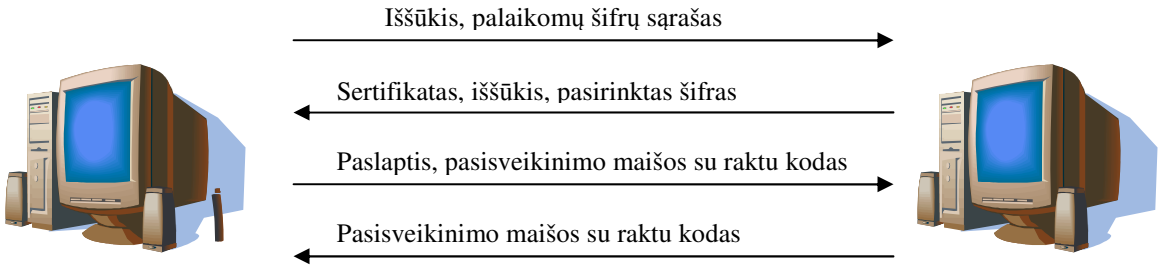
1.5.1. Kriptografiniai protokolai

Daugelis IAI sistemų yra realizuotos naudojant kriptografinius protokolus. Šie protokolai remiasi viešo ir slapto rakto mechanizmais. Protokolai nusako, kaip klientas ir tarnybinė stotis turi apsikeisti raktais, kad būtų galima pradėti koduotą ryšio sesiją. Po to,

kai įvyksta pasikeitimas raktais, klientas gali naudotis ryšiu nebijodamas, kad kas nors gali perskaityti ar pakeisti ryšio kanalu keliaujančią informaciją.

SSL/TLS protokolas

Viešo rakto algoritmuose, šifruojančiojo privatus raktas yra laikomas saugiai, todėl slaptu raktu pasirašyta žinutė laikoma identifikacijos patvirtinimu. SSL veikimo principo schema:



4 pav. SSL veikimo principas

SSL yra transporto protokolas, jis randasi aukštesniame lygmenyje nei TCP, bet žemiau programos sluoksnio. Be aukščiau minėtų SSL savybių yra dar viena, ypač svarbi kalbant apie internetines paslaugas – ryšio užmezgimo metu yra identifikuojamas ne tik klientas tarnybinės stoties atžvilgiu, bet ir tarnybinė stotis kliento atžvilgiu.

Kerberos/DCE protokolas

Kerberos dažniausiai yra naudojamas tarp korporatyvinių tinklų. Kerberos leidžia identifikaciją be identifikacijos duomenų siuntimo. Identifikacijos mechanizmas veikia taip: klientas siunčia savęs atpažinimo duomenis (vardas, asmens kodas, elektroninio pašto adresas) identifikacijos tarnybinei stočiai, ši grąžina jam bilietą (*Ticket-granting ticket - TGT*). TGT yra užšifruotas taip, kad tik klientas turintis savo atpažinimo duomenis atitinkantį slaptažodį gali šį bilietą iššifruoti ir naudoti. Kai klientas nori prisijungti prie tarnybinės stoties, naudojančios Kerberos, jis siunčia TGT į raktų paskirstymo centrą (*Key Distribution Center - KDC*). KDC grąžina sesijos bilietą, kuris sudarytas iš vartotojo identifikacinio vardo ir slapto sesijos rakto. Klientas naudoja sesijos bilietą saugiam prisijungimui prie tarnybinės stoties programinės įrangos.

Kerberos taip pat turi galimybę koduoti ir užtikrinti žinučių turinio autentiškumą. Žinutės koduojamos naudojant DES, o žinutės vientisumas (*integrity*) yra užtikrinamas RSA MD4/MD5 pagalba.

Nors Kerberos bei DCE yra saugios ir patikrintos technologijos, jos netinkamos naudoti tais atvejais, kai sistema turi daug vartotojų.

1.6. Identifikacija

Identifikacijos mechanizmus galima suskirstyti į dvi kategorijas: pirmajai kategorijai būtų priskiriami mechanizmai, pagrįsti slaptažodžiu, antrajai kategorijai priklausytų iššūkiu-atsakymu(*challenge-response*) pagrįsti mechanizmai. Nors slaptažodžiais pagrįsti mechanizmai ir yra populiaresni, jie turi apribojimų, nagrinėjamų žemiau. Iššūkiu-atsakymu pagrįsti mechanizmai yra sunkiau realizuojami, tačiau suteikia aukštesnį saugumo lygį.

1.6.1. Identifikacija slaptažodžiu

Paprasčiausias identifikacijos būdas yra pagrįstas bendra paslaptimi, kuri dažniausiai būna slaptažodžio, asmenį identifikuojančio numerio (PID) ar slaptos frazės forma. Ryškiausia slaptažodžiu pagrįstų sistemų savybė yra identifikacijos nepriklausymas nuo siunčiamos informacijos, siunčiamos identifikaciją atliekančios pusės. Daugelis mūsų yra susidūrė su slaptažodžiu pagrįstomis sistemomis – operacinės sistemos, bankomatai, HTTP Basic identifikacija. HTTP Basic identifikacijos atveju vartotojas pareikalauja resurso, kurio gavimui reikalinga identifikacija ir nesuteikia informacijos, reikalingos identifikacijai atlikti, sistema atmeta prašymą. Vartotojo naršyklė reikalavimo atmetimą interpretuoja ir to rezultate paprašo vartotoją suteikti identifikavimui reikalingą informaciją. Jeigu vartotojo identifikacijos duomenys patenkina tarnybinės stoties reikalavimus, tarnybinė stotis išduoda reikalaujamą informaciją.

Šios sistemos trūkumas yra tas, kad slaptažodis yra siunčiamas interneto tinklu ir yra galimybė, kad siuntimo metu galima kas nors jį perimti. Norint nuo to apsisaugoti, naudojamas kriptografija. Labiausiai paplitęs identifikacijos slaptažodžiu scenarijus yra

pagrįstas SSL – klientas susijungia su tarnybine stotimi naudodami koduotą protokolą ir tik tada yra siunčiami duomenys, reikalingi vartotojo autentifikavimui.

Kita slaptažodžių problema yra ta, kad šiandien yra aibė paslaugų, reikalaujančių vartotoją turėti kiekvienai atskirą slaptažodį. Vartotojas yra nepajėgus atsiminti visus slaptažodžius ir dėl to dažniausiai juos bando užsirašyti, naudoti tuos pačius skirtingose vietose, rinktis trumpus ir lengvai įsimenamus slaptažodžius, kas mažina saugumą, nes tokie slaptažodžiai lengvai atspėjami žodyno atakos metu. Siekiant sumažinti žmoniškąjį faktorių, dažnai yra reikalaujamas tam tikras slaptažodžio sudėtingumas (reikalaujama įterpti skaičius ar simbolius) arba reikalaujama slaptažodžius periodiškai pasikeisti.

Slaptažodžius galima pamiršti, perimti, nulaužti, tačiau jie vis dar naudojami šiandienos programinėje įrangoje, tačiau juos reikėtų naudoti tik žemo rizikos lygio programinėje įrangoje, aukštesnio saugumo reikalaujančiose sistemose tikslinga naudoti PKI pagrįstas sistemas.

1.6.2. Identifikacija iššūkiu-atsakymu (IAI)

IAI sistemose identifikuojanti pusė pirmiausiai išsiunčia norinčiam identifikuotis iššūkį. Norintysis identifikuotis atlieka tam tikrus veiksmus su gautu iššūkiu ir gautą rezultatą siunčia tarnybinei stočiai. Jeigu rezultatai yra priimtini, vartotojas yra identifikuojamas. Šios sistemos pastebimas bruožas yra tai, kad kliento atsakymas priklauso nuo iššūkio. HTTP digest identifikacijos tipas yra vienas iš realizuotų nagrinėjamo modelio pavyzdžių.

IAI sistemos yra saugesnės už slaptažodžiu pagrįstas sistemas, nes jos nesiunčia identifikuojančios informacijos internetu. Identifikuojanti informacija visada lieka vartotojo kompiuteryje. Dar vienas privalumas yra tai, kad kliento atsakymas priklauso nuo iššūkio, todėl eliminuojamos atkartojimo (*replay*) atakos.

IAI sistemos veikia nesiųsdamos slaptažodžio internetu, tačiau slaptažodžiai visgi jose yra naudojami – klientas naudoja slaptažodį gauto iššūkio pakeitimui.

2. ASP.NET 2.0 Saugumo priemonių tyrimas

2.1. Saugumo metodikų našumo tyrimas

Medicininėje informacinėje sistemoje yra aibė įvairių paslaugų, kurios skiriasi savo duomenų saugumo kritiškumu, funkciniais reikalavimais (mažas vėlinimas, našumas), duomenų apimtimis (paciento įrašai, kardiogramų duomenys, rentgeno fotografijos ir pan.), todėl projektuojant tokio tipo paslaugas svarbu žinoti, kurie saugumo mechanizmai geriausiai tinka konkrečioms užduotims.

Saugumui užtikrinti naudojami mechanizmai turi poveikį šioms sistemos savybėms:

- našumą
- išplečiamumą
- pritaikomumą
- saugumą

Kadangi tai yra vieni svarbiausių sistemos parametrų, analizavome, kokį poveikį .NET ASP 2.0 realizuoti saugumo mechanizmai turi šiems sistemos parametrams.

2.1.1. Tyrimo priemonės ir strategija

Savo testams naudojome ACT (*Microsoft Application Center Test*) įrankį, kuris buvo sukurtas tarnybinių stočių našumo bei išplečiamumo problemų tyrimui. Šio įrankio pagalba taip pat galima tirti ir ASPX puslapius, bei juose esančius komponentus. ACT stimuliuoja didelės vartotojų grupės veiksmus, pavyzdžiui, prisijungimus prie tarnybinės stoties, užklausų siuntimus jai bei reakciją į gautus atsakymus. Šio įrankio pagalba taip pat galima kurti realistiškus testavimo scenarijus naudojant atsitiktiniu būdu parinktus parametrų rinkinius, nes realiame sistemos naudojime vartotojai dažniausiai siunčia užklausas, kurios yra nepanašios viena į kitą. Dar viena savybė, kuri mūsų analizėje buvo itin svarbi – ACT gali įrašinėti testavimo rezultatus, kuriuos vėliau galima naudoti analizei.

ACT supranta *Basic, Kerberos* ir *Digest* identifikavimo tipus, tačiau jame nerealizuotas *ASP.NET Forms* identifikavimo tipas, todėl mes tiesiogiai redagavome užklausos duomenis, kad imituoti šį identifikavimo tipą iš kliento pusės.

Tyrime naudotų kompiuterių konfigūracija:

Klientas

Procesoriaus tipas: AMD, 1,1Mhz, K7

Procesorių skaičius: 1

Kietasis diskas: 80GB, 8Mb spartinančioji atmintinė

Operatyvinė atmintinė: 1GB

Programinė įranga: Windows XP HE, .NET Framework 2.0 beta 2, ACT

Tarnybinė stotis:

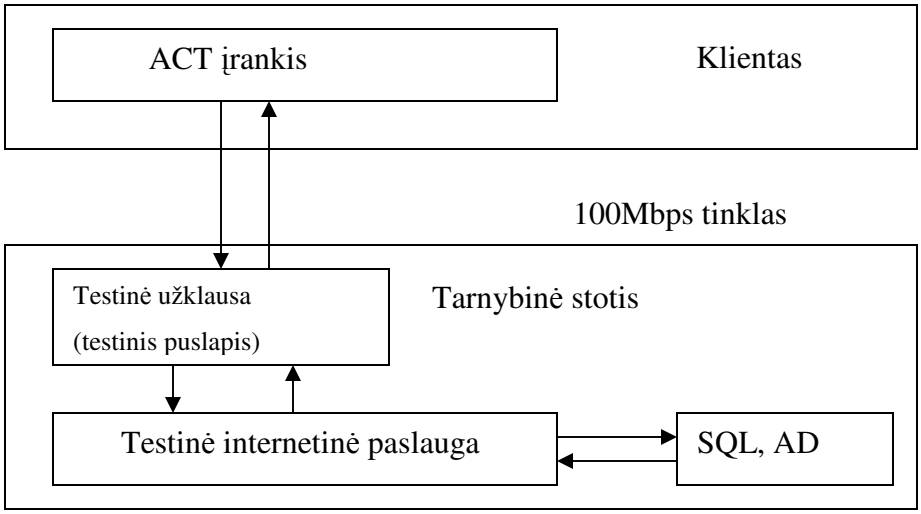
CPU: Xeon 2,4Mhz/512/533

CPU skaičius: 2

HDD: 4x36Gb, SCSII, Raid 10

RAM: 4GB

Programinė įranga: Windows 2000, .NET Framework 2.0 beta 2, MS SQL EE



5 pav. Testavimo stendas

Pralaidumą ir vėlavimą laikome pagrindiniais našumo rodikliais. Pralaidumu vadinsime skaičių, kuris nusako kiek kliento užklausų buvo apdorota duotam duomenų kiekiui per tam tikrą laiko vienetą, paprastai per sekundę. Kartais pikinis pralaidumas gali būti nepriimtinas analizuojant sistemos pritaikomumą, todėl analizuojamas ir vėlinimas, kuris prilyginimas atsakymo laikui ir surandamas iš ACT rezultatų kiekvienam iš paruoštų testų. Tačiau reikia pastebėti, kad šie du parametrai yra skirti skirtingų mechanizmų palyginimui, jie nenusako absoliutaus sistemos našumo.

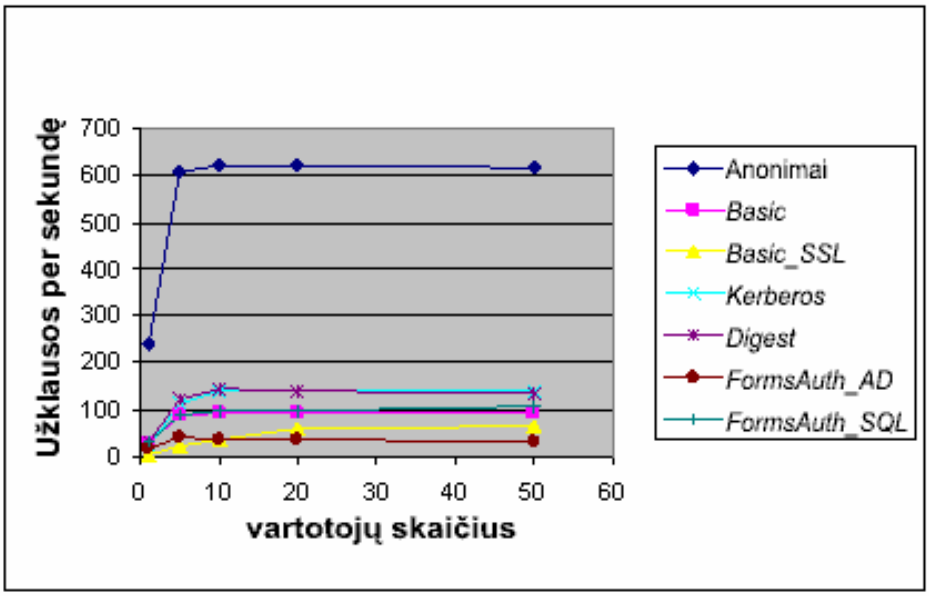
Reikia pastebėti, kad visi metodai buvo tiriami atskirai, tuo tarpu realiose sistemose yra naudojamos kelių metodų kombinacijos.

2.1.2. Kliento identifikavimo našumo bei saugumo tyrimas

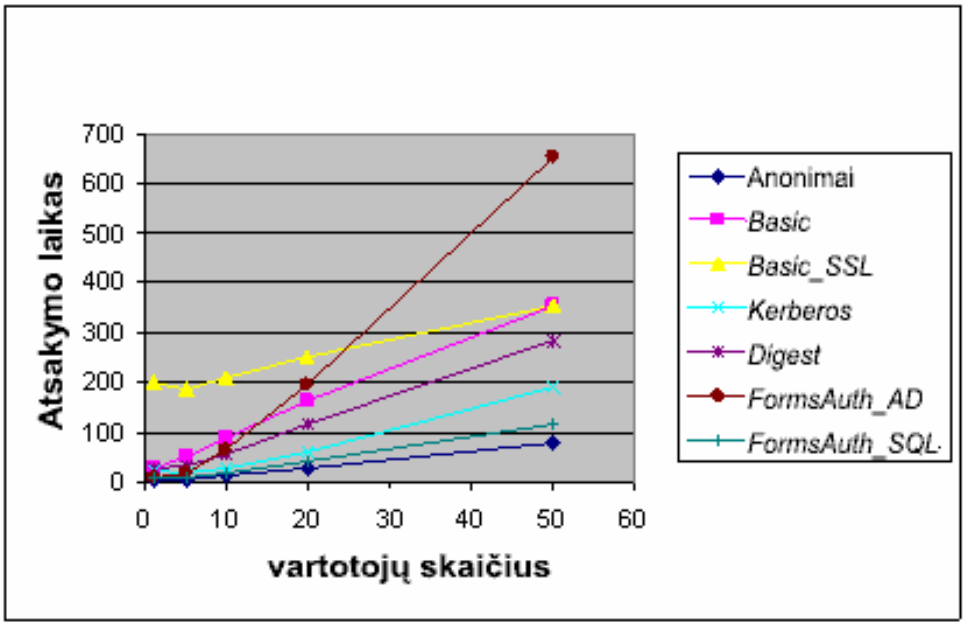
Tarnybinė stotis identifikuoja klientą tokiu būdu: pirmiausiai gaunami vartotojo duomenis (paprastai vartotojo vardą ir slaptažodį), tada šie duomenys yra tikrinami pasitelkiant realizuotą identifikacijos mechanizmą – pavyzdžiui, ieškoma įrašo pagal gautus vartotojo duomenis SQL tipo duomenų bazėje ir jeigu randamas įrašas su numatytomis prisijungimui teisėmis, vartotojas identifikuojamas.

Šiame teste ACT vartotojas siunčia užklausą tarnybinei stočiai, ši paprašo vartotoją duomenų identifikacijai, vartotojas juos pateikia ir tarnybinė stotis autentifikavusi vartotoją atsako trumpu tekstu.

Pastaba: šis testas nemodeliuoja atvejo, kai vartotojas siunčia tarnybinei stočiai bilietą, kuris patvirtina vartotojo anksčiau atliktą autentifikavimą.



6 pav. Identifikavimo metodų pralaidumo palyginimas



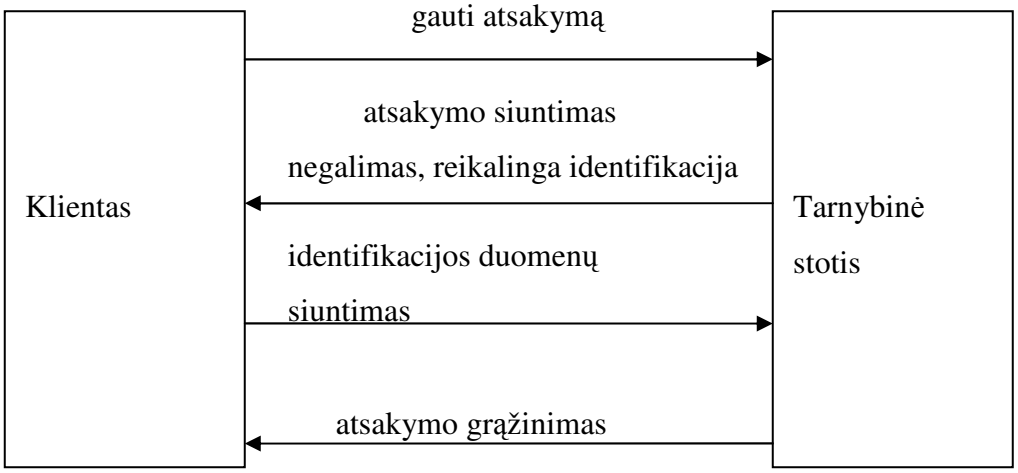
7 pav. Identifikavimo metodų vėlinimo palyginimas

Analizuojant grafikus, nesunku pastebėti, kad naudojant autentifikavimą su SQL duomenų bazėmis, gaunamas neblogas pralaidumo/vėlinimo santykis, be to tai saugesnis būdas už Basic šeimos tipų identifikaciją, todėl nenuostabu, kad tai bene labiausiai paplitęs identifikavimo būdas. Taipogi iš grafiko matyti, kad *Kerberos* taip pat yra

pakankamai pralaidus modelis, tačiau dėl naudojamos kriptografijos jo vėlinimas yra kiek didesnis nei SQL tipo autentifikacijos.

Kaip ir reikėjo tikėtis, anoniminis prisijungimo būdas yra efektyviausias našumo atžvilgiu, tačiau tai mažiausiai saugus metodas.

Visais kitais metodais yra siunčiamos papildomos užklausos, kas ir lemia mažesnį našumą, schematiškai bendru atveju identifikavimo procesas atrodo taip:



8 pav. Identifikavimo proceso schema.

Kaip matyti iš grafiko, Digest ir Kerberos identifikavimo modeliai yra labai panašus savo našumu, tačiau jų veikimas yra skirtingas. Digest identifikavimo atveju, tarnybinė stotis užklausia kliento vartotojo vardo ir slaptažodžio. Slaptažodžiui yra pritaikomas vienakryptis maišos algoritmas(VMA) ir gautas maišos kodas yra naudojamas atsakymo kodavimui, kuris tada yra siunčiamas į tarnybinę stotį ir klientas yra identifikuojamas. Slaptažodis nėra siunčiamas atviru tekstu, dėl to *Digest* yra pranašesnis už *Basic* saugumo požiūriu. Didžiausias Digest modelio trūkumas yra tas, kad jis nėra plačiai pripažįstamas, jį palaiko tik kelios naršyklės bei interneto tarnybinės stotys, kas riboja jo paplitimą, kas ypač svarbu į paslaugas orientuotoje architektūroje.

Naudojant Kerberos, naršyklė bando naudoti srities(*domain*) vartotojo prisijungimo duomenis prisijungimui prie nagrinėjamos internetinės tarnybinės stoties, ir tik nepavykus tai padaryti, yra paprašoma kliento įvesti prisijungimo duomenis konkrečiai tarnybinei stočiai. Prisijungimo duomenys yra tiesiogiai siunčiami į bilietų išdavimo tarnybinę stotį (*Ticket Granting service server*), kuri identifikuoja vartotoją ir išduoda

Kerberos bilietą klientui. Šis bilietas yra laikinas sertifikatas, kuriame yra vartotoją tarnybinėje stotyje identifikuojanti informacija. Klientas paprastai šį bilietą išsisaugoja savo kompiuteryje ir naudoja vėlesniems prisijungimams, kol baigiasi bilieto galiojimo laikas.

Reikia atkreipti dėmesį, kad *ASP.NET Form Authentication* yra lėtesnis modelis nei visi kiti *Windows* sistemoje naudojami identifikavimo modeliai. Taip yra greičiausiai todėl, kad ASP.NET modelyje yra atliekama keletas papildomų nukreipimų prieš vaizduojant kliento reikalaujamą puslapį. *FormsAuth_AD* atveju, sistema programiškai ieško vartotojo *Active Directory* duomenų bazėje. Jeigu pateiktas vartotojo vardas ir slaptažodis randamas minėtoje duomenų bazėje, vartotojas identifikuojamas. Panašiai vykdoma identifikacija ir *FormsAuth_SQL* modelyje, tik šiuo atveju yra ieškoma SQL tipo duomenų bazėje. Slaptažodžiai šiose duomenų bazėse nėra saugomi atviru tekstu, dažniausiai tam yra naudojamas vienakryptės maišos metu gautas kodas, kuriam sugeneruoti naudojamas SHA1 algoritmas. Šis procesas reikalauja papildomų procesoriaus taktų vartotojo slaptažodžio maišos kodui generuoti.

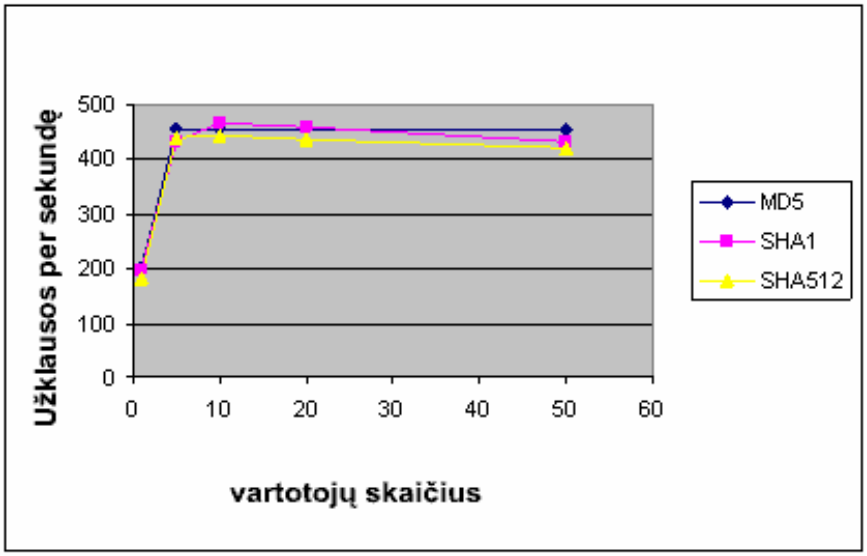
ASP.NET Forms identifikacija yra tokia pat nesaugi, kaip ir Basic, nes slaptažodis yra siunčiamas internetu atviro teksto pavidalu. Nepaisant to, yra skirtumas tarp šių dviejų modelių – *ASP.NET Forms* atveju prisijungimo duomenys yra siunčiami vieną kartą, vėlesniems prisijungimams naudojamas bilietas, tuo tarpu naudojant *Basic* tipo identifikaciją prisijungimo duomenys yra siunčiami kiekvieno prisijungimo metu. Norint padidinti *ASP.NET Forms* saugumą, galima naudoti SSL protokolą, tačiau nuo to nukentės našumas. Bendru atveju *ASP.NET Forms* naudojimas be SSL yra potencialus atakos taikinys.

2.1.3. Maišos algoritmų našumo bei saugumo tyrimas

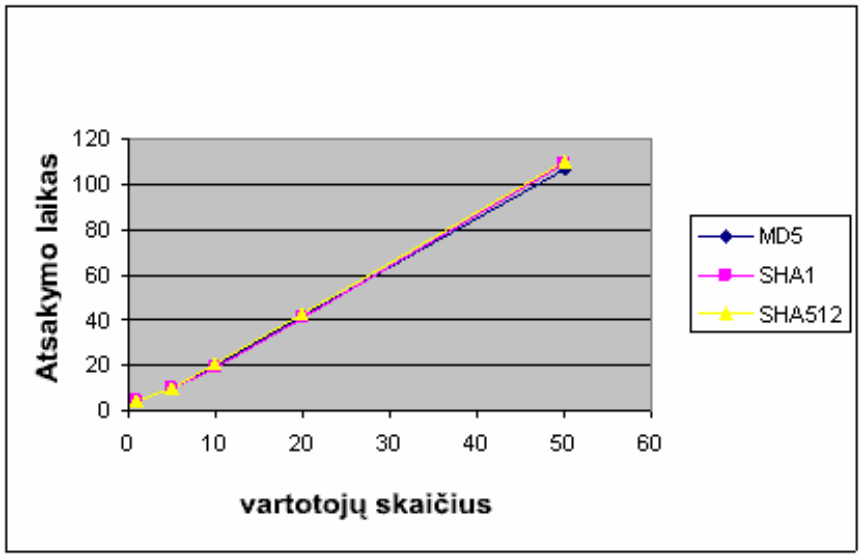
Maišos algoritmai yra skirti duomenims apibūdinti mažu, apibrėžto dydžio kodu. Analizei pasirinkome MD5 ir SHA1 algoritmus, nes tai vieni labiausiai paplitusių (.NET karkase yra realizacijos ir SHA256, SHA384 – vienintelis skirtumas tarp šių realizacijų yra gaunamo maišos kodo dydis, todėl savo testuose nusprendėme naudoti tik dvi realizacijas). Analizei naudojome System.Security.Cryptography esančias klases. MD5 .Net karkase yra tik viena realizacija, pavadinimu MD5CryptoServiceProvider. SHA256,

SHA384 ir SHA512 realizacijos nėra įtrauktos į CryptoAPI, šie algoritmai yra realizuoti tiesiogiai valdomame kode (*managed code*) ir į karkasą įtraukti tik todėl, kad būtų tenkinami AES raktų generavimo reikalavimai, o ne dėl to, kad jie būtų saugesni už SHA1. Egzistuoja įsitikinimas, kad SHA1 yra labiausiai tinkamas duomenų maišai. SHA1 ir SHA512 testuose naudojamos SHA1Managed ir SHA512Managed realizacijos, esančios System.Security.Cryptography.

Testams naudojome trijų dydžių duomenų bylas(4KB, 135KB ir 1MB), kad galėtume pažiūrėti kokią įtaką duomenų kiekis turi našumui.

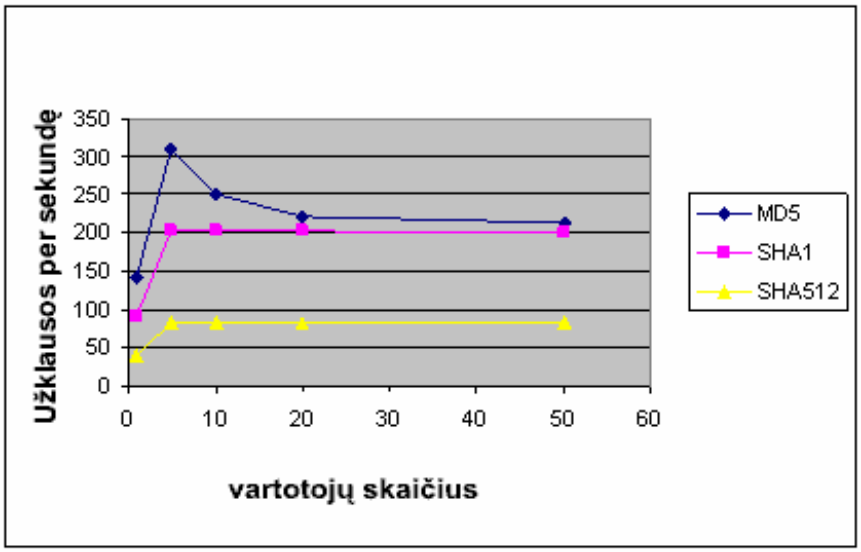


9 pav. Maišos algoritmų pralaidumo palyginimas (4KB duomenų)

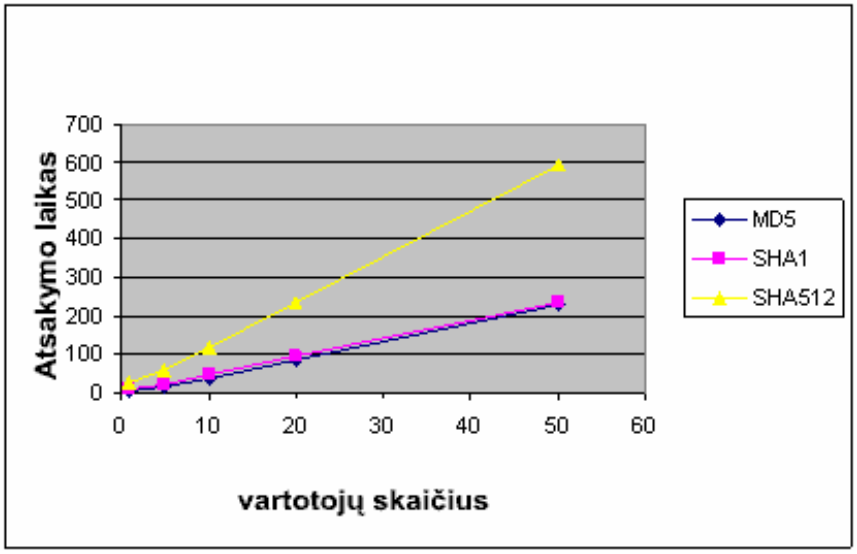


10 pav. Maišos algoritmų vėlinimo palyginimas (4KB duomenų).

Kaip matyti iš grafikų, visi algoritmai yra labai panašūs savo našumu, tik SHA512 šiek tiek atsilieka. MD5 gaunamas maišos kodas yra 128 bitų dydžio, tuo tarpu SHA1 gaunamas kodas yra 160 bitų. SHA512 sugeneruoja 512bitų dydžio maišos kodą, tokiu būdu yra saugiausias iš tiriamų algoritmų, nes didžiausio ilgio kodą yra sunkiausiai nulaužti naudojant perrinkimą, tačiau šie skirtumai jokiu būdu nėra algoritmų silpnos vietos.



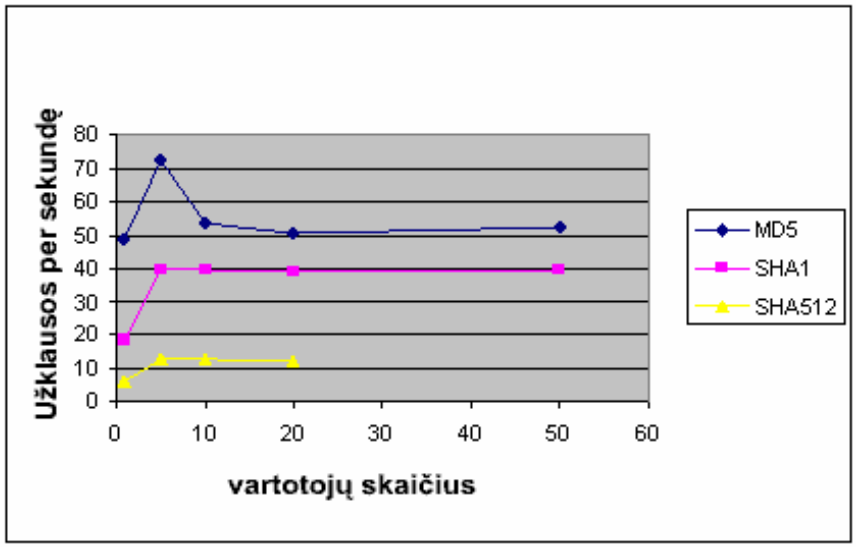
11 pav. Maišos algoritmų pralaidumo palyginimas (135KB duomenų)



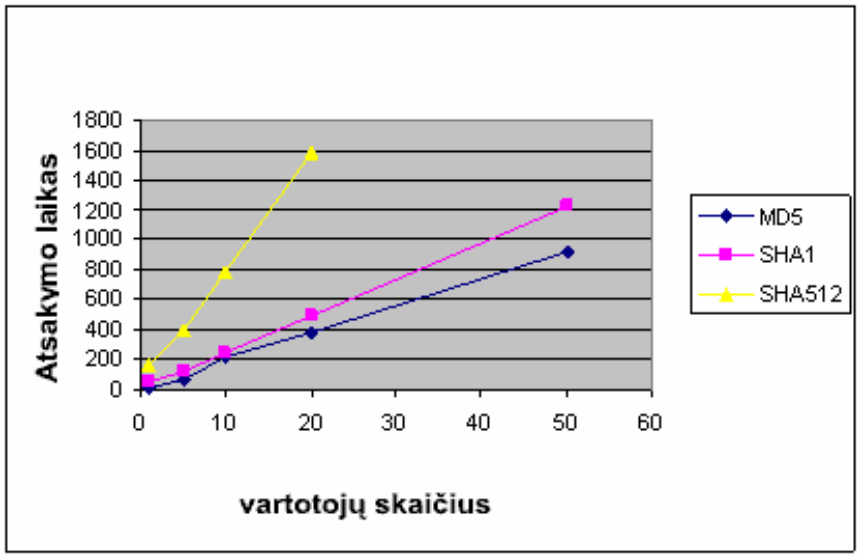
12 pav. Maišos algoritmų vėlinimo palyginimas (135KB duomenų)

Padidinus duomenų bylą, matome, kad našumo skirtumas tarp skirtingų algoritmų ženkliai padidėjo. Esant 5 vartotojams vienu metu, MD5 apie 33% yra greitesnis už

SHA1, nors kodo ilgiai yra pakankamai panašūs. Ir nors šiandien turima aparatūra yra sunku nulaužti MD5, teorinės atakos yra įmanomos. SHA512 našumas taip pat sumažėjo, padidėjus duomenų kiekiui, jis maždaug 55% lėtesnis už SHA1. Derėtų priminti, kad ilgesnis maišos kodas užtikrina geresnį saugumą, tačiau kaip matyti iš grafikų, stipriai veikia našumą.



13 pav. Maišos algoritmų pralaidumo palyginimas (1MB duomenų)



14 pav. Maišos algoritmų vėlinimo palyginimas (1MB duomenų)

Našumo skirtumas dar labiau pastebimas esančiam dar didesniam duomenų kiekiui, esant 1Mb duomenų, skirtumas tarp SHA1 ir MD5 išaugo iki 43% procentų, esant 5 vartotojams, tuo tarpu SHA1 ir SHA512 skirtumas išaugo iki 72%, be to v4linimo grafike

aiškiai išsiskyrė MD5 ir SHA1 skirtumas, kuris nebuvo taip aiškiai pastebimas esant mažesniam duomenų kiekiui.

2.1.4. Simetrinio rakto šifravimo algoritmų našumo ir saugumo palyginimas

Simetrinio rakto algoritmus testavome užkoduodami ir iškoduodami užkoduotus duomenis. Atlikome testus su 4KB, 100KB bei 500KB dydžio duomenimis, norėdami įsitikinti, kaip duomenų apimtis veikia našumą.

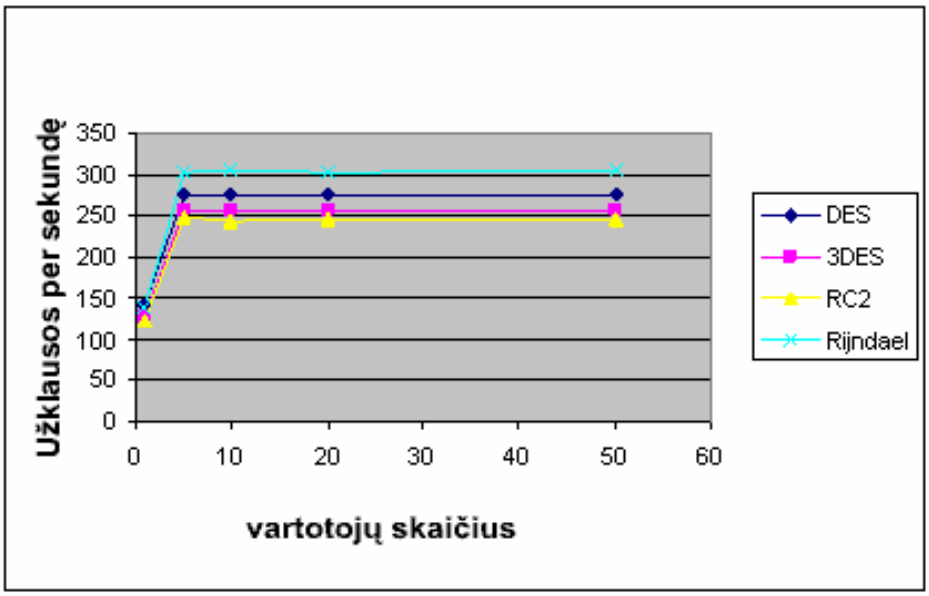
DES, 3DES ir RC2 testams naudojome valdomas apgaubiančias klases (*managed wrappers*), esančias System.Security.Cryptography, kurios apgaubė nevaldomas realizacijas iš CryptoAPI (klasių pavadinimai: DESCryptoServiceProvider, TripleDESCryptoServiceProvider ir RC2CryptoServiceProvider). Vienintelė pilnai valdomoje realizacijoje buvo Rijundeal klasė, esanti System.Security.Cryptography, kuri ir buvo naudojama testuose.

Testuose buvo naudojami šie blokų ir raktų dydžiai:

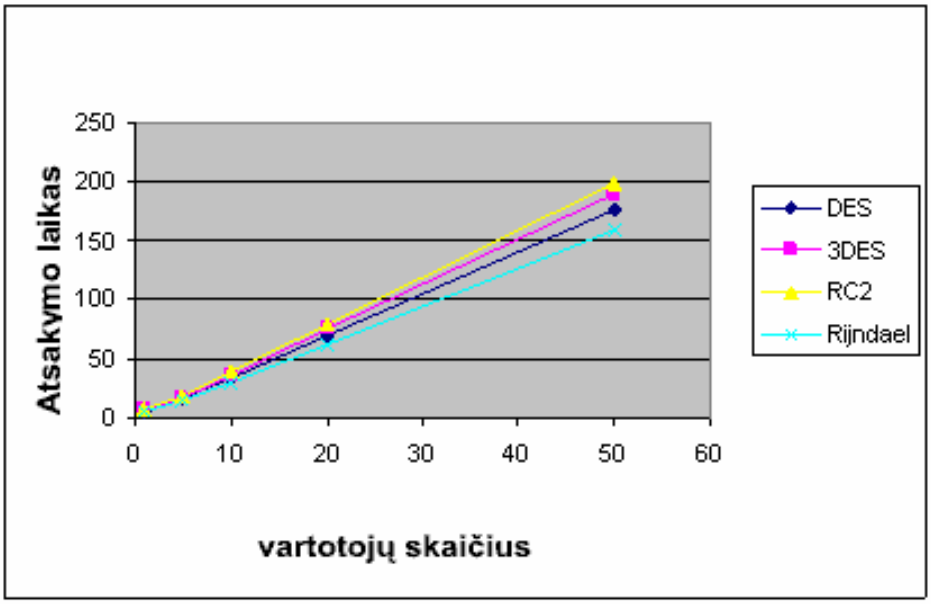
3 lentelė. Tyrime naudotų simetrinių šifravimo algoritmų parametrų reikšmės

Algoritmas	Rakto dydis (bitais)	Bloko dydis (bitais)
DES	64	64
3DES	192	64
RC2	128	64
Rijndael	256	128

3DES, RC2 ir Rijndael galima naudoti ir su kitais raktų ilgiais, tačiau pasirinkome didžiausius leidžiamus raktų ilgius. Kadangi ilgesnis raktas reikalauja daugiau laiko ir paslaugų iš atakuojančiojo, maksimalių rakto ilgių pasirinkimas leido mums išmatuoti kiekvieno algoritmo našumą esant didžiausiam saugumui. Pažymėtina, kad skirtinguose algoritmuose naudojamas vienodo ilgio (pvz. 128 bitų) raktas nereiškia, kad skirtingais algoritmais apsaugotos sistemos bus vienodai saugios.



15 pav. Simetrinių šifravimo algoritmų pralaidumo palyginimas (4KB duomenų)



16 pav. Simetrinių šifravimo algoritmų vėlinimo palyginimas (4KB duomenų)

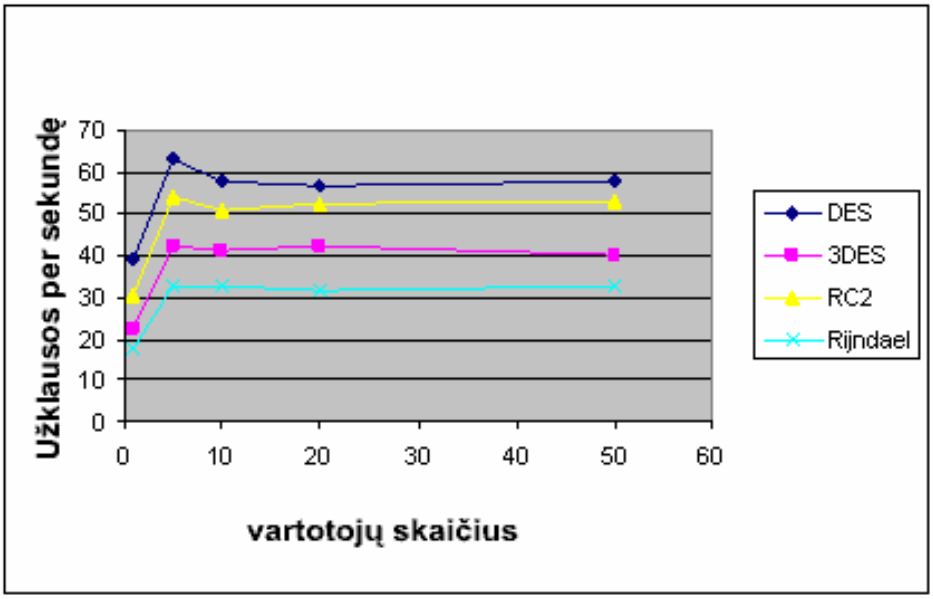
Esant nedideliam duomenų kiekiui, matome, kad Rijndael yra greičiausias iš visų algoritmų. Šis algoritmas pagrįstas kintamu bloko ir rakto ilgiu (galima pasirinkti 128, 192 arba 256 bitų). Etapų, generuojančių kriptotekstą, skaičius taip pat yra kintamas, jis priklauso nuo rakto bei bloko ilgių.

DES koduoja ir iškoduoja duomenis 64 bitų blokuose, naudodamas 64 bitų raktą. Kiekvienas duomenų blokas yra 16 kartų apdorojamas, kad būtų gautas kriptotekstas.

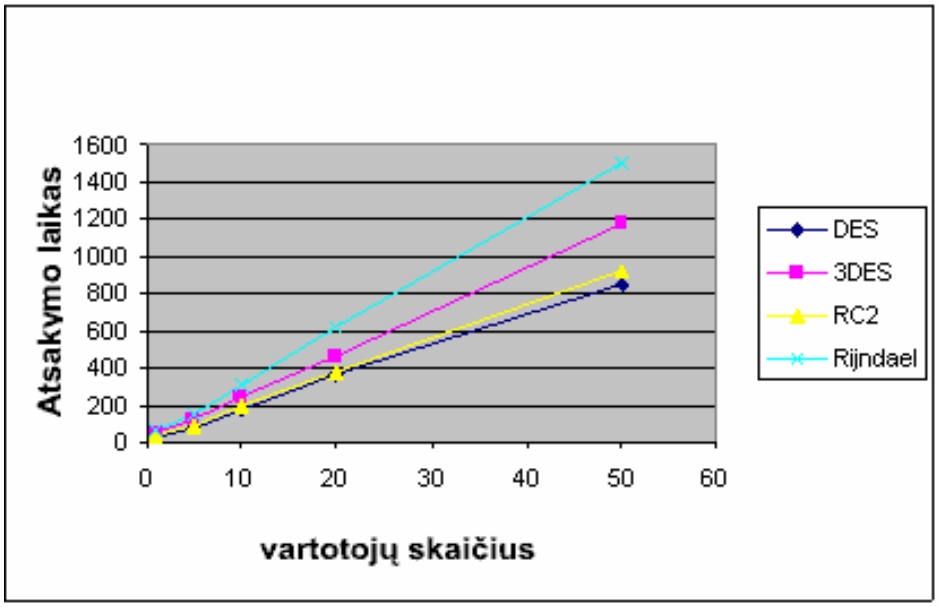
Nors šis algoritmas ir greitesnis už 3DES ir RC2, pernelyg trumpas raktas naudojamas kodavimui, todėl naudojantis perrinkimo būdu turint greitus kompiuterius įmanoma nulaužti slaptažodį ir iššifruoti duomenis.

3DES buvo sukurtas norint padidinti DES saugumą. Šio algoritmo veikimo principas yra pagrįstas trijų skirtingų raktų panaudojimu, koduojant tris kartus ta patį bloką (to paties rakto naudojimas šiame algoritme neduoda jokios naudos). Kiekvienas blokas yra užkoduojamas DES principu naudojant pirmą raktą, tada iškoduojamas antru raktu ir galiausiai užkoduojamas vėl trečiu raktu.

RC2, sprendžiant iš grafikų, yra pats lėčiausias algoritmas, kai koduojamas nedidelis duomenų kiekis. Taip yra iš dalies dėl to, kad šis algoritmas pradžioje susidaro daug skaičiavimų reikalaujančią nuo rakto priklausančią reikšmių lentelę, šios lentelės sudarymas yra neefektyvus, kai koduojami maži duomenų kiekiai. RC2 yra kintamo rakto dydžio kodavimo algoritmas, sukurtas kaip alternatyva DES.



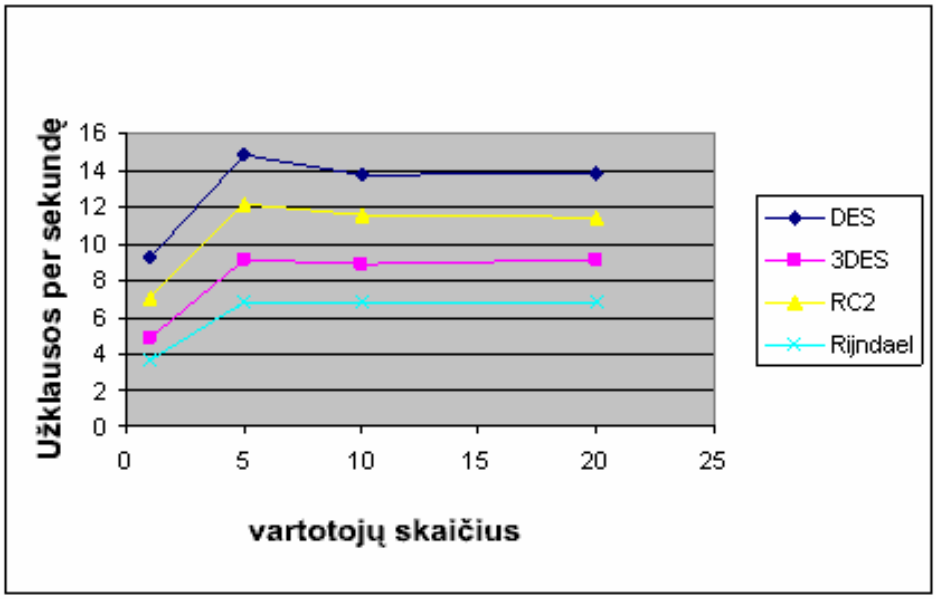
17 pav. Simetrinių šifravimo algoritmų pralaidumo palyginimas (100KB duomenų)



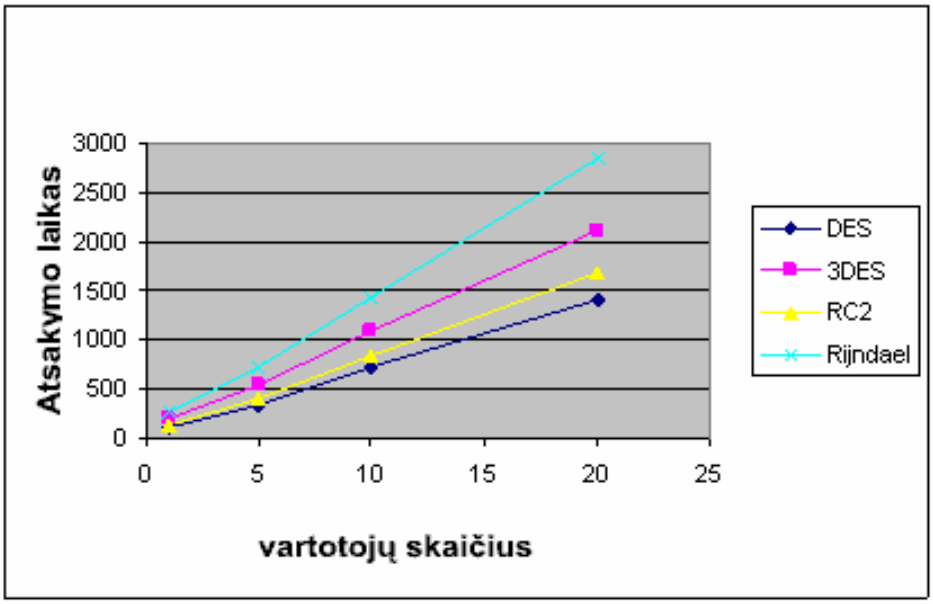
18 pav. Simetrinių šifravimo algoritmų vėlinimo palyginimas (100KB duomenų)

Didindami koduojamų ir iškoduojamų duomenų kiekį, matome visiškai kitą vaizdą, nei prieš tai buvusiame teste. DES yra greičiausias, po jo seka RC2, kuris yra apie 20% greitesnis už 3DES. Šiame teste RC2 algoritmo kuriama reikšmių lentelė, priešingai nei prieš tai buvusiame teste, pasiteisina. Rijndael šiuo atveju buvo lėčiausias, apie 25% lėtesnis nei 3DES. Taip pat reikėtų nepamiršti, kad Rijndael mes naudojame patį ilgiausią raktą, todėl juo koduojami duomenys yra saugesni nei koduojami kitais nagrinėjamais algoritmais (spaudoje buvo pasirodę pranešimai apie galimas atakas prieš Rijndael algoritmu koduojamus duomenis, kai gali būti naudojami metodai geresni už perrinkimą). Taipogi, mes naudojome 192 bitų raktą 3DES atveju, todėl šiuo atveju 3DES yra saugesnis už DES ir RC2 kodavimą.

Dar kartą reikėtų pabrėžti, kad tokio paties ilgio rakto naudojimas nereiškia, kad skirtingais algoritmais koduojami duomenys bus vienodai saugūs. Skirtingi algoritmai turi skirtingas charakteristikas ir nesuteikia to paties saugumo esant vienodo ilgio raktams.



19 pav. Simetrinių šifravimo algoritmų pralaidumo palyginimas (500KB duomenų)



20 pav. Simetrinių šifravimo algoritmų vėlinimo palyginimas (500KB duomenų)

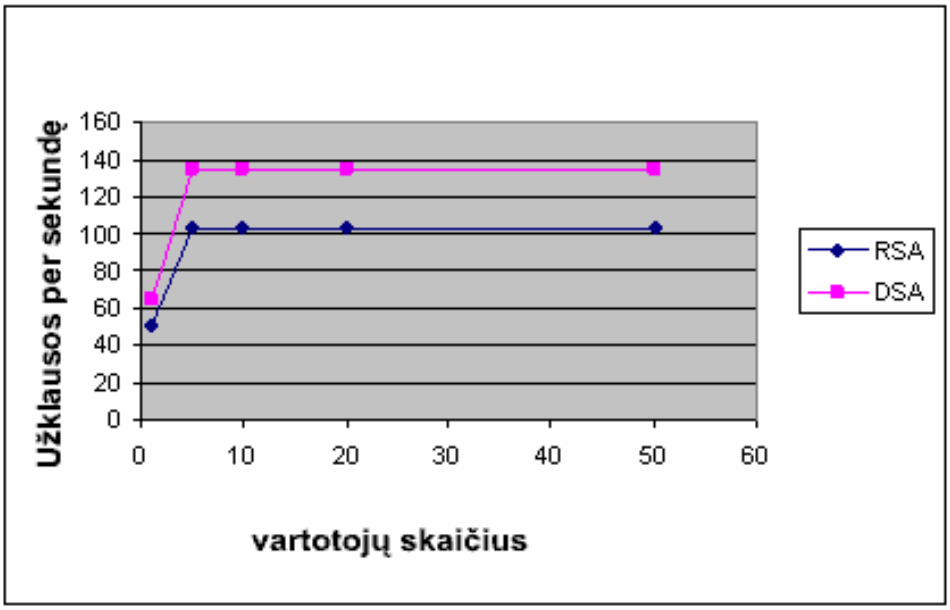
Gautieji grafikai išlaiko tendencijas, kurias matėme grafikuose, kai tyrėme 100KB duomenų.

2.1.5. Asimetrinių šifravimo algoritmų našumo ir saugumo palyginimas

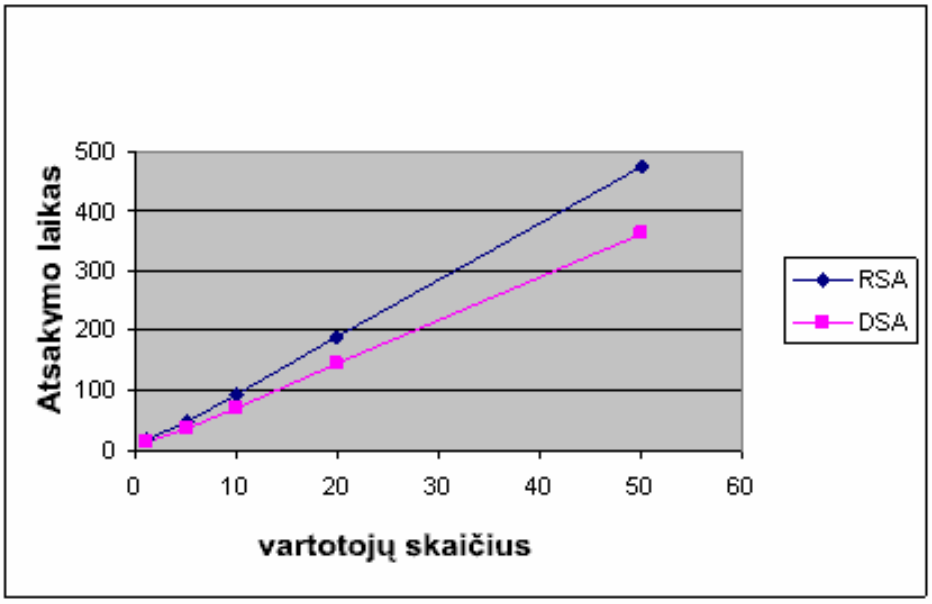
Šifravimas naudojant asimetrinio rakto algoritmus yra labai lėtas, tai ypač pastebima, jeigu šifruojamas didelis duomenų kiekis, nes paprasčiausiai jie buvo skirti kritinei informacijai šifruoti, dideli duomenų kiekiai turėtų būti šifruojami naudojant simetrinio rakto algoritmus. Asimetriniai algoritmai gali būti naudojami raktų apsikeitimui.

Du pagrindiniai asimetriniai algoritmai yra RSA ir DSA. RSA gali būti naudojamas ir šifravimui ir parašo generavimui, tuo tarpu DSA gali būti naudojamas tik parašo generavimui. Mes palyginome, kaip RSA ir DSA algoritmai greitai gali sugeneruoti elektroninį parašą ir kaip greitai galima šį parašą patikrinti. Tam naudojome RSACryptoServiceProvider ir DSACryptoServiceProvider klases, kurios yra apgaubiančios nevaldomas (*unmanaged*) RSA ir DSA realizacijas, esančias CryptoAPI.

RSA buvo naudojamas 1024 bitų raktas, DSA – 512 bitų raktas.



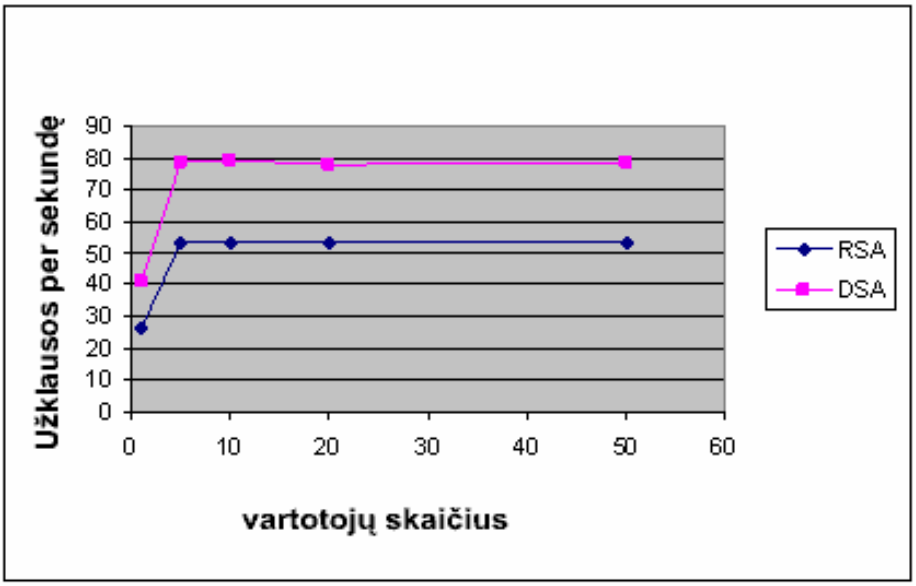
21 pav. Asimetrinių šifravimo algoritmų pralaidumo palyginimas (100KB duomenų)



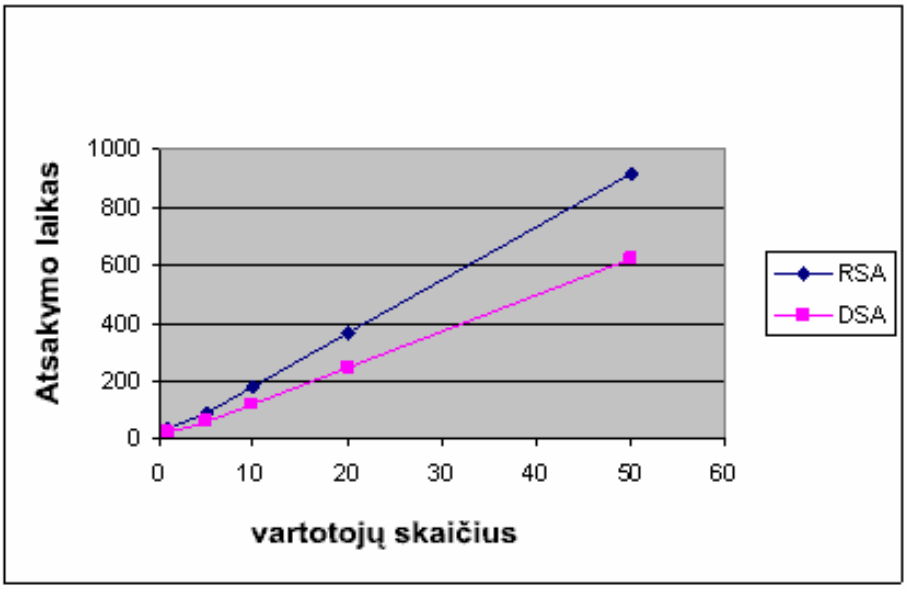
22 pav. Asimetrinių šifravimo algoritmų vėlinimo palyginimas (100KB duomenų)

Kaip matyti iš grafikų, DSA yra apie 29% greitesnis už RSA generuojant elektroninį parašą. RSA elektroninio parašo generavimo metu privatus raktas naudojamas žinutės santraukos šifravimui. Nors DSA ir panašus į RSA, tačiau DSA nešifruoja žinutės santraukų su privačiu raktu bei ne dešifruoja jų su viešu raktu. Vietoj to, DSA naudoja specialias matematines funkcijas, kurių pagalba sugeneruojamas skaitmeninis parašas, susidedantis iš dviejų 160 bitų skaičių, kurie yra gaunami iš žinutės santraukos ir privataus rakto.

Atlikome testą ir su 500KB duomenų, siekdami išsiaiškinti, ar nepasikeis santykinis našumas šių dviejų algoritmų.



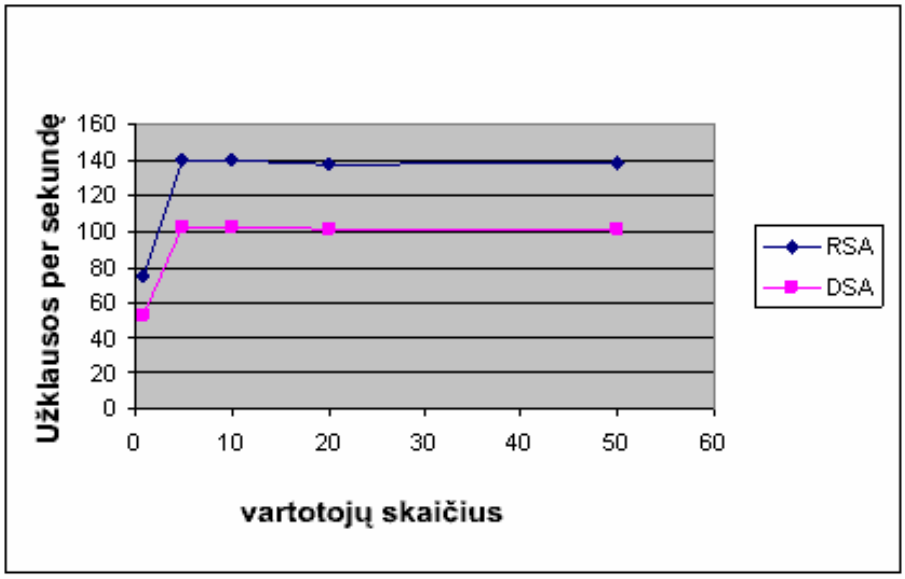
23 pav. Asimetrinių šifravimo algoritmų pralaidumo palyginimas (500KB duomenų)



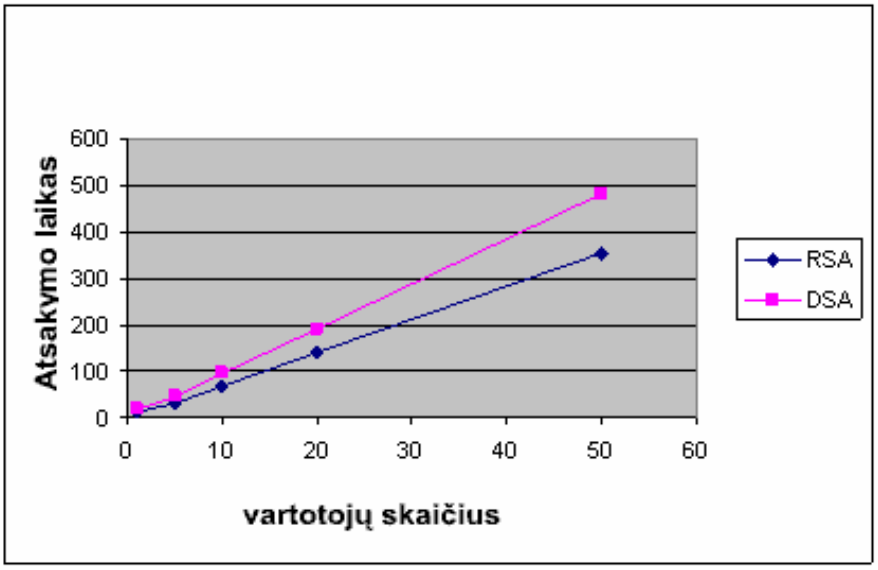
24 pav. Asimetrinių šifravimo algoritmų vėlinimo palyginimas (500KB duomenų)

Kaip matyti iš grafikų, situacija išliko panaši kaip ir prieš tai vykdytame teste, DSA yra gerokai našesnis ir esant didesniam duomenų kiekiui.

Toliau atlikome skaitmeninio parašo tikrinimą, norėdami patikrinti, kuris iš algoritmų yra efektyvesnis.



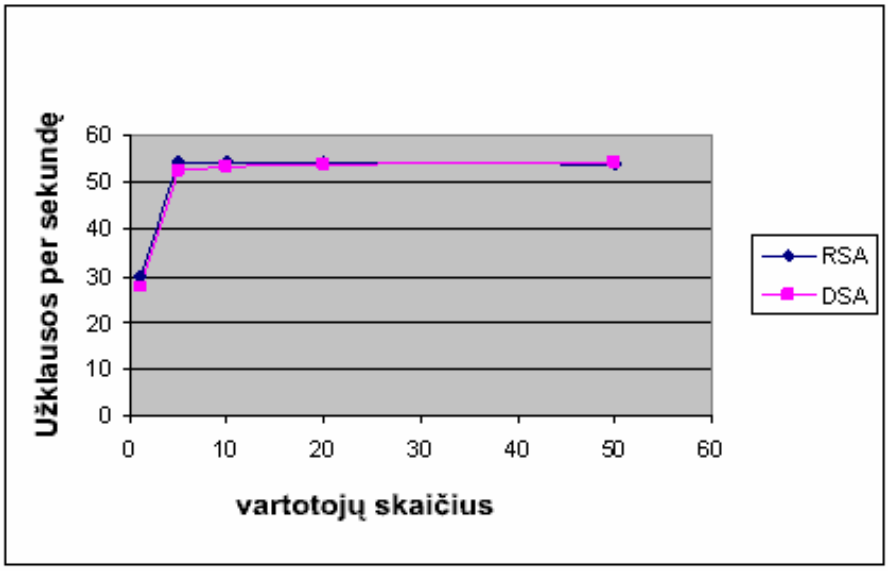
25 pav. Asimetrinių šifravimo algoritmų parašo tikrinimo pralaidumo palyginimas (100KB duomenų)



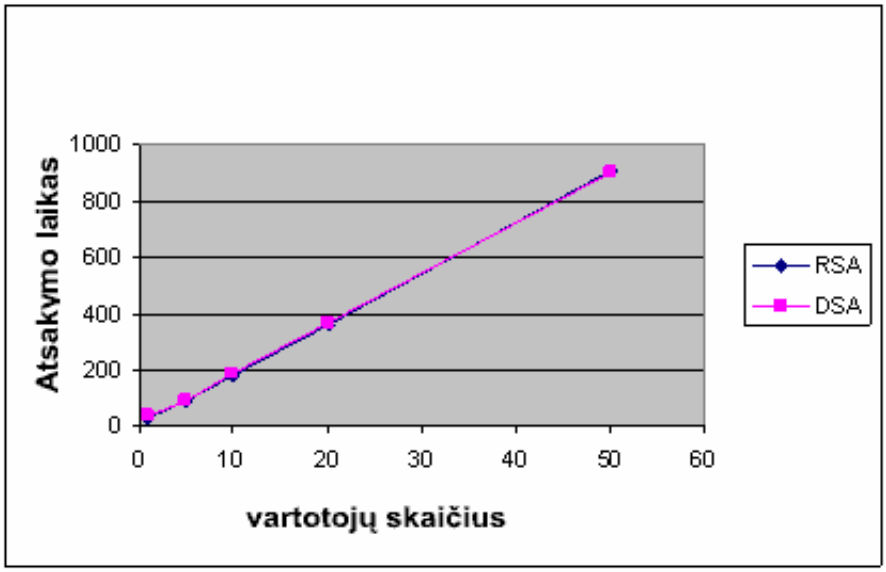
26 pav. Asimetrinių šifravimo algoritmų parašo tikrinimo vėlinimo palyginimas (100KB duomenų)

Kaip matyti, tikrinant parašus RSA su DSA apsikeitė vietomis, RSA parašo tikrinimą atliko greičiau. RSA algoritmas naudoja viešą raktą parašo tikrinimui. Algoritmas generuoja naują žinutės santrauką iš gautų duomenų, iššifruoja originalią žinutę su viešu raktu ir lygina sugeneruotą santrauką su iššifruotąja. Jeigu santraukos sutampa, žinutės vientisumas ir sutapimas su originalu yra patvirtinamas. Žinutę

pasirašiusiojo tapatybė patvirtinama ir todėl, kad viešas raktas gali iššifruoti tik tuos duomenis, kurie buvo užšifruoti atitinkamu privačiu raktu.



27 pav. Asimetrinių šifravimo algoritmų parašo tikrinimo pralaidumo palyginimas (500KB duomenų)



28 pav. Asimetrinių šifravimo algoritmų parašo tikrinimo vėlinimo palyginimas (500KB duomenų)

Tikrinant parašą didesniam duomenų kiekiui, gauti įdomūs rezultatai – RSA ir DSA algoritmų našumas praktiškai sutampa.

2.2. Naujos ASP.NET 2.0 priemonės, susijusios su saugumu

2.2.1. C# 2.0 kalbos papildymai

2.2.1.1. Gentys (*generics*)

C# yra saugi duomenų tipų atžvilgiu (*type-safe*). Saugumas duomenų tipų atžvilgiu reiškia, kad potencialios duomenų priskyrimo klaidos, padarytos programuotojo (kai vienam duomenų tipui priklausančiam kintamajam priskiriama kitam duomenų tipui priklausančio kintamojo reikšmė) bus randamos kompiliuojant programos kodą, taigi prieš vykdant programą. Tai ypač svarbu internete veikiančioms programoms, nes įsilaužėlis aptikęs tokią klaidą gali ja pasinaudoti.

1.1.NET versijoje (2003) yra atvejai, kad c# kompiliatoriaus duomenų tipo saugumo kontrolė gali nepastebėti neteisingo duomenų priskyrimo. Vienas iš tokių atvejų yra, kai naudojamos kolekcijos. Kolekcijos yra skirtos Object tipo objektų kolekcionavimui. Kadangi visi duomenų tipai C# kalboje yra išveldimi iš Object klasės, tai vienoje kolekcijoje gali būti kelių skirtingų tipų kintamieji, kas reiškia, kad kolekcijos atveju nėra duomenų tipo tikrinimo. Dar didesnė problema yra tai, kad kiekvieną kartą iš kolekcijos paimant objektą, jam reikia priskirti (*cast*) teisingą duomenų tipą, kas sumažina programos našumą ir pablogina kodo skaitomumą. Jeigu į kolekciją yra įtraukiama reikšmės tipas (pvz. integer), jis yra patalpinamas į dėžę(*boxing*) ir išimamas iš jos, kai kintamasis paimamas iš kolekcijos (reikia du kartus pritaikyti teisingą tipą).

Šioms problemoms spręsti, C# 2.0 specifikacijoje yra įvedama nauja sąvoka – tai specifinio duomenų tipo kolekcijos. Šios naujos programinės priemonės pagalba yra suteikiamas duomenų tipų saugumas kolekcijoms ir nebereikia rūpintis duomenų tipo parinkimu, imant elementą iš kolekcijos.

Genties pavyzdys:

```
public class Sarasas <T>
        {
                T[] elementai;
                int skaitliukas;
```

```
        public void Prideti (T elementas)
        {
                //……
        }
        public void Isimti (T elementas)
        {
                //……
        }
}
Sarasas<Pacientas> ps = new Sarasas<Pacientas>();
```

Naujajame .NET karkase yra ir klasės, susijusios su gentimis. Šios klasės padeda saugiau ir efektyviau atlikti dažnai programavime naudojamus veiksmus su duomenimis.

4 lentelė. Klasės, susijusios su gentimis

Comparer<T>	Palygina du to paties genties tipo objektus
Dictionary<K, V>	Kolekcija, sudaryta iš rakto/reikšmės porų
Dictionary<K, V>.KeyCollection	Grąžina raktų kolekciją iš Dictionary<K, V>
Dictionary<K, V>.ValueCollection	Grąžina reikšmių kolekciją iš Dictionary<K, V>
Ienumerable<T>	Kolekcija, kurią galima pereiti naudojant foreach operatorių
LinkedList<T>	Dvikryptis sąrašas
LinkedListNode<T>	Dvikrypčio sąrašo elemento klasė
List<T>	Dinaminis sąrašas
Queue<T>	Eilės tipo sąrašas
ReadOnlyCollection<T>	Tik skaitymui skirta kolekcija
SortedDictionary<K, V>	Kolekcija, kurioje rakto/reikšmės poros surūšiuotos naudojant IComparer<T>.
Stack<T>	Stekas – kolekcija, pagrįsta principu „paskutinis įėjęs, pirmas išeina“

2.2.1.2. Tušti tipai (*Nullable types*)

Jeigu pažvelgsime į .NET karkaso pirmąją versiją pamatysime, kad pagrindinis skirtumas tarp nuorodos(*reference*) kintamojo tipo ir reikšmės(*value*) kintamojo tipo yra

tai, kad pirmajam tipui galime priskirti null reikšmę, t.y., jeigu mes norime parodyti, kad kintamasis neturi reikšmės, mes jam priskiriame null reikšmę. Jeigu mes bandysime null priskyrinėti reikšmės tipo kintamajam, bus priskirta konkretaus tipo reikšmė pagal nutylėjimą (pvz. Integer tipui tai bus 0, o tai irgi yra viena iš galimų integer reikšmių, todėl toks kintamasis turi reikšmę). Antroji .Net karkaso versija leidžia priskirti tuščia reikšmę ir reikšmės kintamiesiems.

Šio naujo papildymo nauda akivaizdi. Sakykime turime pacientų duomenų bazę, kurioje yra lentelė pacientai. Šioje lentelėje yra duomenys apie ligonių kasų skiriamas išmokas už invalidumą. Jeigu pacientas yra miręs, nesant tuščio tipo, į sumos laukelį bus įrašomi nuliai, o tai gali būti interpretuojama klaidingai (pvz., kad pacientas vis dar gyvas, tačiau jam nepervedama invalidumo pašalpa). Iš dalies šią problemą sprendė System.Data.SqlTypes, bet šioje bibliotekoje yra daug su SQL susijusių dalykų, kas mažina programinės įrangos plečiamumą, našumą ir pan. Kai kuriais atvejais programuotojas laikydavo vieną iš galimų reikšmių tuščios reikšmės vaizdavimui, pavyzdžiui -1, kalbant apie integer tipą. Tačiau bendro susitarimo nėra (jo ir būti negali, pavyzdžiui boolean tipas turi tik dvi galimas reikšmes). Tušti tipai yra geriausias sprendimas šio tipo problemoms spręsti.

2.2.2. Vartotojo sąsajos komponentai

2.2.2.1. Tiekėjo modelis (*Provider model*)

Didžiausia ASP.NET 2.0 karkaso naujovė yra su saugumu susiję vartotojo sąsajos elementai. Daugelis kasdien naudojamų saugumo komponentų yra standartizuoti ir realizuoti pačiame karkase, t.y. programuotojui reikia daug mažiau rašyti pačiam, jis gali naudotis didelės ir patyrusios programuotojų bei testuotojų komandos sukurtais komponentais programinės įrangos saugumo užtikrinimui.

ASP.NET 2.0 karkaso saugumo posistemė remiasi tiekėjo modeliu. Tiekėjo modelis duoda standartinį mechanizmą naujų komponentų, kurie realizuoja įvairias paslaugas, prijungimui. ASP.NET 2.0 karkasas saugumo posistemėje naudoja dviejų tipų tiekėjus: narystės tiekėją (*membership provider*) bei vaidmenų tiekėją(*role provider*). Narystės tiekėjas saugo informaciją apie vartotojų vardus ir slaptažodžius, tuo tarpu vaidmenų tiekėjas naudojamas vartotojo teisių ir privilegijų saugojimui.

ASP.NET 2.0 yra pateikiamas su dviem tiekėjais: AccessMembershipProvider ir SqlMembershipProvider. Pagal nutylėjimą, narystės tiekėjas yra AccessMembershipProvider (galutinėje versijoje jis bus pakeistas), naudojant šį tiekėją vartotojų duomenys yra saugomi Microsoft Access duomenų bazėje. Taip pat yra galimybė susikurti ir naudoti savo narystės tiekėją, jeigu norima, pavyzdžiui, saugoti vartotojų informaciją XML byloje, FoxPro duomenų bazėje ar tiesiog nutolusioje internetinėje paslaugoje. Norint sukurti savo narystės tiekėją, reikia realizuoti visus metodus ir žymes MembershipProvider klasėje. Narystės tiekėjas yra aprašomas web.config byloje, todėl norint jį pakeisti kitu, užtenka pakeisti tiekėjo pavadinimą vienoje byloje:

```
<configuration>
        <system.web>
                <membership defaultProvider="NewProvider" />
        </system.web>
</configuration
```

Tiekėją galima pakeisti ir netaisant programinio kodo, jį pakeisti naudojantis kartu su ASP.NET 2.0 pateikiamais įrankiais – MMC konsolės įskiepiu arba svetainių administravimo įrankiu (*Web Site Administration tool*).

2.2.2.2. Prisijungimo (*Login*) ir tikrinimo (*Validation*) komponentai

ASP.NET karkasas yra pateikiamas su aibe naujų, su saugumu susijusių, vartotojo aplinkos komponentų, kuriuos būtų galima suskirstyti į dvi grupes: tai prisijungimo komponentai bei įvesties tikrinimo komponentai. Naudojant šiuos komponentus galima apseiti praktiškai be papildomo programinio kodo ir naudotis Microsoft laboratorijose nušlifuotais bei standartizuotais komponentais.

5 lentelė. Vartotojo sąsajos prisijungimo komponentai

Komponento pavadinimas	Paskirtis
Login	Vartotojo prisijungimo komponentas
LoginView	Skirtingų šablonų rodymui vartotojams, turintiems skirtingus vaidmenis

CreateUserWizard	Naujų vartotojų kūrimo forma
PasswordRecovery	Vartotojo slaptažodžio atkūrimo forma
ChangePassword	Vartotojo slaptažodžio keitimo forma

Prisijungimo komponentai išnaudoja visus tiekėjo modelio privalumus – tai permatomas ryšys su vartotojo duomenų baze, programuotojui nereikia jaudintis dėl komunikavimo su konkrečia duomenų baze subtilybių.

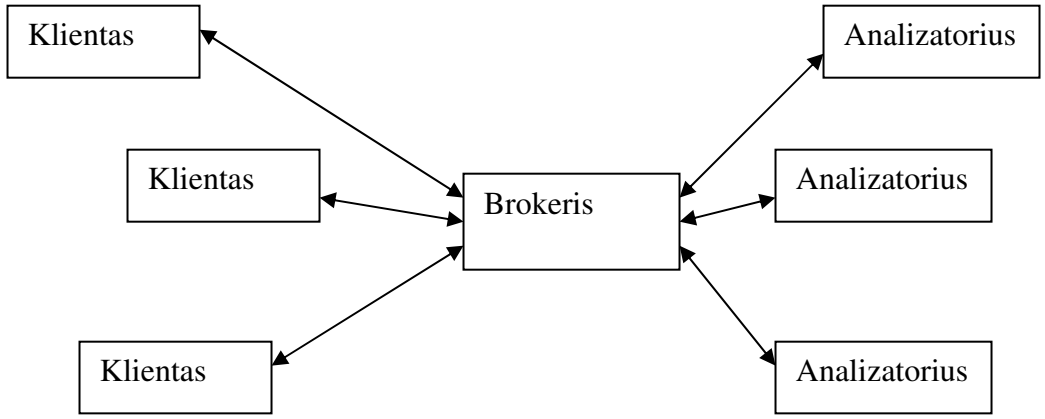
Įvesties tikrinimo komponentai padeda programuotojui kurti saugias duomenų įvedimo formas. Esminis privalumas šių komponentų yra tai, kad jie tikrina įvestus duomenis tiek kliento pusėje, tiek ir tarnybinėje stotyje, prieš pateikiant juos apdorojimui. Kliento pusėje duomenys tikrinami pasitelkiant Java Script priemonės, tačiau apsidraudžiant nuo piktavališko šių priemonių atjungimo kliento pusėje, duomenys yra dar kartą patikrinami tarnybinėje stotyje.

6 lentelė. Vartotojo sąsajos įvesties tikrinimo komponentai

Komponento pavadinimas	Paskirtis
RequiredFieldValidator	Tikrinti, ar užpildytas būtinas laukas
RangeValidator	Dydžio patikrinimas
RegularExpressValidator	Įvestų duomenų tikrinimas naudojant reguliariąsias išraiškas
CompareValidator	Tikrinama, ar įvesta reikšmė sutampa su kita įvesta reikšme
CustomValidator	Vartotojo apibrėžtas tikrinimo komponentas
ValidationSummary	Įvestų duomenų tikrinimo suvestinė

3. Kardiologijos paslaugų modelio programinė realizacija

Kardiologinių paslaugų modelis buvo realizuotas trijų lygių sistema:



29 pav. Kardiologinių paslaugų modelis

Klientu šiuo atveju gali būti nebūtinai galutinis vartotojas, tai gali būti kitos sistemos brokeris, internetinė paslauga ir pan. Brokeris apdoroja užklausas, ateinančias iš klientų ir šias užklausas paskirsto esantiems analizatoriams. Mūsų modelyje analizatoriai – tai tarnybinės stotys, kuriose yra įdiegta MatLab paketo programinė įranga. Šiose tarnybinėse stotyse yra MatLab paketui suprantama kalba aprašyti kardiogramų analizės algoritmai, kurie apgaubiančios (*wrapper*) klasės pagalba jungiasi prie brokerio. Klientas bei analizatoriai jungiasi per internetinės paslaugos standartizuotą XML sąsają. Analizatorių apgaubiančiosios klasės bei brokeris yra konkrečiu atveju yra realizuoti Visual Studio 2003 priemonėmis, klientas – naudojant Visual Studio 2005 priemones. Kadangi brokeris naudoja XML sąsają, metodų aprašymai yra naudojant WSDL, tai prie jungtis gali bet kokios platformos elementai (klientai bei analizatoriai). Pacientų bei analizatorių duomenų bazės yra saugomos brokerio tarnybinėje stotyje, prie šių duomenų bazių saugumo sumetimais priėjimas iš išorės nėra įmanomas, klientų bei analizatorių identifikacija bei autorizacija yra vykdoma brokerio metodų pagalba. Dėl šio sprendimo buvo realizuotos specialios klasės. Šios klasės realizuotos remiantis tiekėjo modeliu, kad būtų galima išnaudoti narystės modulio, esančio Visual Studio 2005, privalumus. Duomenų srautus pagal apimtį suskirstėme į dvi kategorijas: vienai priklauso kardiogramos bei analizės rezultatai, kitai priklauso pagalbiniai duomenys (paciento

įrašai, analizių, kardiogramų sąrašai ir kt.). Apsikeitimas pagalbiniais duomenimis vyksta naudojant SSL protokolą, duomenys yra papildomai šifruojami, nes sistema gali būti išplėsta nuosekliai, t.y. gali būti įterpti tarpiniai taškai, todėl SSL neužtikrina absoliutaus sprendimo mūsų atveju. Kardiogramos yra perduodamos įprastiniu, neapsaugotu kanalu, tačiau yra šifruojami simetriniais algoritmais. Toks sprendimas buvo pasirinktas dėl kelių priežasčių – visų pirma dėl to, kad SSL kanalu perduodant didelius duomenų srautus labai apkraunama tarnybinė stotis (bei tarpinės tarnybinės stotys) - brokeris pats neapdoroja duomenų, juos tik maršrutizuoja, todėl SSL protokolo daromas iššifravimas brokeryje būtų brokerio resursų bereikalingas švaistymas.

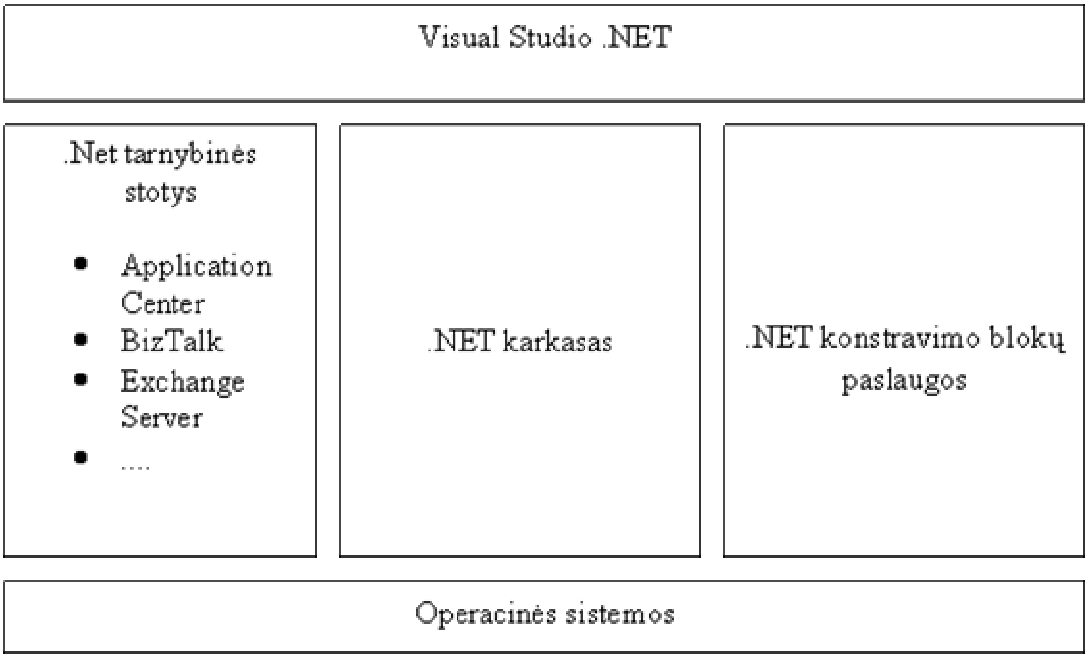
Kliento dalyje buvo panaudota didžioji dalis priemonių, kurias tyrėme bei analizavome savo darbe – pradedant naujomis C# galimybėmis (gentys bei tušti tipai), vartotojo sąsajos elementais, narystės bei vaidmenų moduliais ir baigiant naršyklėje realizuotu svetainės administravimo įrankiu.

4. IŠVADOS

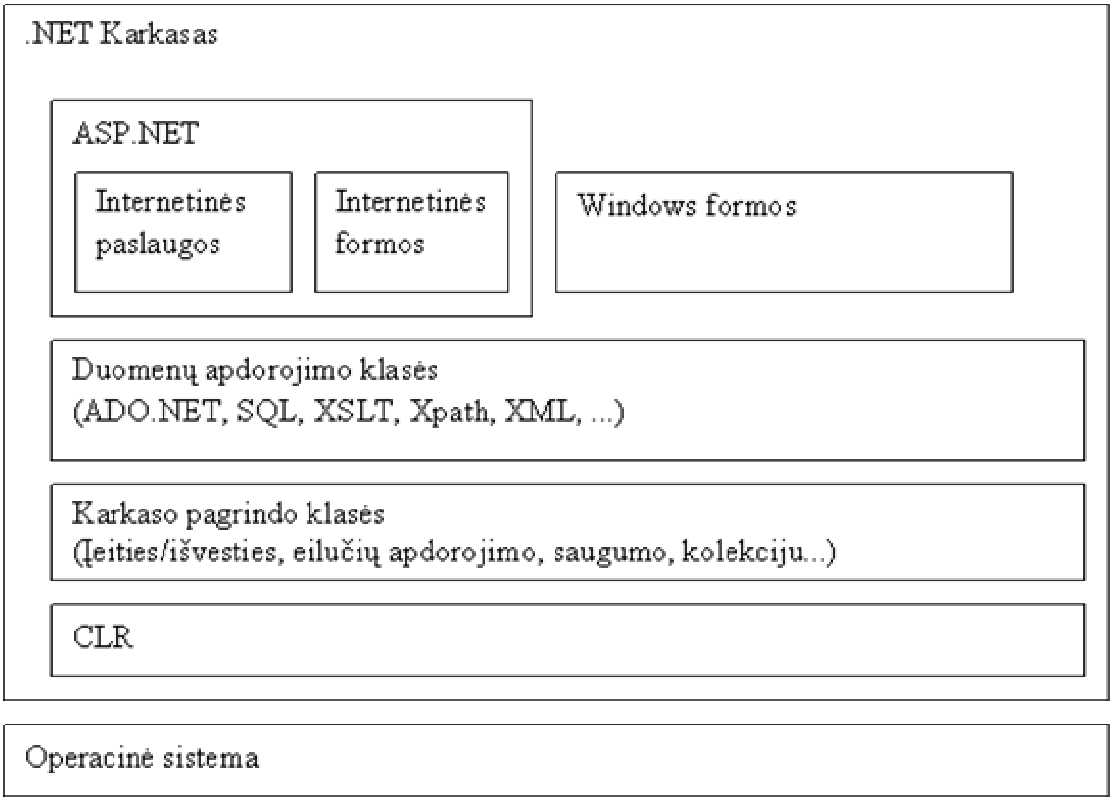
- Saugumo metodų našumo tyrimo metu nustatėme, kad didžiausią našumą galima pasiekti esančius saugumo modelius tarpusavyje kombinuojant, atsižvelgiant į duomenų apimtis bei siunčiamos informacijos kritiškumą.
- Mūsų sukurtoje trijų lygių sistemoje SSL protokolas neužtikrina absoliutaus saugumo, nes duomenys yra iššifruojami brokeryje, kuris šiuo atveju yra tarpinis taškas ir jame duomenų iššifravimas yra tik papildomas sistemos apkrovimas bei potenciali silpna vieta saugumo atžvilgiu. Naujų priemonių (WS-Security, SAML), pritaikytų internetinių paslaugų saugumui užtikrinti, kai yra daugiau nei du komunikuojantys taškai, standartiniame ASP.NET karkase nėra.
- SSL naudojome tik raktų apsikeitimui bei vartotojų identifikacijai, kardiogramų duomenis našumo bei saugumo sumetimais šifravome naudodami simetrinius algoritimus. .NET karkasas nepalaiko standartinių XML šifravimo standartų, todėl buvo prarasta dalis sistemos išplečiamumo.
- Panaudojus profesionaliose testavimo laboratorijose sukurtus narystės bei vaidmenų modulius, esančius naujajame karkase, pavyko minimizuoti programuotojo klaidos faktorių.
- Realizavę savo vartotojų bei vaidmenų tiekėjus įsitikinome tiekėjo modelio teikiamais privalumais (lengvai pakeičiamas tiekėjas, standartizuota sąsaja, minimalus programinio kodo kiekis).
- Sprendimai, naudoti C# 1.0 kalbos trūkumams apeiti, turi būti pakeisti naujais C# kalbos išplėtimais (gentimis bei tuščiais tipais).

1. PRIEDAS. .NET karkasas

.NET karkasas vis dar yra pakankamai naujas dalykas šiandienos informacinių sistemų realizavime, tačiau užima vis stipresnes pozicijas, ypač tose informacinėse sistemose, kuriose buvo naudojamos COM(COM+) technologijos. Tai reiškia, kad anksčiau ar vėliau, .NET pagrįstos sistemos sudarys vis didesnę dalį internetinių, tarpusavio sąveikos (*middleware*) ir galutinių sistemų (*back-office*) programinėje įrangoje.



30 pav. .NET karkaso pozicija sluoksniuotoje architektūroje



31 pav. .NET karkaso sluoksniai

Trumpai apžvelkime specifinius sluoksnius:

• CLR (Common Language Runtime) – yra karkaso pagrindas, šis sluoksnis valdo programos kodą vykdymo metu, atlikdamas elementarias funkcijas, kaip kad atminties, procesų valdymo, skaičiavimų paskirstymo, taipogi rūpinasi duomenų tipų saugumu ir kitomis formomis užtikrina kodo saugumą,

• Sekantis sluoksnis yra pagrindo klasės, suteikiančios elementarų funkcionalumą, reikalingą duomenų apdorojimui

• Aukščiau yra duomenų apdorojimo sluoksnis, kuris naudodamasis žemiau esančio sluoksnio klasėmis manipuliuoja duomenimis – pavyzdžiui nagrinėja XML dokumentų gramatiką.

• Galiausiai paviršinis sluoksnis, kuriame dažniausiai kuriamos vartojo ir tarpprogramines sąsajos.

Microsoft akcentuoja šias pagrindines .NET karkaso technines charakteristikas:

• paprastesnis (paskirstytų) komponentų kūrimo procesas, nes programuotojui reikia rašyti mažiau programinio kodo lyginant su COM(COM+). Programuotojas paprasčiausiai apibrėžia klases, nesirūpindamas DLL inicializacija, objektų gyvavimo ciklo valdymu, sąsajos paskelbimu ir kitais dalykais, be kurių COM(COM+) programavimas neįsivaizduojamas.

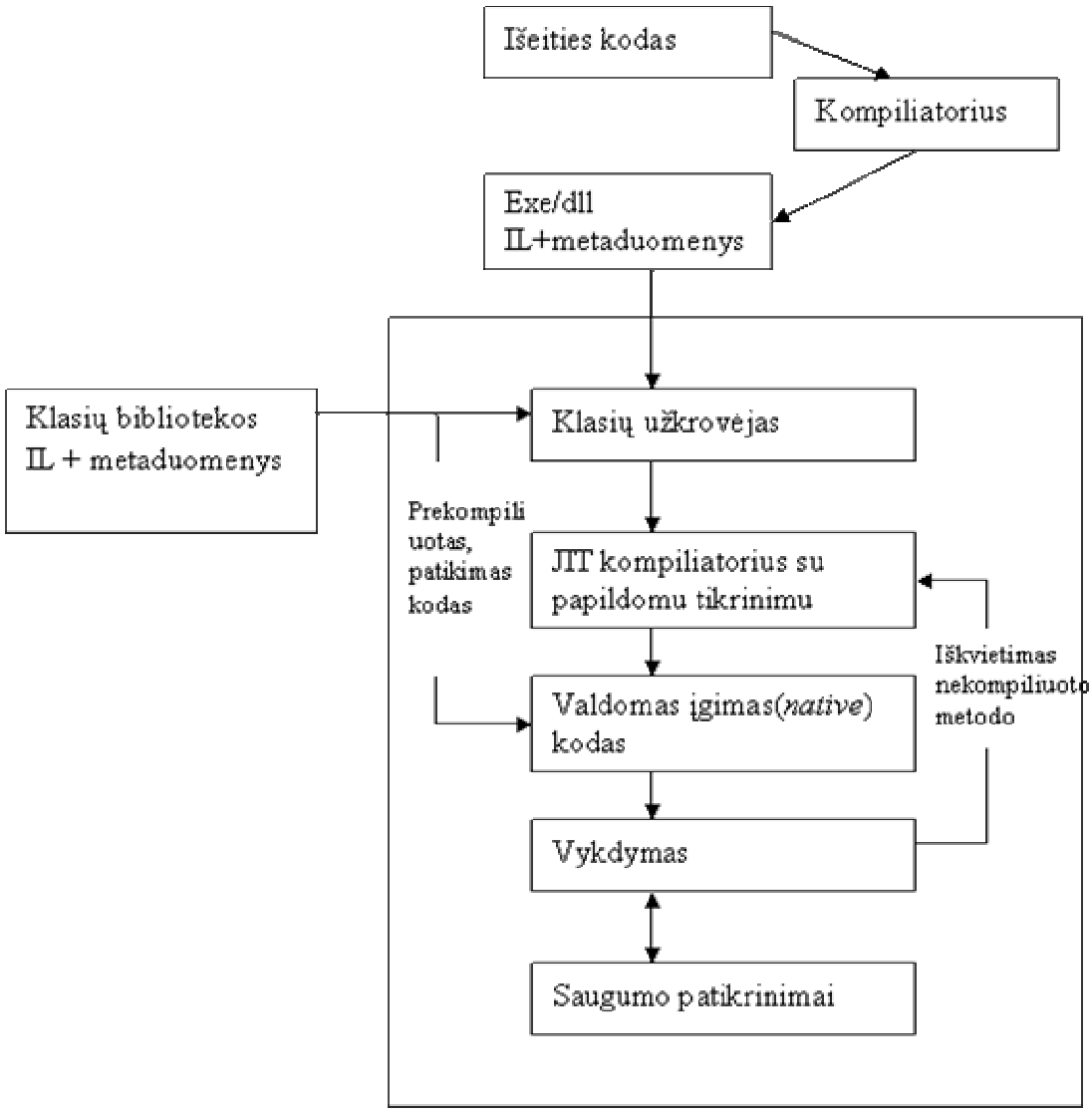
• COM dvejetainio kontrakto modelis leidžia kurti programinę įrangą nepriklausomai nuo programavimo kalbos. .NET yra realizuota kalbų integracija su bendrais duomenų tipais, o taip pat ir gynybos, valdymo ir naujų tipų naudojimo mechanizmais. Tačiau ne kiekviena kalba gali būti vienodai integruota į CLR, dažniausiai dėl pačios kalbos ribotumo.

• CLR naudojamas atminties ir procesų valdymas, duomenų tipų saugumas, dinaminis susiejimas (*dynamic binding*), masyvo ribų tikrinimas – visa tai supaprastina programuotojo darbą bei sumažina žmoniškojo (programuotojo) faktoriaus poveikį saugumui. Kodas, kuris yra valdomas CLR yra apibrėžiamas kaip “valdomas kodas” (*managed code*), šį išsireiškimą me ir toliau naudosime savo darbe.

• .Net turi puikias priemones paskirstytos programinės įrangos kūrimui, kaip kad XML, SOAP, HTTP, o taip pat ir ORB sluoksnio valdymo priemones.

• .NET išsprendė programos versijų problemą, su kuria daugelis programuotojų susidūrė kurdami programinę įranga, pagrįsta COM DLL architektūra.

• .NET realizuotas naujas saugumo modelis, kuriame yra leidimų posistemė, kodo grupės ir kitos galimybės.



32 pav. Programos veikimo schema

Literatūra

1. LaMacchia, A. B.; Lange, S.; Lyons, M.; Martin R.; Price, K. T. *.NET Framework security.* Indianapolis: Addison Wesley, 2002. 816p. ISBN 0-672-32184-X.
2. ZapThink, LLC. *A Guide to securing XML and Web Services*. Waltham, MA, USA, 2004. Prieiga per internetą <www.zapthink.com>.
3. Homer, A.; Sussman, D.; Howard, D. *ASP.NET v.2.0 – The beta version*. Addison Wesley, 2004. 672p. ISBN 0-321-25727-8.
4. Evjen, B. *ASP.NET 2.0 Preview*. Indianapolis: Wiley Publishing, Inc., 2004. 472p. ISBN 0-7645-7286-5.
5. Negm, W. *Anatomy of WebServices Attack*. Boston, MA, USA, 2004. [žiūrėta 2004m. rugsėjo12d.]. Prieiga per internetą <http://www.forumsystems.com>.
6. Lindstrom, P. *Attacking and defending Web Services*. Malvern, PA, USA, 2004. Prieiga per internetą <http://www.spiresecurity.com>.
7. Microsoft. *Beta MSDN library for Visual Studio 2005.* [žiūrėta 2005m. gegužės 2d]. Prieiga per internetą <http://whidbey.msdn.microsoft.com/library/>.
8. Weaver, A.C.; Dwyer, J. K.; Snyder, A. M.; Van Dyke, J.; Hu, J.; Chen, H.; Mulholland, T.; Marshall, A. *Federated, Secure Trust Networks for Distributed Healthcare IT Services.* [žiūrėta 2005m. sausio 20d.]. Prieiga per internetą <http://www.cs.virginia.edu/~acw/security/Federated,%20Secure%20Trust%20Networks%20for%20Distributed%20Healthcare%20IT%20Services.pdf>.
9. Ramachandran, J. *Designing secure architectures*. Indianapolis: Willey, 2002. 483p. ISBN: 0-471-20602-4.
10. Gritzalis, S.; Iliadis, J.; Gritzalis, D.;Spinellis S.; Katsikas, S. *Developing Secure Web-based Medical Applications.* Athens, Greece. [žiūrėta 2005m. sausio 25d.]. Prieiga per internetą <http://www.dmst.aueb.gr/dds/pubs/jrnl/1999-MedInf-Euromed/html/euromed.pdf>.
11. Habib, A.; Hefeeda, M. M.; Bhargava, B. K. *Detecting Service Violations and DoS Attacks*. West Lafayette. [žiūrėta 2004m. gegužės 12d.]. Prieiga per internetą <http://www.isoc.org/isoc/conferences/ndss/03/proceedings/papers/12.pdf>.
12. Weiler, N. *Honeypots for Distributed Denial of Service Attacks.* Zurich, Switzerland. [žiūrėta 2004m. spalio 12d.]. Prieiga per internetą <http://www.tik.ee.ethz.ch/~weiler/papers/wetice02.pdf>.
13. Tulloch, M. *Microsoft Encyclopedia of security*. Wasington: Microsoft Press, 2003. 444p. ISBN 0-7356-1877-1.

14. Alteon Websystems Inc. *Mitigating Denial of Service.* San Jose, California. [žiūrėta 2004m. rugsėjo 15d.]. Prieiga per internetą <http://www.alteon.com>.
15. Anzböck, R.; Dustdar, Sch. *Modeling and Implementing Medical Web services.* Wien, Austria. [žiūrėta 2005m. sausio 25d.]. Prieiga per internetą <http://www.infosys.tuwien.ac.at/Staff/sd/papers/Modeling%20and%20Implementing%20Medical%20Services_DKE.pdf>.
16. Sun Microsystems, Inc. *Securing Web Services – Concepts, Standarts, and Requirements*. Santa Clara, CA, USA, 2003. Prieiga per internetą <http://sun.com/software>.
17. Reactivity, Inc. *Security Within the XML Infrastructure*. Belmont, CA, USA, 2005. Prieiga per internetą <http://www.reactivity.com>.
18. Fernandez, E. B.; Larrondo Petrie, M. M.; Sorgente, T. *Security models for medical and genetic information.* Boca Raton, Fl, USA. [žiūrėta 2005m. sausio 30d.]. Prieiga per internetą <http://polaris.cse.fau.edu/~ed/AvilaV3.pdf>.
19. Faust, S. *SOAP Web Services Attacks (Part 1 – Introduction and simple injection).* Atlanta, GA, USA, 2003. Prieiga per internetą <http://www.spydinamics.com>.
20. Reactivity, Inc. *The Executive Guide to Web Services Security*. Belmont, CA, USA, 2004. Prieiga per internetą <http://www.reactivity.com>.
21. Cabrera, L.F.; Copeland, G.; Freund, T.; Klein, J.; Langworthy, D.; Leymann, F.; Orchard, D.; Robinson, I.; Storey, T.; Thatte, S. *Web Services Business Activity Framework*. 2004. Prieiga per internetą <http://msdn.microsoft.com/library/en-us/dnglobspec/html/wsba.asp>.
22. Della-Libera, G.; Dixon, B.; Garg, P.; Hallam- Baker, G.; Hondo M.; Maruyama H.; Nadalin, A.; Nagaratnam, N.; Nash, A.; Philpott, R.; Prafullchandra, H.; Shewchuk, J.; Simon, D.; Waingold, E.; Zolfonoon, R. *Web Services Trust Language (WS-Trust).* 2002. Prieiga per internetą <http://www.verisign.com/wss/WS-Trust.pdf>.
23. Howard, M. *Writing secure code*. Washington: Microsoft Press, 2002. 451p. ISBN 0-7356-1588-8.