Pēteris GRABUSTS

LIETOTŅU DROŠĪBA

Pēteris GRABUSTS. *Lietotņu drošība* (Otrais izdevums). Rēzekne: Rēzeknes Tehnoloģiju akadēmija. 180 lpp.

Recenzenti:
- Dr. sc. ing. Sergejs PARŠUTINS (RTU)
- Mg. sc. ing. Aleksejs ZORINS (RTA)

Mācību līdzeklis sagatavots un izdots ar Rēzeknes Tehnoloģiju akadēmijas finansiālo atbalstu.

Mācību līdzeklī ir apskatītas lietotņu drošības pamatkoncepcijas un drošības platformu ieviešanas nostādnes, kuras balstītas uz labākās pieredzes apkopojumu, analizēta kriptogrāfijas izmantošana .NET programmās, drošības politika .NET Framework rīku vadīšanā, konfigurēšanā un lietotņu drošības testēšana.
Kurss orientēts uz Microsoft piedāvāto drošības ieviešanas un labāko paņēmienu izmantošanas stratēģiju un balstīts uz Microsoft kursa „2840A: Implementing Security for Applications" materiāliem.
Mācību līdzeklis ir paredzēts Rēzeknes Tehnoloģiju akadēmijas Inženieru fakultātes studiju programmas „Programmēšanas inženieris" studentiem. To var izmantot arī citu datorzinātņu studiju programmu studenti, kā arī visi interesenti, kas vēlas iepazīties ar mūsdienu lietotņu drošības pamatnostādnēm.
Mācību vielas sekmīgai apguvei nepieciešamas padziļinātas iemaņas programmēšanā un darbā ar Microsoft.NET platformu. Obligāts nosacījums ir piemēru izanalizēšana. Teorētisko zināšanu nostiprināšanai doti programmu koda piemēri (realizēti un pārbaudīti Visual Studio 2022 vidē programmēšanas valodā C#, ja nav norādīts citādi), kā arī testa jautājumi katras nodaļas beigās.

Tehniskā redaktore: Vita Ansone

ISBN 978-9984-44-292-1

Satura rādītājs

Apzīmējumu skaidrojums

.NET	*Microsoft*.NET tehnoloģijas apzīmējums (*Microsoft* programmatūras un tīmekļa pielietojumu izstrādes tehnoloģija)
.NET Core	*.NET Core* ir jauna *.NET Framework* versija, kas ir bezmaksas atvērtā koda vispārējas nozīmes izstrādes platforma, ko uztur *Microsoft*
ACE	*(Access Control Entries)* – piekļuves vadības saraksta ieraksti (ACE ievadne)
ACL	*(Access Control List)* – piekļuves vadības saraksts
API	*(Application Programming Interface)* – veids, kā divas vai vairākas datorprogrammas vai komponenti var sazināties savā starpā
ASP.NET	*(Active Server Pages)* – *Microsoft* tīmekļa pielietojumu un servisu izstrādes tehnoloģija
Blazor	jauna *.ASP.NET Core* tehnoloģija, kas atbalsta gan servera puses atveidošanu, gan klienta interaktivitāti vienā programmēšanas modelī
CA	*(Certificate Authority)* – sertificēšanas institūcija
CAS	*(Code Access Security)* – koda piekļuves drošība
CLR	*(Common Language Runtime)* – kopējais valodu izpildlaiks
CLS	(*Common Language Specification*) – kopējā valodu specifikācija
COM	*(Component Object Model)* – programmkomponentu objektmodelis
DACL	*(Discretionary Access Control List)* – īpašnieka ACL
DES	*(Data Encryption Standard)* – simetriskās šifrēšanas algoritms
DREAD	(*Damage potential, Reproducibility, Exploitability, Affected users, Discoverability*) – draudu risku novērtēšanas metodika
GAC	(*Global Assembly Cache*) – globālā asambleju glabātava
HTML	*(Hyper Text Markup Language)* – hiperteksta iezīmēšanas valoda
HTTP	*(Hyper Text Transport Protocol)* – hiperteksta transporta protokols Interneta tīklā
ICMP	(*Internet Control Message Protocol)* – Interneta tīkla vadības ziņojumu protokols
IIS	*(Internet Information Services*) – Interneta tīkla informācijas pakalpojumu dienests
MD5	(*Message Digest*) – jaucējfunkciju šifrēšanas algoritms
MSDN	(*Microsoft Developer Network*) – *Microsoft* produktu attīstītāju tīkls
MSIL	(*Microsoft Intermediate Language*) – *Microsoft* starpvaloda
MVC	(*Model-View-Controller*) - programmatūras projektēšanas modelis, ko parasti izmanto lietotāja saskarņu izstrādei
NTFS	(*NT File System*) – *Microsoft* datņu sistēmu standarts
NTLM	(*NT Lan Manager*) – *Microsoft* autentificēšanas protokols
PGP	(*Pretty Good Privacy*) – datorprogramma, kas ļauj veikt elektroniskā veidā uzdotas informācijas šifrēšanas un ciparparaksta operācijas
Razor	jauna *.ASP.NET Core* tehnoloģija, kas padara uz lapām orientētu scenāriju kodēšanu vieglāku un produktīvāku
PKI	(*Public Key Infrastructure*) – publiskās atslēgas infrastruktūra
RBS	(*Role-Based Security*) – lomās bāzēta aizsardzība
RSA	(*Rivest, Shamir, Adelman algorithm*) – publiskās atslēgšifrēšanas algoritms
SD³+C	*(Secure by Design, Default, Deployment and Communication)* – *Microsoft* drošības nodrošināšanas stratēģija
SHA	(*Secure Hash Algorithm*) – jaucējfunkciju šifrēšanas algoritms
SOAP	(*Simple Object Access Protocol*) – vienkāršais objektu piekļuves protokols
SQL	*(Structured Query Language)* – strukturētā vaicājumvaloda
SSL/TLS	(*Secure Sockets Layer / Transport Layer Security*) – drošligzdu slānis jeb protokols drošas un privātas saziņas nodrošināšanai Interneta tīklā

STRIDE	(*Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege*) – draudu identifikācijas modelis
TCP	*(Transmission Control Protocol)* – pārraides vadības protokols
UI	*(User Interface)* – lietotāja saskarne
URL	(*Uniform Resource Locator*) – vienotais resursu vietrādis
WWW,Web	*(World Wide Web)* – tīmeklis
X.509	publiskās atslēgas infrastruktūras standarts
XML	*(eXtensible Markup Language)* – paplašināmās iezīmēšanas valoda

Tekstā izmantotie apzīmējumi

 - definīcijas, apzīmējumi, termini

 - svarīgs skaidrojums vai rekomendācija

 - piemēri, vingrinājumi

 - piemēri, kurus ieteicams izpildīt ar datoru

 - testa jautājumi

1. Pārskats par lietotņu drošību

Aizsargāta lietotne ir programma, kas nodrošina informācijas un skaitļošanas resursu konfidencialitāti visos līmeņos. Uzticama jeb droša skaitļošana (*Trustworthy Computing*) nav tikai kārtējā mārketinga kampaņa, bet nopietns solis, ko realizē *Microsoft* programmatūras aizsardzībai.

Šī mācību līdzekļa izstrādē pārsvarā tika izmantoti materiāli no oficiālajām *Microsoft* vietnēm. Dažādus materiālus par lietotņu drošības aspektiem var atrast ieteicamās literatūras sarakstā darba beigās. Tā kā latviešu valodā daudzi termini vēl nav aktualizēti, tekstā tie ir minēti arī angļu valodā. Daudzu jēdzienu latviskojums var būt diskutējams.

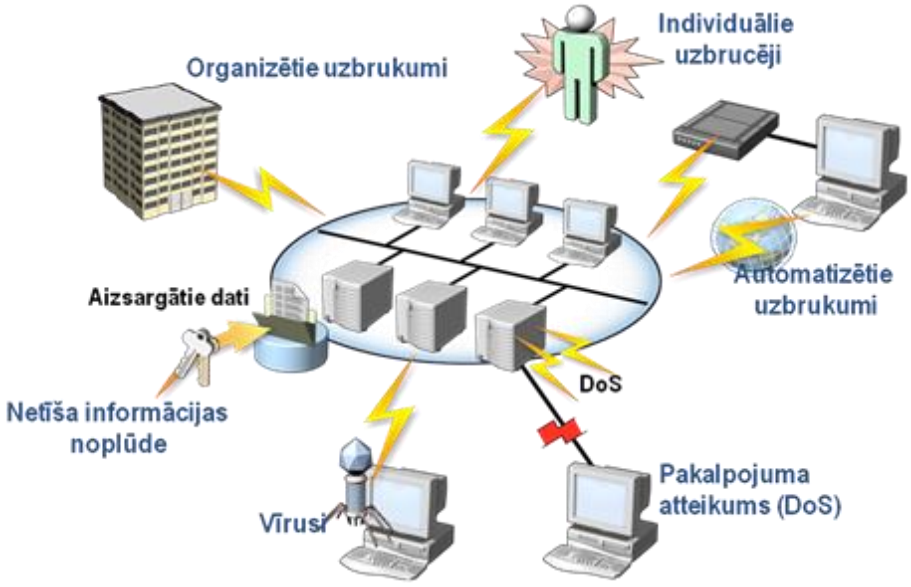
Pirmajā nodaļā apskatīts lietotņu drošības koncepcijas svarīgums, analizēti dažādi drošības jautājumi, apskatīta izstrādātāja loma drošu lietotņu radīšanā, kā arī problēmas, kas rodas drošības sistēmu realizācijā.

1.1. Lietotņu drošības svarīgums

Viss izklāstītais mācību līdzekļa materiāls ir balstīts uz *Microsoft* vietnes un *Microsoft* oficiālā kursa „*2840A: Implementing Security for Applications*" materiāliem (sk. 1.1. att.) un veltīts *Microsoft* piedāvātās drošības ieviešanas un labāko paņēmienu izmantošanas stratēģijas elementu apskatam un analīzei [1], [2].

1.1. att. **Lietotņu drošības kursa logo**

Mūsdienās lietotņu drošības svarīgumu var raksturot ar aizvien pieaugošajiem datorsistēmu un tīklu apdraudējumiem. Tiek izvirzītas paaugstinātas prasības lietotņu drošam darbam datorsistēmās un Interneta tīkla vidē (tipiskākie apdraudējumi parādīti 1.2. attēlā).

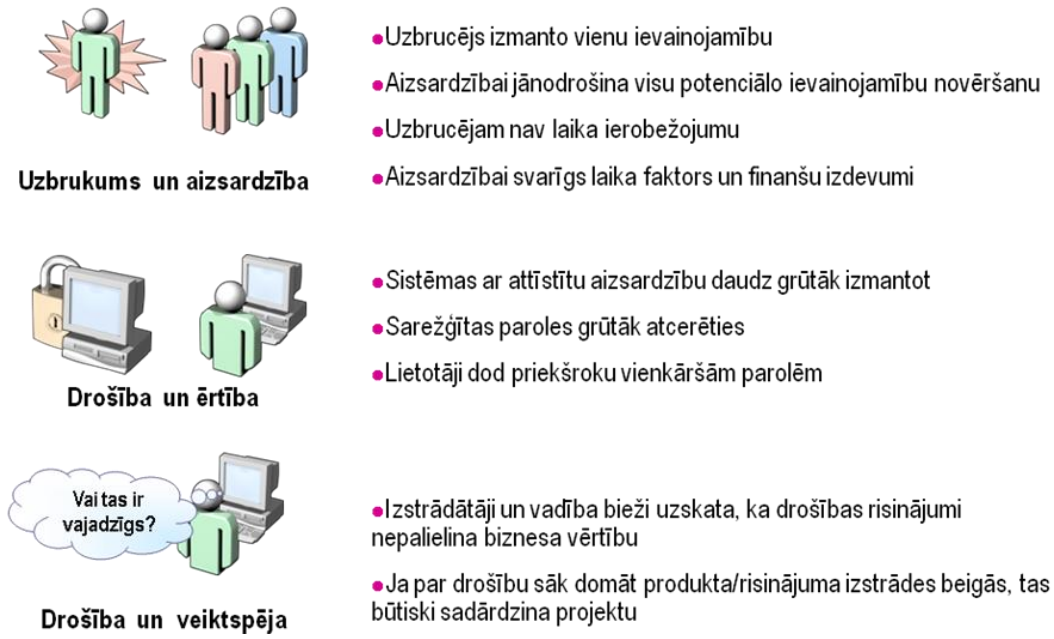


1.2. att. **Tipiski datorsistēmu apdraudējumi**

Vājai drošības sistēmai var būt dramatiskas sekas:

- nozagts intelektuālais īpašums;
- sistēmas dīkstāve;
- veiktspējas zaudēšana;
- kaitējums biznesa reputācijai;
- sagrauta klientu uzticība;
- finansiālie zaudējumi.

Dažas problēmas, kas rodas drošības sistēmu realizācijā, parādītas 1.3. attēlā.



1.3. att. **Drošības sistēmu realizācijas problēmas**

Drošības risinājumu ieviešanu dažkārt sarežģī projektā iesaistītā personāla vājā izpratne par drošības īstenošanas jautājumiem (sk. 1.4. att.).

1.4. att. **Projekta personāla drošības īstenošanas izpratne**

Ja drošība nav prioritāte, visticamāk projekta īstenošanā var atteikties no drošības.

Vispārīgie lietotņu drošības apdraudējumi atspoguļoti 1.1. tabulā.

1.1. tabula

Lietotņu drošības apdraudējumi

Kategorija	Apdraudējums
Nepārbaudīti ievaddati *(nonvalidated data)*	- bufera pārpilde *(buffer overrun)* - SQL injekcija *(SQL injection)* - starpvietņu skriptošana *(cross-site scripting)* - kanonizēšana *(canonicalization)*
Pārkāpta autentificēšana *(broken authentication)*	- pārtveršana tīklā *(network eavesdropping)* - rupja spēka uzbrukumi *(brute-force attack)* - sīkdatņu atkārtota parādīšana *(cookie replay)*
Pārkāpta pilnvarošana *(broken authorization)*	- privilēģiju paaugstināšana *(elevation of privilege)* - konfidenciālu datu atklāšana *(disclosure of confidential data)* - datu falsifikācija *(data tampering)*
Nedroša konfigurācijas pārvaldība *(insecure configuration management)*	- nepilnvarota piekļuve administrēšanas resursiem *(unauthorized access to administration interfaces)*
Slepenu datu atklāšana *(exposing sensitive data)*	- piekļuve slepeniem datiem *(access sensitive data in storage)* - datu falsifikācija *(data tampering)*
Slikta sesijas pārvaldība *(poor session management)*	- sesijas pārtveršana *(session hijacking)* - sesijas atkārtošana *(session replay)*
Vāja kriptogrāfijas metožu izmantošana *(weak cryptography)*	- slikta paroļu ģenerēšana vai pārvaldība *(poor key generation or key key management)*
Manipulācijas ar parametriem *(parameter manipulation)*	- manipulācijas ar vaicājumu rindām *(query string manipulation)* - manipulācijas ar formu laukiem *(form field manipulation)*
Nepiemērota izņēmumu apstrāde *(improper exception management)*	- informācijas atklāšana *(information disclosure)* - pakalpojumu atteikums *(denial of service)*
Slikta auditēšana un žurnālu vadība *(poor auditing and logging)*	- lietotāja darbību atteikums *(user denies performing an operation)*

Piemēra nolūkā apskatīti šādi visbiežāk izplatītie lietotņu drošības draudi:

- bufera pārpilde [3];
- SQL injekcija [4];
- starpvietņu skriptošana [5];
- kanonizēšana [6].

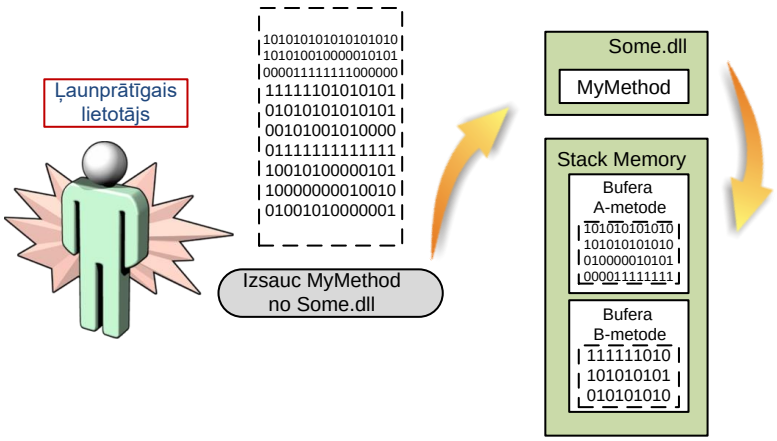
Bufera pārpilde

Buferis (*buffer*) – atmiņa īslaicīgai datu glabāšanai, ko pārsūta starp divām ierīcēm, lai izlīdzinātu to darbības ātrumus, signālu līmeņus vai nodrošinātu asinhronu pārraidi.

Pārpilde (*overflow*) – noteiktu datu vai programmu uzglabāšanai datorā paredzētā atmiņas apjoma pārsniegšana.

Bufera pārpilde (steka pārpildes veidā) notiek gadījumā, kad programma ieraksta datus ārpus tai izdalītā bufera robežām. Tā rezultātā var tikt sabojāti dati, kas atrodas uzreiz aiz bufera. Stekā izvietotie mainīgie fiziski atrodas blakus izsaucošās funkcijas atpakaļadresei uz programmu. Par tādu kļūdu iemeslu parasti ir dati, ko lietotājs ievada un nodod rindu apstrādes funkcijām, piemēram, `strcpy`. Tā rezultātā īstā atpakaļadrese var tikt pārrakstīta ar „viltus" adresi. Bufera pārpilde ir viens no populārākajiem datorsistēmu apdraudējumu veidiem, jo augsta līmeņa programmēšanas valodas uzglabā atmiņā gan programmu, gan datus.

Bufera pārpilde var izraisīt programmas nekorektu darba beigšanos vai uzkāršanos, kas izraisa atteikumu apkalpošanā. Atsevišķi pārpildes veidi, piemēram, pārpilde stekā, dod iespēju uzbrucējam ielādēt un izpildīt savu mašīnkodu ar konkrētās programmas tiesībām (sk. 1.5. att.).

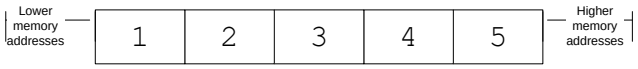


1.5. att. **Bufera pārpildes ilustrēšana**

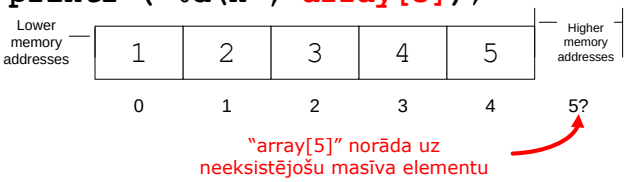
Bufera pārpildes situāciju rašanās potenciālās iespējas demonstrētas 1.1. un 1.2. piemērā.

1.1. piemērs. Izeja ārpus masīva robežām

```
int array[5] = { 1, 2, 3, 4, 5 };
```




```
int array[5] = { 1, 2, 3, 4, 5 };
printf ("%d\n", array[0]);
printf ("%d\n", array[4]);
printf ("%d\n", array[5]);
```



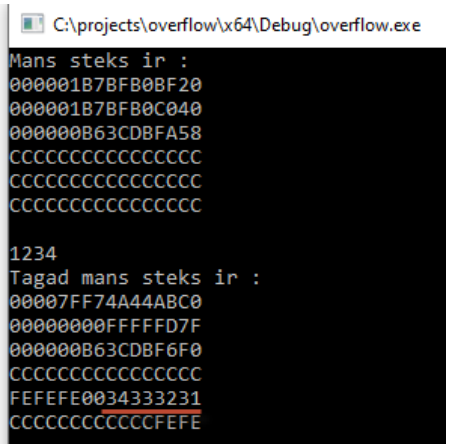
1.2. piemērs. Bufera pārpildes demonstrācija (C++)

```
#include <iostream>
#include <stdio.h>
#include <conio.h>
#include <string.h>
void ievade(const char* input)
{
   char buf[10];          // Buferis 10 mainīgajiem
   printf("Mans steks ir :\n%p\n%p\n%p\n%p\n%p\n%p\n\n");
   strcpy_s(buf, input);
```

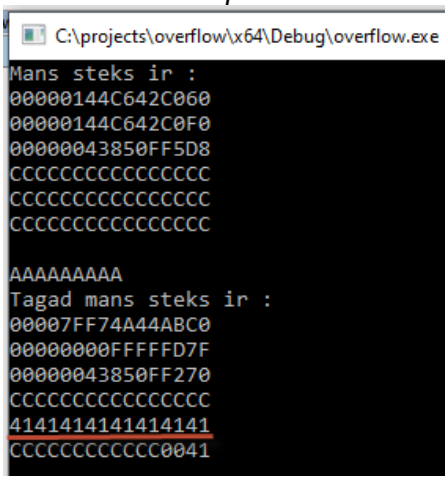
```
    printf("%s\n", buf);
    printf("Tagad mans steks ir :\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n\n");
    _getch();
}
void navarg(void)
{
    printf("Nav argumentu!\n");
    _getch();
}
int main(int argc, char* argv[])
{
    if (argc != 2)
    {
        navarg();
    }
    else
    {
        ievade(argv[1]);
    }
    return 0;
}
```

Rezultāts

Pirmajā gadījumā komandrindā tika uzrakstīts: *overflow 1234*. Rezultāts ir šāds (heksadecimālajā notācijā):



Otrajā gadījumā komandrindā tika uzrakstīts: *overflow AAAAAAAAA*. Rezultāts ir šāds:



SQL injekcija

SQL injekcija (*injection*) – datorsistēmu apdraudējuma veids, kurā uzbrucējs ar klienta tiesībām mēģina piekļūt datubāzei, kas ir dotās tīmekļa lapas vai aplikācijas pamatā. Mūsdienās SQL injekcija ir vispopulārākais aplikācijas līmeņa uzbrukuma veids. SQL injekcijas pamatā ir nepareizi uzrakstītas programmas vai tīmekļa aplikācijas, kas lietotājam ļauj veikt datubāzes vaicājumus, kuras nav paredzējis programmatūras vai mājaslapas izstrādātājs. Šai situācijai pamatā parasti ir nepietiekama lietotāju ievadītās informācijas pārbaude, kuras rezultātā netiek filtrēti SQL īpašie simboli.

1.3. piemērs. SQL injekcija

Dota autorizācijas forma, kurā jāievada lietotājvārds un parole. Formas nosūtīšanas rezultātā tiek izpildīts šāds SQL pieprasījums:

SELECT * FROM lietotaji WHERE lietotajvards='LIETOTAJVARDS' AND parole='PAROLE';

Uzbrucējam nepieciešams piekļūt attiecīgās lietotnes administratora ierakstam. Lai to izdarītu, SQL vaicājumu nepieciešams šādi modificēt:

SELECT * FROM lietotaji WHERE lietotajvards='admin' OR lietotajvards=kautkas AND parole='%%'

Ievadot autorizācijas formā lietotājvārda vietā *admin' OR lietotajvards=kaut kas*, būs iespējams autorizēties kā lietotājam "admin", nezinot paroli. Dotais SQL piemērs pirmajā gadījumā atlasīs ierakstu lietotāju tabulā kā meklēšanas kritērijus, izmantojot lietotāja vārdu UN paroli. Otrajā gadījumā kā meklēšanas kritērijs tiks izmantots lietotāja vārds UN parole VAI tikai lietotāja vārds. Modificējot datubāzes vaicājumu un pievienojot tā beigās loģisko nosacījumu, kas vienmēr izpildās, uzbrucējs lielākajā daļā gadījumu būs spējīgs nolasīt visus dotās datubāzes tabulas ierakstus vai arī no kļūdas paziņojumiem izdarīt kādus secinājumus.

Lai izvairītos no šāda tipa uzbrukumiem, nepieciešams veikt drošības pasākumus – vienmēr un visur rūpīgi filtrēt ievades parametrus, t. i., pārbaudīt, vai lietotāju ievadītais teksts nesatur kādu no SQL robežsimboliem.

Starpvietņu skriptošana

Skripts *(script)* – instrukciju virkne, kas nosaka, kā programmai jāveic kāda specifiska procedūra, piem., ieiešana elektroniskā pasta sistēmā. Dažādās programmās skripta iespējas ir jau iebūvētas, dažus skriptus veido automātiski, pierakstot, kādus taustiņus un komandu izvēlnes lietotājs izmanto procedūru laikā. Globālā tīmekļa kontekstā skripts ir programma, kas atrodas kādā tīmekļa serverī un apstrādā no pārlūkprogrammas saņemtos pieprasījumus.

Starpvietņu skriptošana XSS *(cross-site scripingt)* – XSS (*Cross Site Scripting*) uzbrukums ir uzbrukuma veids tīmekļa aplikācijām. Uzbrucējs izmanto XSS, lai iemānītu savu izveidoto skriptu gala lietotājam. Tā kā lietotāja pārlūkprogramma uzskata, ka skripts ir saņemts no uzticama informācijas avota un tai nav nekādas iespējas uzzināt to, ka skriptam nedrīkst uzticēties, programma palaidīs skriptu, tādējādi nodrošinot tam piekļuvi jebkurām lietotāja sīkdatnēm (*cookies*), sesiju datiem un citai informācijai, kuru lietotāja pārlūkprogrammā izmanto tīmekļa lapā. Ar šādu skriptu palīdzību iespējams pārrakstīt HTML lapas saturu.

1.4. piemērs. XSS paraugs

Tīmekļa lapa, kurai kā parametrs tiek nodots kāda cilvēka vārds vai iesauka, lai apsveiktu to dzimšanas dienā:
http://example.lv/happy_birthday.php?name=Bob
Atverot šo adresi, lietotājam parādās uzraksts "Happy birthday, Bob", kas HTML valodā izskatās šādi:

<html><body>
Happy birthday, Bob
</body></html>

Bet, atverot šo adresi:
http://example.lv/happy_birthday.php?name=*Bob*
lapas saturs pārvērtīsies par:

<html><body>
Happy birthday, *Bob*
</body></html>

Šajā gadījumā tiek injicēta HTML *font* birka, kas pati par sevi neko ļaunu lietotājam nenodara, bet *font tag* vietā iespējams ievietot *script* birku, ar kuras palīdzību lapā var ievietot *Java* valodas skriptu.

Kanonizēšana

Datorzinātnē kanonizēšana ir process, kas pārvērš ievades dažādos iespējamos attēlošanas veidus vienā „standarta" kanoniskā prezentācijā (novešana kanoniskā formā).

Datņu vārdi, ceļi līdz datnēm (*path*) un vietrādis (URL) ir resursi, kuriem visbiežāk nākas piemērot kanonizēšanas procedūru, jo to attēlojumu veidi var būt dažādi.

Datņu vārdi ir kanonizēšanas problēmas subjekts. Visi nākamie vārdi, piemēram, paredz piekļuvi vienai un tai pašai datnei:

- *MyLongFile.txt* – tas ir nepārprotams standarta datnes vārds;
- *MyLongFile.txt.* – *Win32* datņu sistēma nosaka, ka punkts beigās nevar būt un tas tiek ignorēts;
- *MyLong~1.txt* – FAT pieprasa 8 zīmju datņu vārdus (8.3. formāts), bet FAT32 un NTFS uztur garus datņu vārdus. Saderības nodrošināšanai ar 8.3. formātu garie vārdi tiek saīsināti;
- *MylongFile.txt::$DATA* – NTFS datņu sistēmā *$DATA* ir datu straumējuma (*stream*) apzīmējums pēc noklusējuma. Lietotājs, piemēram, ar teksta redaktoru var izveidot datni citā straumējumā, un datņu sistēma to neatpazīs: *notepad tests.txt:aaa*.

Tāpat ir daudz veidu, kā attēlot simbolus vietrādī URL.

URL (*Uniform Resource Locator*) – vienotais resursu vietrādis. Tā ir adrese, kas pārlūkprogrammā norāda, kur var atrast kādu konkrētu Interneta tīkla resursu. Vienveida resursu vietrādim varētu būt, piem., šāda struktūra: *http://www.edzi.lza.lv/history.htm*. Vietrāža pirmā daļa (līdz dubultajai slīpsvītrai) norāda piekļuves metodi (šajā piemērā – hiperteksta transporta protokolu). Teksts starp dubulto slīpsvītru un vienkāršo slīpsvītru norāda serveri, bet teksts aiz vienkāršās slīpsvītras – mapi vai datni.

1.2. Nozares ekspertu ieteikumi

Drošības jautājumi ir jāapskata:

- visās projektēšanas stadijās;
- visos informācijas sistēmas līmeņos.

Pastāv vairākas drošības paaugstināšanas pieejas (sk. 1.2. tabulu).

1.2. tabula

Drošības paaugstināšanas pieejas

Pieeja	Ieguvums
Izstrādātāju komandas apmācība	Drošības tehnoloģiju pareiza pielietošana
Apdraudējumu analīze	Ievainojamību identificēšana
Koda pārbaude	Drošs kods: • piekļuvei tīklā; • startēšanai ar parametriem pēc noklusējuma; • protokolu bez autentifikācijas izmantošanai; • startēšanai ar paaugstinātām privilēģijām
Rīku izmantošana	Rūpīgāka potenciālo ievainojamību testēšana
Infrastruktūras izmantošana	Augstāka drošība ar SSL/TLS
Gatavu komponentu izmantošana	*CryptoAPI* un .NET *Cryptography* vārdu telpas
Pāreja uz izpildāmo kodu	Izvairīšanās no potenciālas ievainojamības

Pirmais un vissvarīgākais solis labas drošības ieviešanai ir pārliecība par to, ka drošības nodrošināšana ir daļa no projekta izstrādes. *Microsoft* piedāvā drošības nodrošināšanas stratēģiju, kas tika nosaukta par SD^3+C struktūru ***(Secure by Design, by Default, in Deployment, by Communication)*** [7].

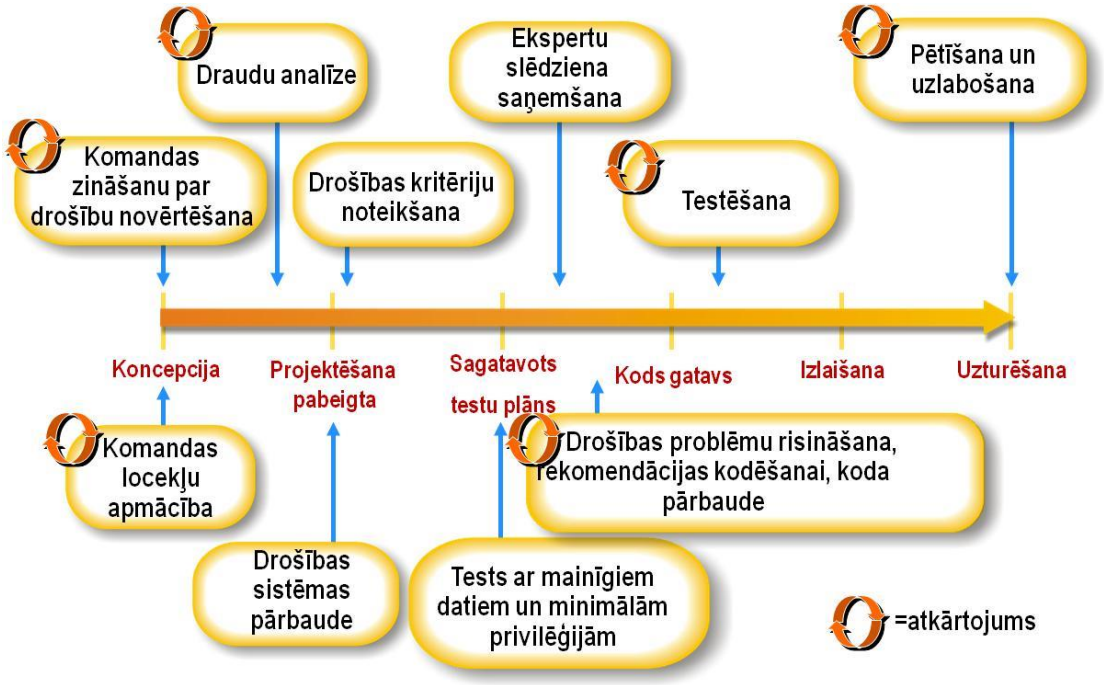
SD^3+C struktūra satur četrus galvenos komponentus (sk. 1.3. tabulu).

1.3. tabula

SD^3+C struktūras komponenti

Komponents	Iezīmes
Drošība projektēšanā *(Secure by Design)*	Drošības sistēma projektēšanā un kodēšanā Apdraudējumu analīze Ievainojamības samazināšana
Drošības iestatījumi pēc noklusējuma *(Secure by Default)*	Uzbrukuma iespēju samazināšana Iestatījumu pēc noklusējuma neatslēgšana Minimālās nepieciešamās privilēģijas
Drošība ieviešanā un ekspluatācijā *(Secure in Deployment)*	Aizsardzība: apdraudējumu konstatēšana, pretdarbība, atjaunošana un vadība Vadības process: detalizēts apraksts Personāls: apmācība
Drošība saziņā *(Secure by Communication)*	Skaidra un kodolīga dokumentācija Intereškopas

Drošības jautājumu risināšana dažādos izstrādes posmos parādīta 1.6. attēlā.



1.6. att. **Drošības jautājumi izstrādes posmos**

Būtiska nozīme projektēšanas procesā ir apdraudējumu modelēšanai:

- svarīga projektēšanas sastāvdaļa;
- samazina drošības sistēmas izstrādes izmaksas;
- palīdz izstrādātāju komandai:
  - identificēt neaizsargātās vietas;
  - noteikt, kādus apdraudējumus nepieciešams samazināt un kā to panākt.

Apdraudējumu modelēšanas procesa posmi parādīti 1.7. attēlā.

1.7. att. **Apdraudējumu modelēšanas posmi**

Nepieciešams identificēt šos draudus un apkopot informāciju par potenciālajiem apdraudējumiem. STRIDE modelis (***S**poofing, **T**ampering, **R**epudiation, **I**nformation disclosure, **D**enial of service, **E**levation of privilege*) [8] nosaka potenciālo apdraudējumu tipus (sk. 1.4. tabulu).

1.4. tabula

Draudu identifikācijas STRIDE modelis

Apdraudējumu tips	Piemēri
Izlikšanās (*Spoofing*)	Elektronisko sūtījumu viltošana Autentificēšanas pakešu viltošana
Falsifikācija (*Tampering*)	Pa tīklu pārraidāmo datu modificēšana Datņu modificēšana
Dalības noliegums (*Repudiation*)	Kritiskas datnes izdzēšana Pirkuma veikšana ar turpmāku atteikšanos no savām darbībām
Informācijas atklāšana (*Information disclosure*)	Nesankcionēta pieeja Nelikumīga konfidenciālas informācijas publicēšana
Pakalpojumu atteikums (*Denial of service*)	Tīkla aizpildīšana ar SYN paketēm Tīkla resursu noslogošana ar lielu skaitu viltotu ICMP pakešu
Privilēģiju paaugstinājums (*Elevation of privilegies*)	Sistēmas privilēģiju iegūšana, izmantojot bufera pārpildīšanos Nelikumīga administratora tiesību iegūšana

SYN (TCP) (*Transmission Control Protocol*) – pārraides vadības protokols ir viens no galvenajiem Interneta tīkla protokoliem. TCP ļauj tīkla datoru programmām izveidot savienojumus vienai ar otru, caur kuriem var pārsūtīt datus. Šis protokols garantē drošu un secīgu datu pārraidi un atšķir dažādu programmu savienojumu datus (lietojot pieslēgvietu).

ICMP (*Internet Control Message Protocol)* – Interneta tīkla vadības ziņojumu protokols, t. i., tīkla slāņa protokols, ko izmanto, lai informētu par kļūdām un citiem gadījumiem, kas radušies datu pārraides laikā. Tāpat ICMP izmanto, lai sūtītu testu paketes un citus informatīvos ziņojumus.

Apdraudējumu risku samazināšanas paņēmieni ir šādi:

- neko nedarīt;
- brīdināt lietotāju;
- novērst problēmu;
- novērst cēloni.

Apdraudējumu risku novērtēšanai pastāv metodika, kas tika nosaukta par DREAD (***D**amage potential, **R**eproducibility, **E**xploitability, **A**ffected users, **D**iscoverability*) [9]:

- potenciālais kaitējums (*Damage potential*);
- pavairošana (*Reproducibility*);
- pakļaušanās uzlaušanai (*Exploitability*);
- lietotāji, kas pakļauti uzbrukumam (*Affected users*);
- atklāšanas iespējas (*Discoverability*).

Metodes cīņā ar apdraudējumiem apkopotas 1.5. tabulā.

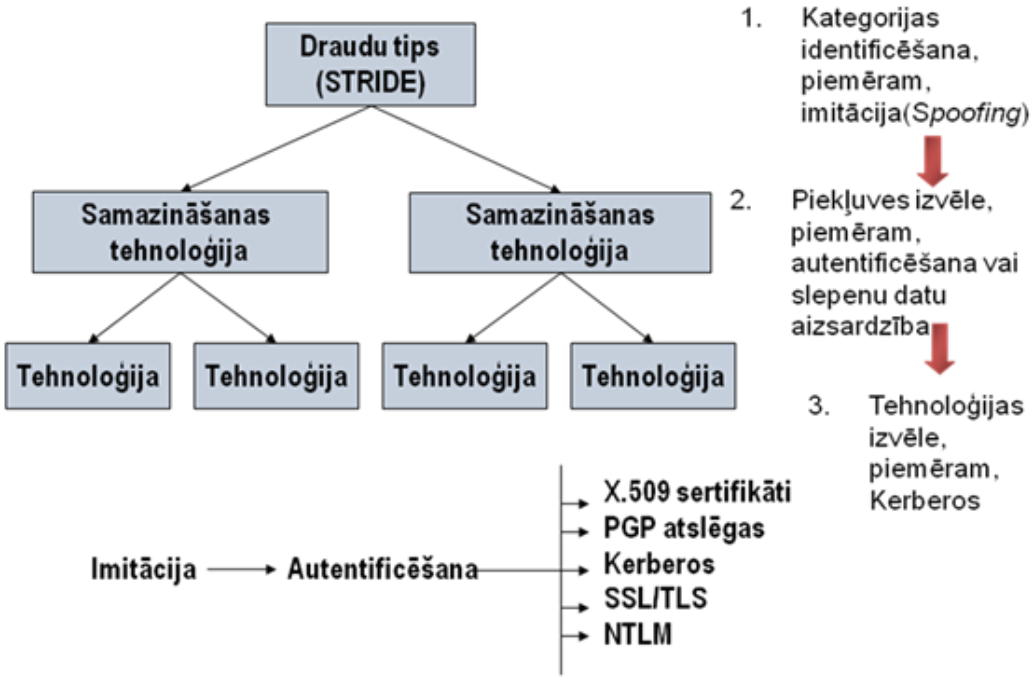
1.5. tabula

Metodes cīņā ar apdraudējumiem

Apdraudējumu tips	Piemēri
Izlikšanās	• Drošs autentificēšanas mehānisms • Slepeno datu aizsardzība
Falsifikācija	• Drošs pilnvarošanas mehānisms • Jaucējfunkciju izmantošana • Ciparparaksts • Droši komunikācijas protokoli
Dalības noliegums	• Ciparparaksts • Datuma un laika iezīmes • Kontrolpunktu trasēšana
Informācijas atklāšana	• Pilnvarošana • Droši komunikāciju protokoli • Šifrēšana • Slepeno datu aizsardzība
Pakalpojumu atteikums	• Drošs autentificēšanas mehānisms • Drošs pilnvarošanas mehānisms • Filtrēšana
Privilēģiju paaugstinājums	• Darbs ar minimālām privilēģijām

Risku samazināšanas process sastāv no trijiem soļiem (sk. 1.8. att.):

1) draudu kategorijas identifikācija pēc STRIDE modeļa;
2) draudu samazināšanas tehnoloģijas izvēle;
3) izvēlētās tehnoloģijas pielietošana.



1.8. att. **Risku samazināšanas process**

X.509 sertifikāts – publiskās atslēgas infrastruktūras (PKI) standarts, kas apraksta publiskās atslēgas sertificēšanas procedūru.

PKI (*Public Key Infrastructure*) – publiskās atslēgas infrastruktūra, t. i., ciparsertifikātu izsniegšanas, sertificēšanas un citu reģistrējošo institūciju sistēma, kas Interneta tīkla transakcijās pārbauda un autentificē katra iesaistītā lietotāja identitāti. Pēdējā laikā droša PKI infrastruktūra tiek uzskatīta par galveno priekšnoteikumu plašai elektroniskās tirdzniecības lietošanai.

PGP (*Pretty Good Privacy*) – datorprogramma, kas ļauj veikt elektroniskā veidā uzdotas informācijas šifrēšanas un ciparparaksta operācijas.

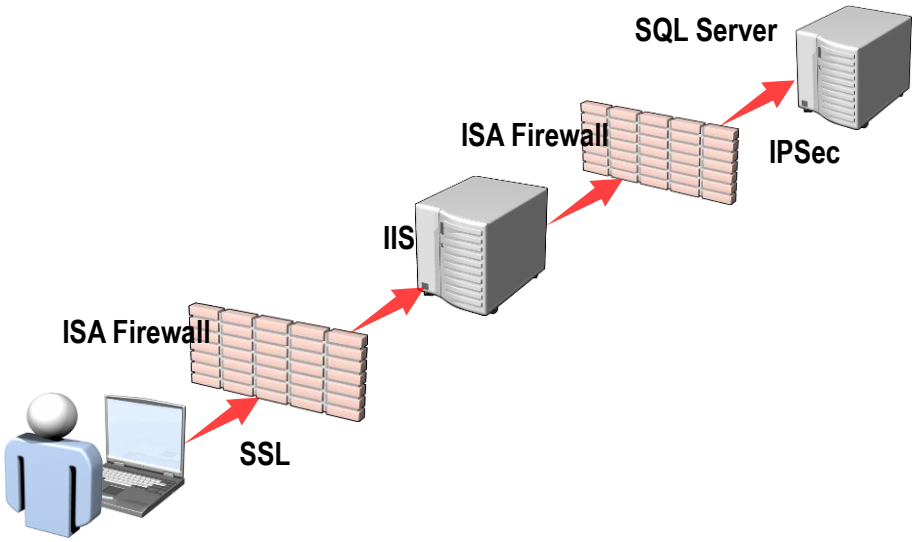
Kerberos – protokols "Kerberos", kas autentificē lietotājus, kuri mēģina pieteikties darbam datoru tīklā, un kas, izmantojot slepenu šifrēšanas atslēgu, šifrē datu apmaiņu starp šiem lietotājiem.

SSL/TLS (*Secure Sockets Layer / Transport Layer Security*) – drošligzdu slānis jeb protokols drošas un privātas saziņas nodrošināšanai Interneta tīklā. Kad seanss, kurš izmanto drošligzdu slāni, ir sācies, serveris aizsūta publisko atslēgu pārlūkprogrammai. Pārlūkprogramma to izmanto, lai nosūtītu patvaļīgi ģenerētu slepeno atslēgu atpakaļ serverim, tādējādi nodrošinot iespēju apmainīties ar slepenajām atslēgām seansa laikā.

NTLM (*NT Lan Manager*) – *Microsoft* autentificēšanas protokols.

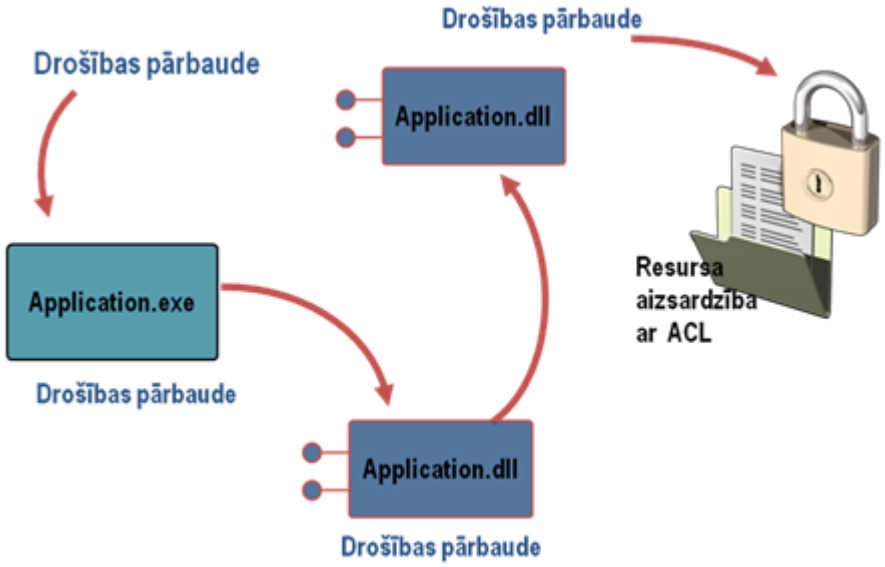
Praktiskajā darbībā eksperti dod virkni rekomendāciju, kuru ievērošana var būtiski palielināt lietotņu drošību.

- Strādāt ar minimālām privilēģijām (*run with least privilege*).
  Tiek uzsvērta labi zināma drošības doktrīna: „Dodam tik daudz tiesību, cik vajag darbam, un ne vairāk!". Privilēģiju paaugstināšana var radīt katastrofālas sekas. Ļaunprātīga lietotāja kodam, kas palaists priviliģētā režīmā, arī ir šīs privilēģijas- daudzi vīrusi izplatās tāpēc, ka saņēmējam ir administratora tiesības.
- Lietot iestatījumus pēc noklusējuma (*apply secure default*).
- Veikt ievades datu validāciju (*validate user input*).
  Tipiskākie draudi lietotnēm: bufera pārpilde, SQL injekcija, starpvietņu skriptošana. Jācenšas pārbaudīt ievadāmo datu tipu, formātu, izmēru vai diapazonu. Jāatceras, ka viss, kas ievadīts, var būt bīstams – kamēr nav pierādīts pretējais;
- Lietot daudzkārtīgu aizsardzību (*use the defense-in-dept model*).
  Pēc iespējas jāizmanto ugunsmūri (*firewall*) (sk. 1.9. att.).



1.9. att. **Ugunsmūru uzstādīšana**

- Katram līmenim jāizmanto attiecīgi pārbaudes kritēriji (sk. 1.10. att.).



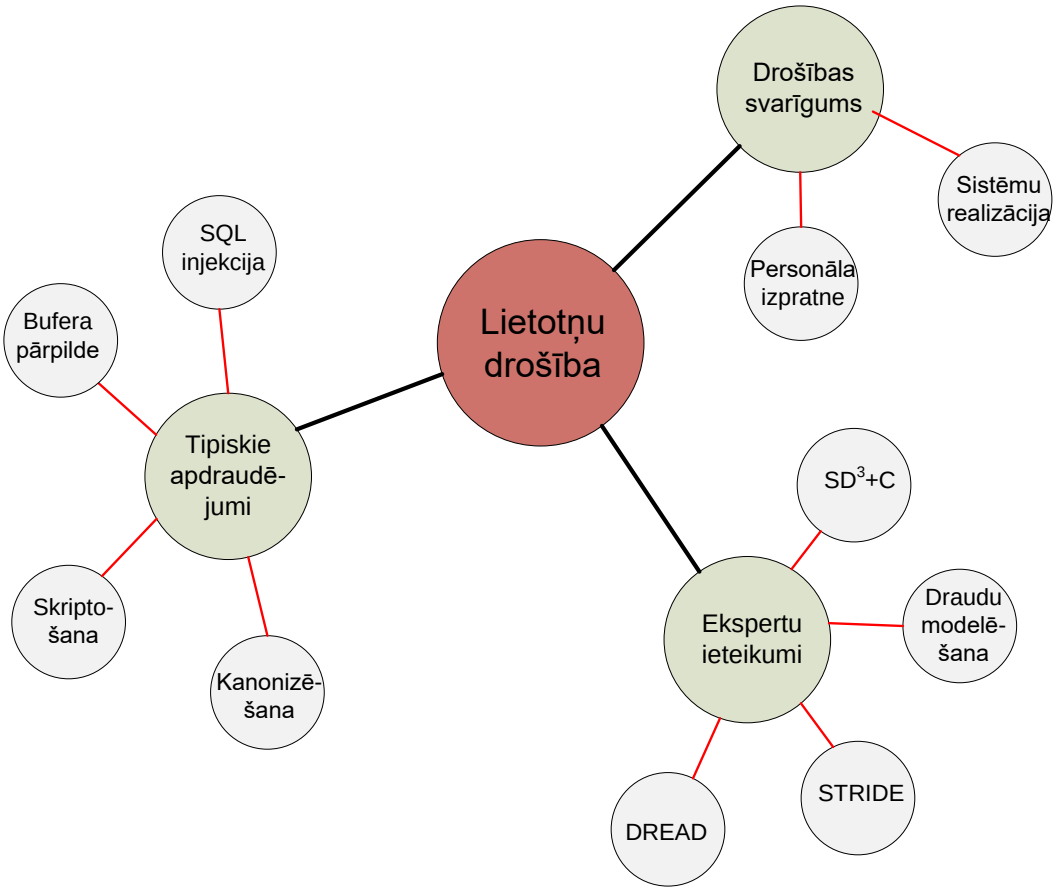
1.10. att. **Drošības pārbaude dažādos līmeņos**

- Visi resursi jāaizsargā ar ACL *(Access Control List)* – piekļuves vadības sarakstu palīdzību:
  - jāprojektē ACL lietotnes resursiem no paša izstrādes sākuma;
  - jāizmanto ACL datnēm, mapēm, tīmekļa lapām, reģistra atslēgām, datu bāzēm, printeriem un *Active Directory* objektiem;
  - jāveido savi ACL lietotnes uzstādīšanas procesā.
- Nepaļauties uz „neskaidrību" drošības jautājumos (*do not rely on obscurity*). Ja kas liekas nesaprotams drošības jautājumos, tas nenozīmē, ka uzbrucējs to neizmantos.
- Neglabāt atslēgas/paroles neaizsargātās datnēs (*do not store secret information*).
- Lietot DPAPI (*Data Protection API*) datu aizsardzībai (*use DPAPI*).
- Veikt drošības sistēmas testēšanu (*test security*):
  - jāiesaista testētāji pašā projekta sākumā;
  - jāizmanto STRIDE pieeja draudu modelēšanā;
  - jātestē viss;
  - jāiegūst pieredze un jāpēta hakeru tehnoloģijas.
- Mācīties no kļūdām (*learn from mistakes*).

Nav iespējams izstrādāt drošu sistēmu, nesaprotot, kādas briesmas tai draud!

Informāciju par dažādiem lietotņu drošības aspektiem var meklēt darbos [10] – [15].

1.11. attēlā parādīts nodaļā minēto svarīgāko jēdzienu koks.



1.11. att. **Pirmajā nodaļā minēto svarīgāko jēdzienu koks**

Pārbaudes testa jautājumi

1. Tika izstrādāta tīmekļa kataloga vadības sistēma, kas jau sagatavota nodošanai klientam. Pēkšņi atklājas defekts lietotāju reģistrācijas lapā, kas var izraisīt drošības draudus. Kā labāk risināt šo problēmu?
   a. Pagaidām atslēgt reģistrēšanās iespēju
   b. Nodot sistēmu lietotājam un vēlāk veikt labojumus
   c. Informēt klientu un pārcelt nodošanas datumu
   d. Nedarīt neko no minētā.

2. Dotajā kodā ir drošības ievainojamība. Identificēt draudu kategoriju, kam pakļauts šis kods.

```
int catvars(char *buf1, char *buf2, unsigned int len1,unsigned int len2)
  {
            char mybuf[256];
      if((len1 + len2) >256){
               return -1;
            }
     memcpy(mybuf, buf1, len1);
            memcpy(mybuf + len1, buf2, len2);
       do_some_stuff(mybuf);
            return 0;
  }
```

a. SQL injekcija
b. Bufera pārpilde
c. Starpvietņu skriptošana
d. Nekas no minētā

3. *Windows .NET Server* dienests *Telnet* darbojas ar parasta lietotāja privilēģijām. Tas nodrošina to, ka ļaunprātīgs lietotājs nevarēs izmantot šī servisa drošības ievainojamību, lai manipulētu ar sistēmu. Šis ir piemērs:
   a. Drošībai projektēšanā
   b. Drošības iestatījumiem pēc noklusējuma
   c. Drošībai ieviešanā un ekspluatācijā
   d. Drošībai saziņā

4. SQL serveris tiek izmantots klientu kredītkaršu informācijas uzglabāšanai. Kamēr tiek īstenota servera drošība, atklājas, ka ļaunprātīgam lietotājam ir iespēja piekļūt datu bāzei ar kredītkaršu informāciju. Identificēt šī drauda STRIDE kategoriju vai kategorijas.
   a. Izlikšanās
   b. Falsifikācija
   c. Dalības noliegums
   d. Informācijas atklāšana
   e. Pakalpojumu atteikums
   f. Privilēģiju paaugstinājums

5. Tiek izstrādāta tīmekļa forma datu uzskaitei, kas satur darbinieka vārdu, vecumu, algas apjomu un struktūrvienību. Kāda veida ievades pārbaude jāveic rekvizītam „vecums”?
   a. Tipa pārbaude
   b. Garuma pārbaude
   c. Diapazona pārbaude
   d. Formāta pārbaude

2. Labākie paņēmieni drošības ieviešanai

Drošības nodrošināšana ir nepārtraukts process. Lai arī vispārīga aizsardzības nodrošināšana ir sarežģīta tēma, tomēr var atrisināt daudzas lietotņu drošības problēmas, izmantojot ekspertu rekomendācijas drošības nodrošināšanai dažādos līmeņos.

Otrajā nodaļā apskatīti resursu aizsardzības labākie paņēmieni, kas attiecas uz COM+, Interneta tīkla informācijas servera un SQL servera drošāku izmantošanu. Pievērsta uzmanība resursu piekļuves vadības saraksta (ACL) izmantošanai, minimālo privilēģiju ieviešanas svarīgumam un auditēšanas nepieciešamībai. Tāpat šajā nodaļā apskatīta kriptogrāfijas iespēju izmantošana un datu aizsardzības koncepcija.

2.1. Ieteikumi IIS, SQL servera un COM+ drošākai izmantošanai

Internet tīkla informācijas pakalpojumu dienests IIS (*Internet Information Services*) ir tīmekļa serveris ar pilnu līdzekļu klāstu, kas ļauj lietot tīmekļa lietojumprogrammas un XML pakalpojumus.

Paplašināmās iezīmēšanas valoda XML *(eXtensible Markup Language)* – programmēšanas valodas SGML apakškopa, kas speciāli izstrādāta darbam ar tīmekļa dokumentiem. Valoda XML nodrošina globālā tīmekļa izstrādātajiem iespēju radīt savas personiskās birkas (kodus), lai nodrošinātu tādu funkciju izpildi, ko nevar nodrošināt ar valodas HTML starpniecību. Atšķirībā no valodas HTML, kuras saites norāda tikai uz vienu dokumentu, valoda XML nodrošina saites, kurās ir atsauces uz vairākiem dokumentiem.

IIS drošības pasākumu komponenti doti 2.1. tabulā.

2.1. tabula

IIS drošības nodrošināšanas komponenti

Komponents	Iezīmes
Šifrēšana	IIS uztur ASV valdības drošības standartu *Fortezza,* kas ietver drošības nodrošināšanu ziņojumu aizsardzībai
Ciparsertifikāti	Elektronisks dokuments, kurš apstiprina publiskās atslēgas piederību kādai personai vai struktūrai
Sertifikātu uzticamības saraksts *(Certificate Trust List – CTL)*	CTL ir sertifikātu jaucējfunkciju vērtību saraksts
Droša saziņa	Nodrošina TLS – drošligzdu slānis jeb protokols drošas un privātas saziņas nodrošināšanai Interneta tīklā
Serveru vārteju kriptogrāfija *(Server-Gated Cryptography)*	SGC izmanto finanšu konstitūcijas privātu dokumentu pārsūtīšanai Interneta tīkla vidē. Tas ir SSL paplašinājums, kas nodrošina 128 bitu šifrēšanu
Autentificēšanas metodes	IIS realizē *Microsoft Windows* autentificēšanu
Autorizācija jeb pilnvarošana	Pilnvaru piešķiršana kādai personai vai personu grupai noteiktu darbību izpildei un resursu izmantošanai
Audits	IIS uztur visu lietotāju piekļuves uzskaites žurnālu
Lietojumprogrammu pūls	IIS izolē atsevišķu tīmekļa lietojumprogrammu izpildi vienā izolētā procesā

IIS arhitektūra piedāvā pret kļūmēm noturīgu procesu modeli, kas ievērojami paaugstina tīmekļa vietņu un lietojumprogrammu uzticamību. Tagad IIS var izolēt atsevišķu tīmekļa lietojumprogrammu vienā izolētā procesā, ko dēvē par lietojumprogrammu pūlu un kas tieši

komunicē ar operētājsistēmas kodolu. Šie izolētie lietojumprogrammu pūli neļauj kādai lietojumprogrammai vai tīmekļa vietnei pārtraukt XML pakalpojumus vai citas tīmekļa lietojumprogrammas serverī. IIS nodrošina arī stāvokļa novērošanas (*health monitoring*) iespējas, lai varētu laikus atklāt un novērst tīmekļa lietojumprogrammu kļūmes.

.NET *Framework* piedāvā tīmekļa bāzes lietojumprogrammu un XML pakalpojumu veidošanas, ieviešanas un palaišanas modeli uz šīs ļoti stabilās platformas. Tā ir produktīva, standartizēta daudzvalodu vide, kas ļauj integrēt esošās investīcijas ar nākamās paaudzes lietojumprogrammām un pakalpojumiem, kā arī lietpratīgi atrisināt sarežģītos Interneta tīkla mēroga lietojumprogrammu ieviešanas un darbības uzdevumus.

Rekomendācijas IIS drošai lietošanai ir šādas:

- lietot NTFS, jo NTFS datņu sistēma ir drošāka par FAT;
- pārskatīt atļaujas tiesības piekļuvei mapēm.
  Pēc noklusējuma *Windows* izveido mapes un piešķir *Full Control* tiesības lietotāju grupai *Everyone*. Nepieciešams regulāri pārbaudīt atļaujas tiesības piekļuvei mapēm;
- izvietot izpildāmās programmas atsevišķās mapēs.
  Tas ļauj vieglāk piešķirt tiesības un veikt auditu (piem., *Visual Studio* projekta datnes izvietotas atsevišķās mapēs);
- pārbaudīt NTFS tiesības tīkla diskdziņos;
- strādāt ar minimālu servisu daudzumu.
  Jācenšas izmantot minimāli nepieciešamo servisu skaitu. Katrs papildu serviss var radīt drošības draudus;
- izmantot auditu;
- konfigurēt *IPSec*.
  Tas ir speciāls protokols, kas atšķirībā no agrākajām pieejām drošības problēmu risināšanai lietojumslānī ir izstrādāts, lai panāktu drošu pakešu apstrādi tīkla slānī;
- konfigurēt *Telnet* serveri.
  Telnet ir Interneta tīkla protokols, kas pieder pie protokolu TCP/IP saimes un ko izmanto attālai piekļuvei tīkla resursiem un termināļu emulēšanai. Šo protokolu bieži lieto, lai nodibinātu sakarus ar ziņojumdēļa sistēmu un lieldatoriem;
- neizmantot *NetBIOS* ar TCP/IP;
- pielietot DACL virtuālajām mapēm un IIS žurnāla datnēm.

SQL serveris ir vispusīgs, integrēts un noslēgts (*end-to-end*) datu risinājums, kas visiem lietotājiem sniedz plašas iespējas, jo tie saņem drošu, uzticamu un produktīvu platformu, uz kuras veidot datu un biznesa informācijas lietojumprogrammas.

SQL servera drošības pasākumu komponenti doti 2.2. tabulā.

2.2. tabula

SQL servera drošības nodrošināšanas komponenti

Komponents	Iezīmes
Šifrēšana	Paroles autentificēšanai tiek šifrētas ar *CryptoAPI* Komunikācija ar klientiem, realizēta ar SSL
Autentificēšana	*Windows* autentificēšana SQL autentificēšana
Uz lomām balstīta pilnvarošana *(Role-based authorization)*	Pilnvarotu lietotāju piekļuves privilēģiju ierobežošana
Audits	SQL Server atbilst neklasificētu IS drošības C2 jeb *EAL3* līmenim. Šis līmenis tiek definēts kā „metodiski testēts un pārbaudīts”, un tas ir pieņemams, kad nepieciešams vidējs drošības līmenis

Pastāv vairākas rekomendācijas SQL servera drošai lietošanai.

- Nelietot *Windows* autentificēšanu.
- Neizpildīt SQL serveri kā *LocalSystem*.
- Plānot paroļu un lomu piemērotību.
- Izvēlēties atbilstošus tīkla protokolus (TCP/IP, *Named Pipes*).
- Izdzēst nevajadzīgās sistēmas glabājamās procedūras (*Registry Access, Cmd Shell:Xp_cmdshell*) .
- Izdzēst nevajadzīgos ODBC (*Open Database* Connectivity) draiverus.
- Nekad nelietot tukšu vai viegli uzminamu paroli.
- Lietot tīkla trafika šifrēšanu.
- Šifrēt datu bāzes ar EFS (*Encrypting File System*) palīdzību.
- Atslēgt SQL *Mail*.
- Bloķēt 1433 pieslēgvietu.
- Izmantot audita iespējas.

COM (*Component Object Model*) – COM tehnoloģija jeb programmkomponentu objektmodelis. *Microsoft* attīstīta tehnoloģija komponentu izstrādāšanai. Programmu komponentus, kas izstrādāti, izmantojot šo specifikāciju, var asamblēt programmās, vai arī tos var pievienot tām esošajām programmām, kas tiek izpildītas operētājsistēmas *Microsoft Windows* vidē, lai paplašinātu šo programmu funkcionālās iespējas. COM komponentus parasti raksta valodā C++ un tos var no programmas atslēgt, nepārkompilējot pašu programmu, tās izpildes gaitā. COM tehnoloģija veido pamatu tādu *Microsoft* tehnoloģiju kā *OLE, ActiveX* un *DirectX* izstrādei.

COM tehnoloģijas drošības pasākumu komponenti doti 2.3. tabulā.

2.3. tabula

COM+ drošības nodrošināšanas komponenti

Komponents	Iezīmes
Uz lomām balstīta pilnvarošana	COM+ drošību var nodrošināt ar programmas kodā iebūvētajiem aizsardzības līdzekļiem vai administratīvā veidā – ar uz lomām balstītu pilnvarošanu. Tas ļauj uzdot drošības parametrus līdz pat lietotāja un metodes līmenim, t. i., noteikt konkrēta lietotāja piekļuvi konkrētiem resursiem. Ja lietotājam piemēro lomu, kas atbilst pieprasītajai metodei vai resursam, tad pieprasījums būs veiksmīgs, pretējā gadījumā pieprasījums netiks apstrādāts
Klienta autentificēšana	Klienta tiesību pārbaude nepieciešamā resursa izmantošanai
Klienta personifikācija un pilnvaru nodošana	COM+ paredz personifikācijas iespēju, kad konkrētam lietotājam tiek deleģētas noteiktas pilnvaras
Programmas iespēju ierobežošanas politika	Paredz ieviest uzticamības līmeņus piekļuvei datnēm, kuras lietotājs izmanto. Pārvalda ar *Local Security Policy* līdzekļiem.

Rekomendācijas COM+ drošai lietošanai.

- Katrai COM+ aplikācijai piešķirt savu kontu (*Service Account*).
- Neizpildīt COM+ aplikācijas kā *LocalSystem* vai izmantot *Interctive User* opciju.
- Izpildīt ar vismazākajām privilēģijām.
- Izmantot tikai nepieciešamos tīkla protokolu.
- Ierobežot DCOM (*Distributed Component Object Model*) pieslēgvietu diapazonu.
- Izmantot *Package* autentificēšanu.
- Komunikāciju drošības nodrošināšanai lietot *Packet Privacy*.
- Pārbaudīt visus lietotāja ievadāmos datus.

2.2. ACL un DACL izmantošana

Microsoft Windows piedāvā vairākus līdzekļus, ar kuru palīdzību tiek kontrolēta lietotāju piekļuve dažādiem datoru resursiem. Par standarta paņēmienu tiek uzskatīts piekļuves vadības saraksts ACL.

ACL (*Access Control List*) – piekļuves vadības saraksts, t. i., datoru lietotāju saraksts, kas norāda, kuriem lietotājiem ir atļauta piekļuve kādam no datoru resursiem un kā lietotāji šo resursu var izmantot.

Atļauja (*Permission*) – noteikumi, kas saistīti ar kāda datoru tīkla vai vairāklietotāju sistēmas konta izmantošanu un kas nosaka, kādi datoru tīkla vai sistēmas resursi ir pieejami konta lietotājam.

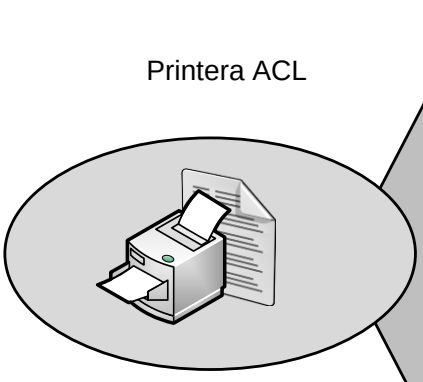
Katram resursu objektam pastāv divu veidu piekļuves vadības saraksti:

- DACL [16];
- SACL [17].

DACL (*Discretionary ACL*) – īpašnieka piekļuves vadības saraksts. Piekļuves vadības saraksts sastāv no atsevišķiem ierakstiem – ACE ievadnēm (*Access Control Entry*).

SACL (*System ACL*) – sistēmas piekļuves vadības saraksts. Apraksta darbības ar objektiem, kas pakļauti auditēšanai.

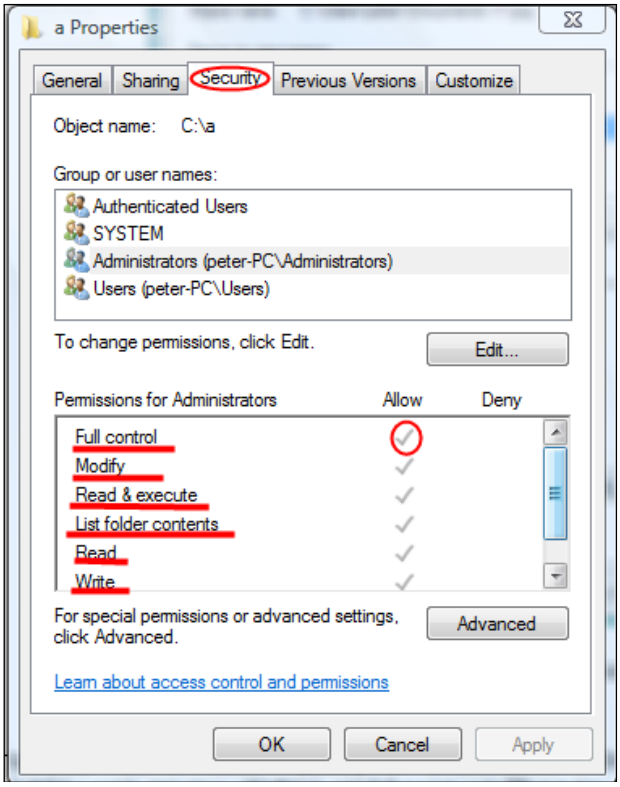
DACL sarakstā (sk. 2.1. att.) uzskaitīti lietotāji, kam ir piekļuves atļaujas tiesības un piekļuves veidi (bieži DACL vietā izmanto vispārīgāku terminu ACL).



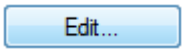
Lietotāja vārds	Atļauja	ACE
1. lietotājs	• Read • Write • Full Control	• Access_allowed • Access_allowed • Access_denied
2. lietotājs	• Read • Write • Full Control	• Access_allowed • Access_denied • Access_denied
3. lietotājs	• Read • Write • Full Control • Audit	• Access_allowed • Access_allowed • Access_allowed • Access_denied

2.1.att. **DACL saraksta piemērs**

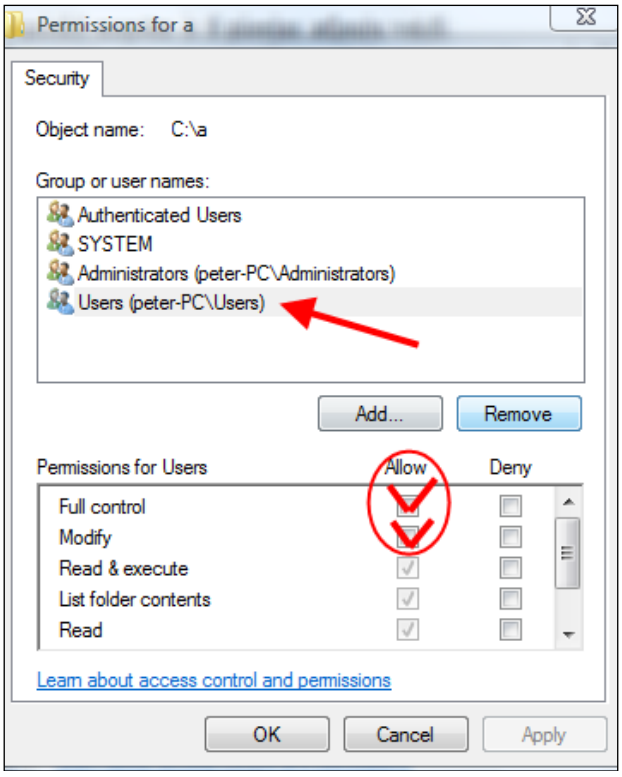
Katrai datnei un mapei NTFS datņu sistēmā ir savs piekļuves vadības saraksts DACL, kas sastāv no atsevišķiem ierakstiem ACE. Piemēram, mapei *C:\A* ir šāds DACL saraksts (sk. 2.2. att.).



2.2. att. **Mapes C:\A DACL saraksts**

Lai noteiktu specifiskas piekļuves atļaujas datnēm vai mapēm, pēc pogas 

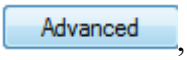
nospiešanas izvēlas vai pievieno konkrētu lietotāju vai lietotāju grupu un iestāda specifiskās *Allow* vai *Deny* atļaujas tiesības (sk. 2.3. att.).



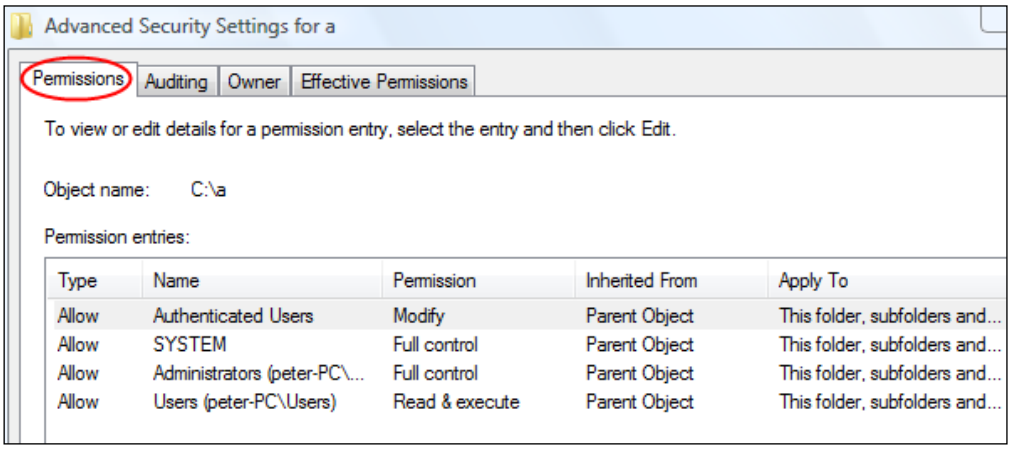
2.3. att. **Lietotāja piekļuves atļauju izmaiņa**

Vispārīgā gadījumā mapēm ir 6 piekļuves atļauju veidi:

1) *List Folder Contents* – lietotājs var tikai redzēt mapē esošās apakšmapes un datnes, bet tam nav piekļuves atļaujas;
2) *Read* – lietotājs var tikai redzēt konkrētā mapē esošās apakšmapes un datnes, kā arī mapes parametrus;
3) *Write* – lietotājs var veidot jaunas datnes un apakšmapes, izmainīt mapes atribūtus un parametrus;
4) *Read & Execute* – lietotājs var veikt visas iepriekšminētās darbības, kā arī piekļūt apakšmapēm;
5) *Modify* – lietotājs var veikt visas iepriekšminētās darbības, arī izdzēst mapi;
6) *Full Control* – lietotājs var veikt visas iepriekšminētās darbības, arī noteikt citu lietotāju piekļuves atļaujas, izdzēst apakšmapes un datnes.

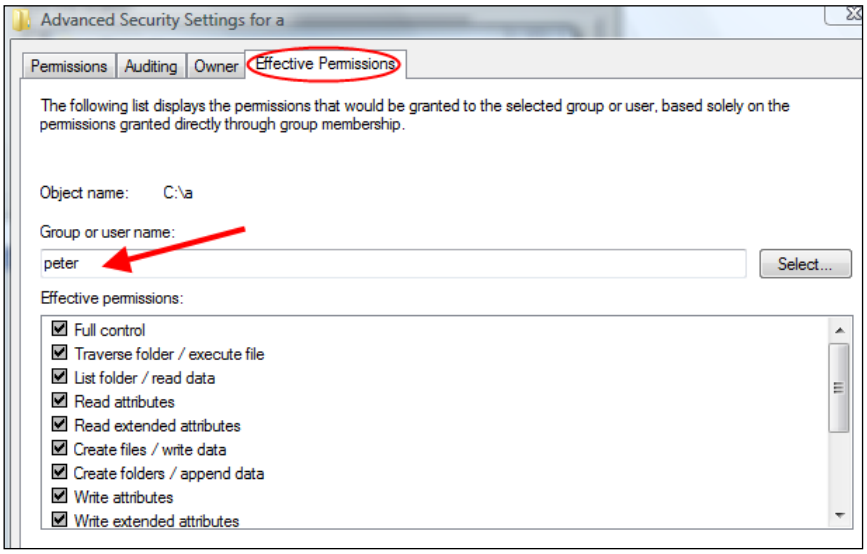
Nospiežot pogu 

, sadaļā **Permissions** redzams mapes pilnais DACL saraksts (sk. 2.4. att.).



2.4.att. **Mapes pilnais DACL saraksts**

Piekļuves atļaujas var iegūt vairākos veidos: piešķirot atļaujas tieši lietotājam, pārņemot atļaujas no pirmsākuma (*parent*) mapes, vai arī lietotājs saņem atļaujas no lietotāju grupas. Lai uzzinātu lietotāja reālās (*effective permissions*) piekļuves atļaujas no vairākām piešķirtajām atļaujām, ir jāapskata sadaļa **Efective Permissions** ar konkrētajam lietotājam piešķirtajām reālajām atļaujām (sk. 2.5. att).



2.5. att. **Lietotāja reālās piekļuves tiesības mapei**

Reālās piekļuves atļaujas tiek noteiktas pēc šādiem principiem:

- atļaujas tiek apvienotas un lietotājam tiek piešķirtas augstākās atļautās piekļuves atļaujas;
- aizliegtajām piekļuves atļaujām ir augstāka prioritāte nekā atļautajām piekļuves atļaujām;
- ja lietotājam nav piešķirtas nekādas piekļuves atļaujas, tad piekļuve ir liegta.

Ja NTFS datņu sistēmā tiek pārvietotas vai kopētas datnes vai mapes, tad jāņem vērā šādi principi:

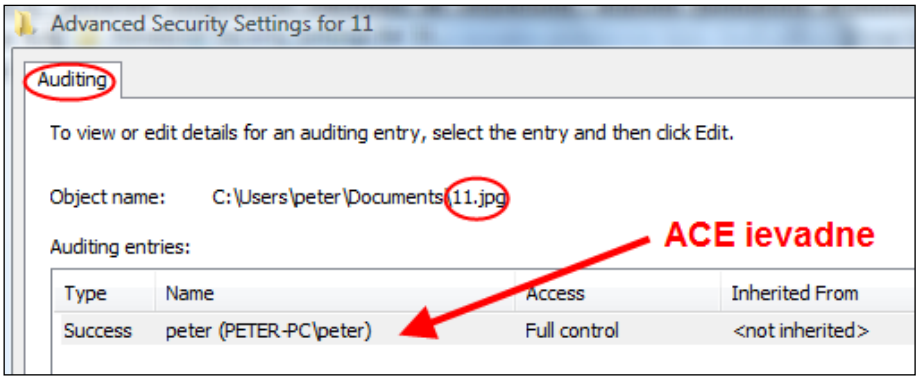
- ja objekti tiek kopēti, piekļuves atļaujas tiesības netiek saglabātas, tādējādi objekti pārņem jaunās mapes piekļuves atļaujas;
- ja objekti tiek pārvietoti, piekļuves atļaujas tiek saglabātas, izņemot tās atļaujas, kas pārņemtas no pirmsākuma (*parent*) mapes. Šīs pārņemtās atļaujas tiks aizvietotas ar jaunās mapes atļaujām;
- ja objekti tiek pārvietoti starp NTFS nodalījumiem (*partition*), piekļuves atļaujas netiek saglabātas un objekti pārņems jaunās mapes piekļuves atļaujas;
- ja objekti tiek pārvietoti uz FAT vai FAT32 datņu sistēmā formatētu disku vai nodalījumu, tad visa informācija par NTFS piekļuves atļaujām tiek zaudēta.

Attiecībā uz NTFS piekļuves atļaujām ir jāņem vērā šādi vispārīgie principi:

- piekļuves atļaujas tiek pārņemtas no pirmsākuma mapēm;
- piekļuves atļaujas ir kopējas, izņemot aizliegtās piekļuves atļaujas;
- piekļuves atļaujas tiek noteiktas lietotājiem un lietotāju grupām;
- lietotājs, kurš izveido datni vai mapi, kļūst par tā saucamo izveidotāju (*creator owner*) ar pilnas kontroles piekļuves atļaujām.

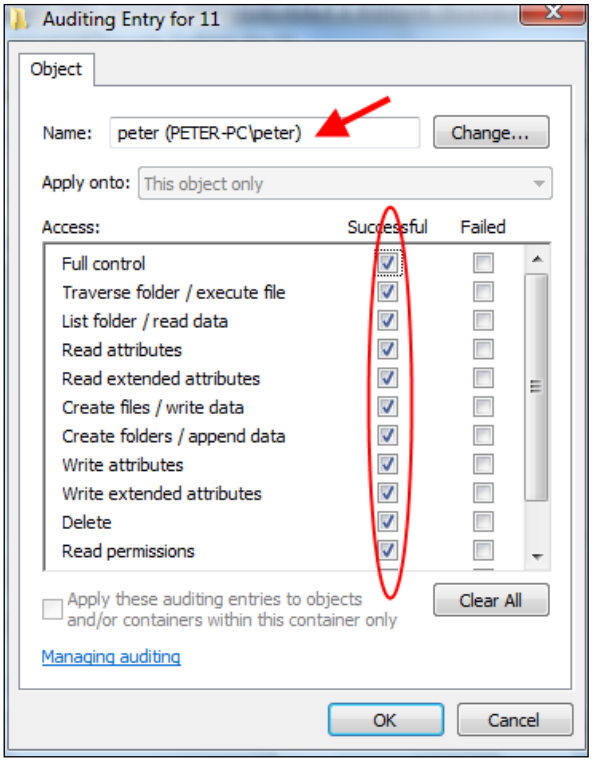
SACL sarakstā uzskaitītas darbības ar objektiem, kuriem piemērota *Windows* auditēšana. Objekta SACL sastāv no ACE ievadnēm, kas nosaka piekļuves tipu, kad lietotājs vai lietotāju grupa piekļūst objektam.

Piemēram, patvaļīgi izvēlētai datnei *11.jpg* SACL saraksts sastāv no vienas ACE ievadnes (sk. 2.6. att.).



2.6. att. **Datnes SACL saraksts**

2.6. attēlā redzams, ka *Windows* pārbaudīs lietotāja *peter* veiksmīgus mēģinājumus darbībās ar datni *11.jpg* (darbību saraksts parādīts 2.7. attēlā).



2.7. att. **Auditējamās darbības ar datni konkrētam lietotājam**

Kad auditēšana ir ieslēgta, informācija par visām sekmīgajām darbībām ar datni *11.jpg* tiks ierakstīta drošības žurnālā.

Lai iegūtu informāciju par ACL, *Visual Studio* vidē izmanto funkcijas `GetEffectiveRightsFromAcl`, `GetSecurityInfo`, `GetNamedSecurityInfo`, `GetExplicitEntriesFromAcl`.

Praksē ērti strādāt ar komandrindas programmu *Cacls*.exe (*Change Access Control List*), ar kuras palīdzību objektiem var izmainīt piekļuves atļaujas. Programmas parametri uzdoti 2.4. tabulā.

2.4. tabula

***Cacls* parametri**

Operācija	Parametrs
Izmaina ACL uzdotai datnei dotajā mapē	/T
Rediģē ACL sarakstu	/E
Turpina darbu, ja saņemts paziņojums *Access-denied*	/C
Nodrošina lietotājam piekļuvi datnei vai mapei. Parametrs *perm* var pieņemt vērtību R (*read*), W (*write*), C (change) vai F (*full control*)	/G user:perm
Atceļ visas lietotāja atļaujas	/R user
Nomaina lietotāja atļaujas	/P user:perm
Aizliedz piekļuvi datnei vai mapei	/D user

 2.1. piemērs

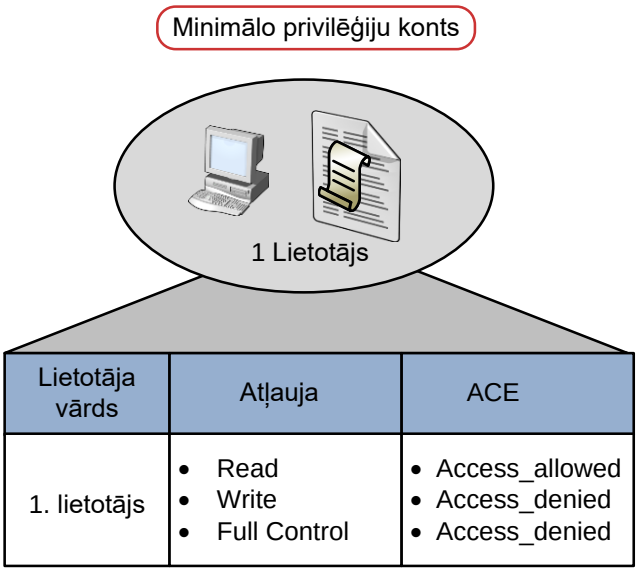
Cacls 11.jpg (parāda informāciju par lietotāju atļaujām piekļuvei datnei)

```
C:\a>cacls 11.jpg
C:\a\11.jpg BUILTIN\Administrators:(ID)F
            NT AUTHORITY\SYSTEM:(ID)F
            BUILTIN\Users:(ID)R
            NT AUTHORITY\Authenticated Users:(ID)C
```

2.3. Minimālo privilēģiju piešķiršana izmantojamajiem *Windows* kontiem

Lietojumprogrammas, kas projektētas pēc minimālo privilēģiju principa, savā darbā nepieprasa administratora konta atļaujas. Tas ļauj izpildīt lietojumprogrammas ar standarta lietotāja atļaujām, kas paaugstina lietotnes drošību, minimizējot potenciālo apdraudējumu.

Minimālo privilēģiju princips nosaka lietotājam piešķirt tikai tās privilēģijas, kas nepieciešamas tam darbu veikšanai. Vienkāršākais privilēģiju ierobežošanas veids ir ieiet sistēmā kā lietotājam bez administratora atļaujām (sk. 2.8. att).



2.8. att. **Minimālo privilēģiju konta piemērs**

Iemesli, kāpēc lietotājam ir nepieciešamas administratora atļaujas, var būt šādi:

- lietojumprogrammu uzstādīšana;
- drukas ierīces draiveru uzstādīšana;
- programmu ielāpu (*patch*) uzstādīšana;
- administratīvo uzdevumu izsaukšanas nepieciešamība.

Katrā no šiem gadījumiem ir nepieciešamas administratora privilēģijas. Tātad, ja lietotājam ir dotas administratora atļaujas kāda uzdevuma veikšanai, tad praktiski nepastāv iespējas ierobežot lietotāju, kas tādējādi ir ieguvis administratora atļauju piekļuvei visiem datora resursiem:

- kā administratoram skatīt Interneta tīkla vietnes;
- uzstādīt jebkādas lietojumprogrammas datorā;
- mainīt tīkla konfigurācijas parametrus;
- atslēgt datoru no domēna utt.

Nepieciešamība lietotājam iegūt administratora privilēģijas radusies vēsturiski, jo gandrīz vienmēr izstrādātājs lietojumprogrammu projektēšanu un testēšanu veic ar administratora privilēģijām. Nav nemaz tik daudz lietojumprogrammu, kas izstrādātas darbam ar minimālām privilēģijām. Tipiska visu lietojumprogrammu, kam nepieciešamas administratora privilēģijas, īpatnība ir arī tā, ka lietojumprogramma ieraksta informāciju par sevi būtiskajās sistēmas mapēs un reģistrā:

- *C:\Windows*
- *C:\Program files*
- *HKEY_Local_Machine*.

Analizējot lietojumprogrammas, kuru uzstādīšanai nepieciešamas administratora privilēģijas, izrādījās, ka aptuveni 90 % no *Windows* programmnodrošinājuma iespējams uzstādīt tikai ar administratora privilēģijām. Turklāt aptuveni 70 % gadījumu programmu palaišanai arī nepieciešamas administratora privilēģijas.

Microsoft lielu uzmanību pievērš šādu problēmu atrisināšanai, kas garantētu minimālo privilēģiju līmeņa modeļa ieviešanu. Izstrādāts lietotāja konta aizsardzības mehānisms UAP (*User Account Protection*), kas, piemēram, ietver vairākus uzdevumus, kuru palaišanai nav nepieciešamas administratora privilēģijas:

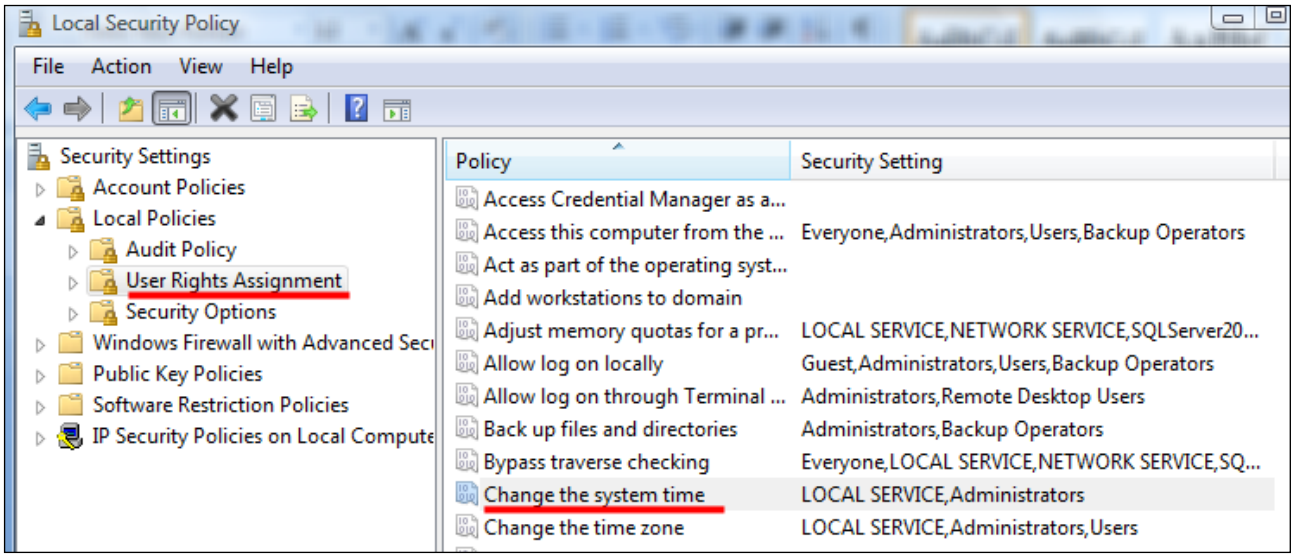
- portatīvo datoru lietotāji var uzstādīt WEP atslēgas, lai pievienotos bezvadu tīklam;
- lietotāji var uzstādīt drukas ierīču draiverus;
- lietotāji var lejupielādēt un uzstādīt programmu ielāpus;
- lietotāji var palaist arvien lielāku lietojumprogrammu skaitu.

Rekomendācijas minimālo privilēģiju principu izmantošanā ir šādas:

- nepieprasīt augstāku piekļuves līmeni resursiem nekā nepieciešams;
- izvietot datnes tur, kur tām var piekļūt standarta lietotāji.

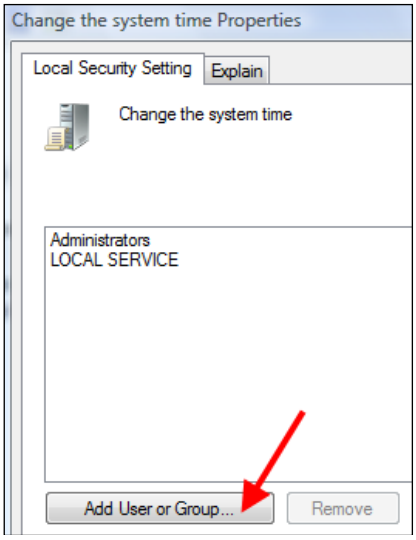
Dažādas Windows *Forms* vai ASP.NET lietotnes ieteicams projektēt darbam standarta lietotāja kontekstā, nevis sistēmas konta kontekstā, t. i., standarta lietotājam var piešķirt papildu privilēģijas. Ja lietojumprogrammas prasa atļaujas, kādu nav lietotāju grupai *Users*, nevajadzētu ļaut palaist šo lietojumprogrammu ar administratora privilēģijām. Tādos gadījumos administratoru grupas (*Administrators*) privilēģijas var deleģēt jebkuram standarta lietotājam. Tādējādi jebkuru lietojumprogrammu var palaist ar kontu, kas neietilpst administratoru grupā, bet kuram ir piešķirtas atsevišķas administratora privilēģijas.

Privilēģiju piešķiršana ietilpst sistēmas administratora funkcijās. Lai varētu pārbaudīt un mainīt lietotāju privilēģijas, jāveic šādu darbību secība: **Start / Control Panel / System and Security/ Administrative Tools / Local Security Policy**, tad jāatver mezgls **Local Policies / User Rights Assignment** (sk. 2.9. att.).



2.9. att. **Lietotāju tiesību saraksts**

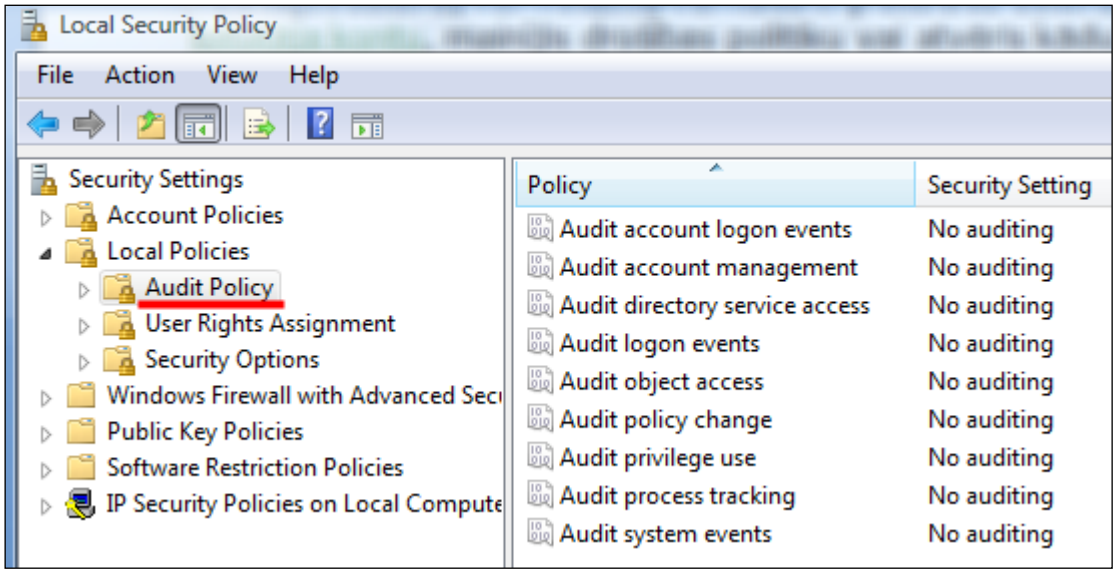
Izdarot dubultklikšķi, piemēram, uz atļaujas **Change the system time**, var pievienot papildu lietotāju, kam būtu tiesības mainīt sistēmas datumu un laiku (sk. 2.10. att.).



2.10. att. **Privilēģiju grupas papildināšanas ar jaunu lietotāju piemērs**

2.4. Audits

Auditēšana jeb pārraudzība palīdz uzturēt datora drošību. Auditējot datoru, var noteikt, vai kāds ir pieteicies datorā, izveidojis jaunu lietotāja kontu, mainījis drošības politiku vai atvēris kādu dokumentu. Auditēšana nepasargā no lietotāja izmaiņu veikšanas, kuram ir izveidots konts datorā, ar tās palīdzību var uzzināt to, kad izmaiņas tika izdarītas un kas tās veica. Ir vairāku veidu notikumi, kurus var pārraudzīt: konta pārvaldību, pieteikšanos, piekļuvi objektiem, politikas mainīšanu, sistēmas notikumus u.c. (sk. 2.11. att). Ja tiek pieņemts lēmums auditēt šos notikumus, *Windows* ieraksta notikumus žurnālā, kuru var apskatīt ar **Event Viewer**.



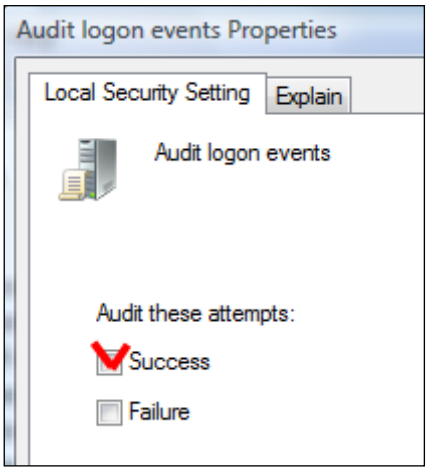
2.11. att. **Auditēšanas politikas saraksts**

Auditēšanas politikas drošības iestatījumi.

- Lietotāja pieteikšanās notikumu auditēšana (*Audit account logon events*). Ja lietotājs reģistrējas lokālajā darbstacijā ar domēna lietotāja kontu, tad darbstacija sūta pieprasījumu domēna kontrollerim lietotāja vārda un tiesību pārbaudei. Tas pats notiek, ja lietotājs pieslēdzas tīklā pie servera.
- Konta pārvaldības auditēšana (*Audit account management*). Tiek pārraudzīts, vai ir mainīts konta nosaukums, izveidots vai dzēsts konts, mainīta parole vai lietotāju grupa.

- Piekļuves direktorijai servisu auditēšana (*Audit directory service Access*). Tiek pārraudzīts, vai ir izdarīta piekļuve *Active Directory* objektam, kuram ir savs sistēmas piekļuves vadības saraksts SACL.
- Pieteikšanās notikumu auditēšana (*Audit logon events*). Tiek pārraudzīti pieteikšanās notikumi lokālajā sistēmā.
- Piekļuves objektiem auditēšana (*Audit object Access*). Tiek pārraudzīta datņu, mapju, reģistra atslēgu, printeru vai citu objektu izmantošana.
- Politikas izmaiņu auditēšana (*Audit policy change*). Tiek pārraudzīti mēģinājumi izmainīt lokālo drošības politiku un to, vai kāds nav mainījis lietotāja tiesību vai auditēšanas politikas uzstādījumus.
- Privilēģiju izmantošanas auditēšana (*Audit privilege use*). Tiek pārraudzīta lietotāja tiesību izmantošana.
- Procesu izsekošanas auditēšana (*Audit process tracking*). Tiek pārraudzīta procesu izpilde, piemēram, programmas aktivizācija vai iziešana no procesa.
- Sistēmas notikumu auditēšana (*Audit System events*). Tiek pārraudzīti ar sistēmas darbību saistītu procesu izpilde – datoru restarts, mēģinājumi neatļauti mainīt konfigurācijas iestatījumus utt.

Veicot dubultklikšķi, piemēram, uz auditēšanas **Audit logon events**, var iestatīt sekmīgas pieslēgšanās mēģinājumu auditēšanas procedūru (sk. 2.12. att.).

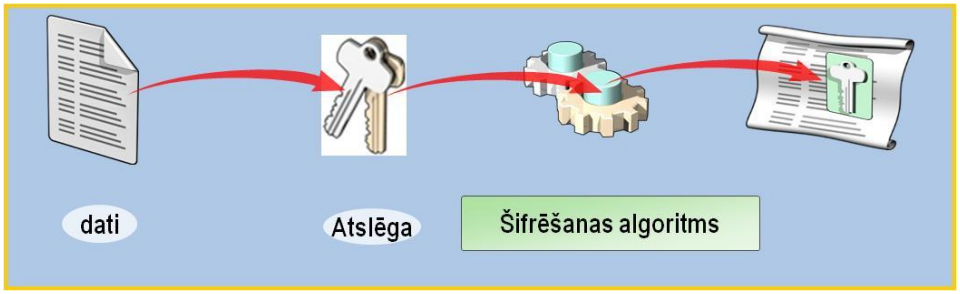


2.12. att. **Sekmīgu pieslēgšanās mēģinājumu auditēšanas iestatīšana**

2.5. Kriptogrāfijas iespēju izmantošana

Informācijas sistēmās datu aizsardzības pamatā ir datu šifrēšana. Šifrēšanas procedūras apskata kriptogrāfija – zinātne par šifrēšanu, kas pēta šifrēšanas metodes un līdzekļus [18].

Lai šāda kriptosistēma darbotos, ir nepieciešamas trīs lietas: šifrējamā informācija (atklātais teksts), šifra atslēga un pats šifrēšanas princips – likums, pēc kura notiek šifrēšanas process jeb atslēgas iedarbība uz atklātu tekstu. Šifrējot atklātu tekstu, iegūst šifrētu tekstu, ko iespējams atšifrēt, izmantojot to pašu atslēgu. Lai arī šāda veida kriptosistēmas jūtami palielina informācijas drošību, tomēr pastāv kāds ļoti būtisks trūkums – kā lai nogādā atslēgu droši? Ir nepieciešams kaut kāds pilnīgi drošs sakaru kanāls šīs slepenās atslēgas sūtīšanai. Ja ir pieejams šāds kanāls, tad kāpēc informācija vispār būtu jāšifrē? To visu būtu iespējams nosūtīt pa šo drošo kanālu. Visas kriptosistēmas, kurām piemīts šāds trūkums, mēdz dēvēt par slepenās atslēgas jeb simetriskās atslēgas kriptosistēmām (sk. 2.13. att.).



2.13. att. **Kriptosistēmas darbības princips**

Šifrēšana ir standartu un protokolu kopa datu un ziņojumu kodēšanai, lai tos varētu drošāk glabāt un pārsūtīt. Pat tad, ja pārsūtīšanas vide (piemēram, Interneta tīkls) nav droša, var izmantot slepenu datņu šifrēšanu, lai ļaunprātīgam lietotājam būtu mazāk iespēju tās saprast un lai nodrošinātu datu integritāti, kā arī saglabātu slepenību.

Var pārbaudīt šifrēto datu un ziņojumu izcelsmi, izmantojot ciparparakstus un sertifikātus. Izmantojot šifrēšanas metodes, šifrēšanas atslēgām jāpaliek slepenībā. Tomēr algoritmus, atslēgu izmērus un datņu formātus var darīt zināmus plašākai sabiedrībai, neapdraudot drošību.

Divas būtiskākās šifrēšanas darbības ir šifrēšana un atšifrēšana. Šifrēšana ietver datu kodēšanu tādā veidā, ka oriģinālās informācijas noteikšana kļūst neiespējama. Veicot atšifrēšanu, kodētie dati, izmantojot šifrēšanas atslēgu, tiek pārvērsti oriģinālajā tekstā.

Lai šifrētu un atšifrētu, vajadzīgs šifrēšanas algoritms un atslēga. Ir daudz šifrēšanas algoritmu, piemēram, datu šifrēšanas standarts DES (*Data Encryption Standard*), RSA (*Rivest/Shamir/Adleman*) šifrs, RC2 un RC5. Katrā no tiem tiek izmantota atslēga kopā ar algoritmu, lai pārvērstu vienkāršu tekstu (ko citi var izlasīt) ciparu tekstā (kas citiem nav izlasāms). Šifrus DES, RC2 un RC5 klasificē kā simetriskās atslēgas tehnoloģiju jeb slepenās atslēgas šifrēšanu, jo datu šifrēšanai un atšifrēšanai tiek izmantota viena un tā pati atslēga. Tātad atslēgai jābūt datus šifrējošās puses un datus atšifrējošās puses kopīgam noslēpumam.

RSA ir klasificēta kā publiskās atslēgas šifrēšana jeb asimetriskā šifrēšana, jo tajā tiek izmantotas divas atslēgas: publiskā atslēga un privātā atslēga.

Šifrēšana (*Encryption*) – datu un ziņojumu apstrādes process, ko veic datu sagatavotājs vai ziņojuma nosūtītājs, lai datus vai ziņojuma saturu nodrošinātu pret nesankcionētu izmantošanu. Lai šādus datus vai ziņojuma saturu varētu izmantot, jāveic tā atšifrēšana.

Publiskā atslēga (*Public Key*) – atslēga, ko publiskās atslēgšifrēšanas procesā lietotājs izplata saviem potenciālajiem korespondentiem un ko šie korespondenti izmanto, lai šifrētu lietotājam adresētos ziņojumus un atšifrētu lietotāja signatūru, kas šifrēta ar lietotāja privāto atslēgu.

Privātā atslēga (*Private Key*) – slepena atslēga, ko publiskās atslēgšifrēšanas procesā lietotājs neatklāj nevienam, bet izmanto, lai apzīmētu nosūtītos ziņojumus, kā arī atšifrētu saņemtos ziņojumus, kas šifrēti, lietojot publisko atslēgu.

Starp atslēgām pastāv matemātiska sakarība, bet nevar uzzināt vienu atslēgu, nezinot otru. Privātā atslēga tiek paturēta noslēpumā – tai jābūt pieejamai vienīgi atslēgu pāra radītājam. Publisko atslēgu var brīvi pārsūtīt, izmantojot nedrošu vidi, piemēram, Interneta tīklu. Lietojot publisko atslēgu sistēmas, abām pusēm nav kopīga noslēpuma. Ja datu šifrēšanai izmanto publisko atslēgu, datus var atšifrēt vienīgi ar privāto atslēgu. Līdzīgā veidā, ja datu šifrēšanai tiek izmantota privātā atslēga, datus var atšifrēt vienīgi ar publisko atslēgu.

RSA šifrēšanas algoritms ir mūsdienās plaši lietots asimetriskās šifrēšanas algoritms. Tas ir vēsturiski pirmais plaši lietotais asimetriskās šifrēšanas algoritms Ar to ātri var nošifrēt un atšifrēt

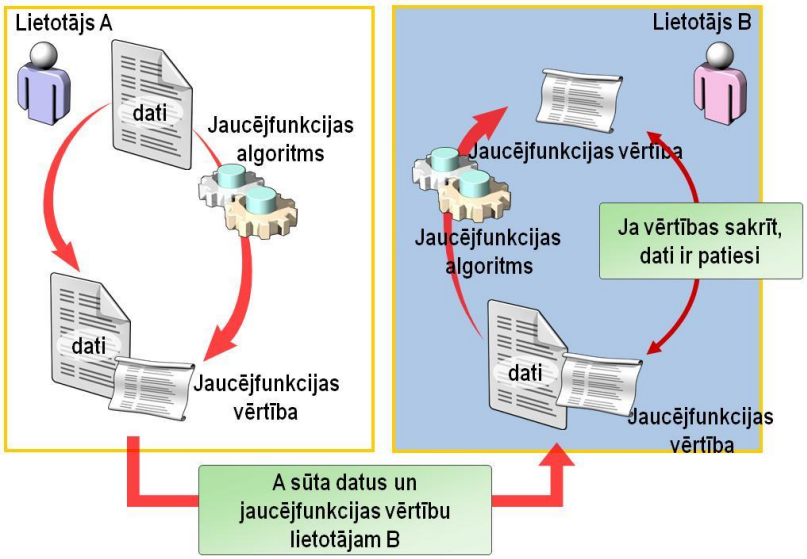
kādu tekstu, bet to ir praktiski neiespējami atšifrēt, nezinot slepeno atslēgu un aprēķinu veikšanai lietojot klasiskos datorus.

Variantu, kur datus šifrē ar publisko atslēgu (un var atšifrēt ar privāto), lieto datu šifrēšanai. Variantu, kur datus šifrē ar privāto atslēgu (un var atšifrēt ar publisko), lieto elektroniskajam parakstam.

Asimetriskie algoritmi (RSA) ir ievērojami lēnāki nekā simetriskie (AES), tāpēc, ja ir jāšifrē liels daudzums datu, ar RSA šifrē tikai simetrisko atslēgu un pašus datus – ar simetrisko šifrēšanu. Asimetriskajiem algoritmiem, lai sasniegtu tādu pašu drošības līmeni, ir nepieciešamas ievērojami garākas atslēgas. (AES simetriskās šifrēšanas algoritmam kaut cik drošs atslēgas garums ir 128 biti, RSA vajag vismaz 1024).

Ar šifrēšanu var aizsargāt datus, bet šifrētā teksta saņēmējs nevar būt drošs, ka sūtītājs ir tā persona, par kuru uzdodas. Personas autentificēšanai tiek izmantots jaucējfunkcijas algoritms (sk. 2.14. att.) vai ciparparaksts (sk. 2.15. att.).

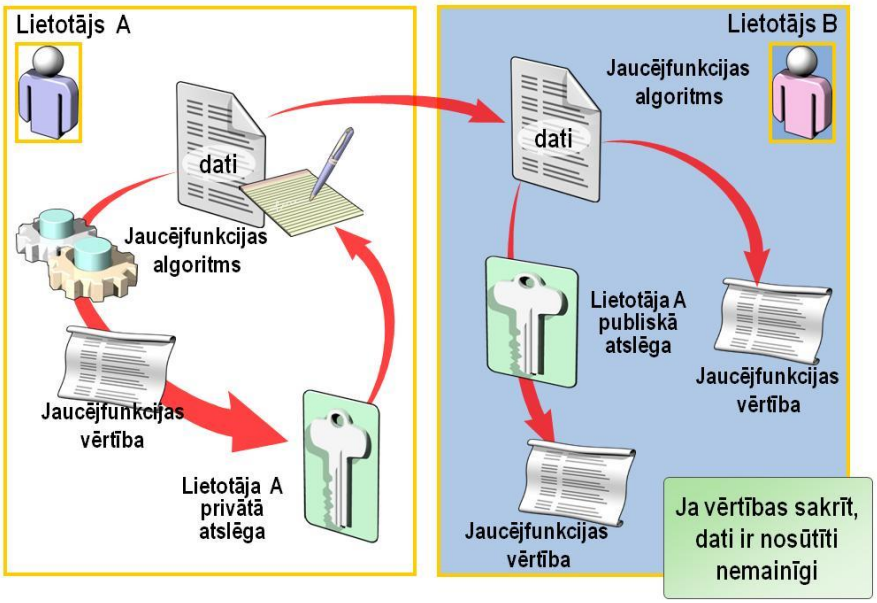
Jaucējfunkcija (*hash function*) – algoritms, kas pārveido dažāda garuma tekstu fiksēta garuma izvadē. Jaucējfunkcijas izmanto, lai veidotu, piem., ciparparakstus vai lai pārvērstu lietotājam nozīmīgu identifikatoru vai atslēgu par norādi, kādā noteiktas struktūras (tabulas) vietā šo informāciju var atrast. Terminu *hash* latviski varētu tulkot kā „jaucējfunkcijas vērtība".



2.14. att. **Datu integritātes pārbaude ar jaucējfunkciju vērtību palīdzību**

Izmantojot jaucējfunkcijas algoritmu, tiek izskaitļota dokumenta jaucējfunkcijas vērtība, ko bieži sauc par īssavilkumu (*digest*). Dokumentu kopā ar īssavilkumu aizsūta adresātam. Saņēmējs izveido no saņemtā dokumenta īssavilkumu un iegūto īssavilkumu salīdzina ar saņemto. Ja to vērtības sakrīt, tad var uzskatīt, ka dokuments ir pareizs.

Ciparparaksts (*Digital Signature*) – dati, kas pievienoti datu blokam vai arī ir iegūti, tos kriptogrāfiski pārveidojot, un kas ļauj datu saņēmējam pārliecināties par datu bloka integritāti un datu avota autentiskumu, kā arī nepieļauj to viltošanu.



2.15. att. **Ciparparaksta pielietošana**

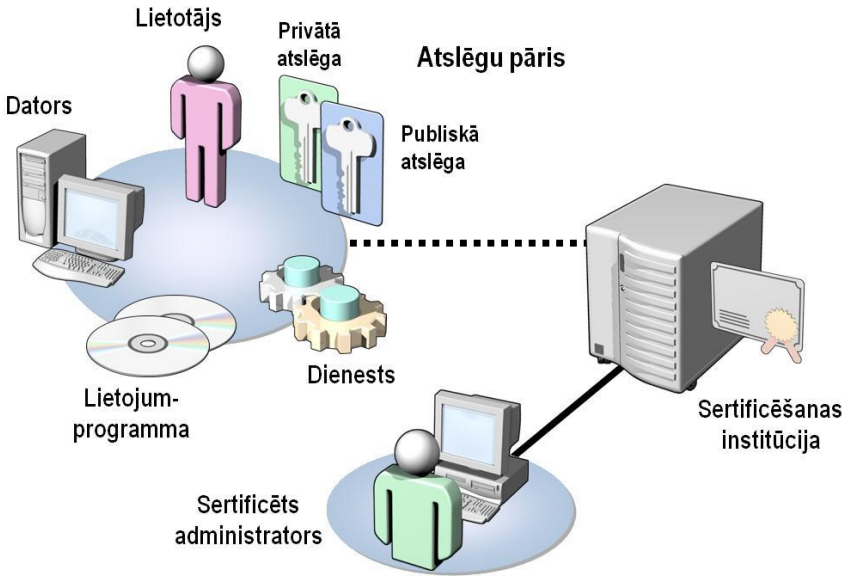
Ciparparaksta izveidošanai autors izmanto savu privāto atslēgu, kura zināma tikai autoram, bet ciparparaksta atšifrēšanai izmanto dokumenta autora publisko atslēgu. Ciparparakstu nevar izgriezt no cita dokumenta, jo kā sākotnējo tekstu ciparparaksta izgatavošanai izmanto dokumenta fragmentu. Saņemot šifrēto dokumentu ar ciparparakstu, dokumenta saņēmējs atšifrē ciparparakstu ar dokumenta autora publisko atslēgu, pārliecinās par autora autentiskumu un ar savu privāto atslēgu atšifrē dokumentu.

Ciparsertifikāts (*Digital Certificate*) – elektronisks dokuments, kurš apstiprina publiskās atslēgas piederību kādai personai vai citai vienībai. Ciparsertifikāts satur lietotāja publisko atslēgu un vārdu, kā arī lietotāja sertificēšanas institūcijas parakstu. Tas parasti satur arī derīguma termiņu, sertificētāja vārdu, lietotāja e-pasta adresi u. c. informāciju. Ciparsertifikāts, kuru izdevis uzticams sertificētājs (CA), ļauj pārliecināties, ka sūtījumu parakstītājs lieto to publisko atslēgu, kuru tam būtu jālieto.

CA (*Certificate Authority*) – sertificēšanas institūcija ir uzticama organizācija, kas izsniedz sertifikātus, ar to apliecinot, ka attiecīgajai personai pieder dotā publiskā atslēga, un tādējādi apstiprinot viņas identitāti.

Ciparsertifikātus var izsniegt organizācijas, kuras ir publiskās atslēgas infrastruktūras PKI locekļi (sk. 2.16. att.). PKI (*Public Key Infrastructure*) – publiskās atslēgas infrastruktūra, t. i., ciparsertifikātu izsniegšanas, sertificēšanas un citu reģistrējošo institūciju sistēma, kas Interneta tīkla transakcijās pārbauda un autentificē katra iesaistītā lietotāja identitāti. Pēdējā laikā droša PKI infrastruktūra tiek uzskatīta par galveno priekšnoteikumu plašai elektroniskās tirdzniecības lietošanai.

Ciparsertifikāts ir hierarhiska bināra veida struktūra, kas satur informāciju par publiskās atslēgas turētāju un tas atbilst X.509 sertifikācijas standartam. X.509 sertifikāts – publiskās atslēgas infrastruktūras standarts, kas specificē publiskās atslēgas sertificēšanas procedūru.



2.16. att. **Ciparsertificēšanas shēma**

X.509 sertifikāts, kā minimums, sastāv no šādiem laukiem:

- versijas numura – skaitļi 0,1 vai 2 , kas atbilst sertifikāta versijām 1, 2 vai 3;
- seriālā numura – unikāls identifikators;
- signatūras algoritma – algoritms, kuru izmanto ciparparaksta iegūšanai;
- sertifikāta izsniedzēja vārda;
- sertifikāta derīguma termiņa;
- privātās atslēgas īpašnieka vārda;
- publiskās atslēgas informācijas – satur publiskās atslēgas datus un algoritmu, kuru izmanto atslēgas iegūšanai;
- sertifikāta izsniedzēja unikālā identifikatora – tikai 2. un 3. versijā;
- sertifikāta īpašnieka unikālā identifikatora;
- paplašinājuma – sertifikāta datnes paplašinājuma (tikai 3. versijā);
- sertifikāta signatūras algoritma;
- sertifikāta signatūras.

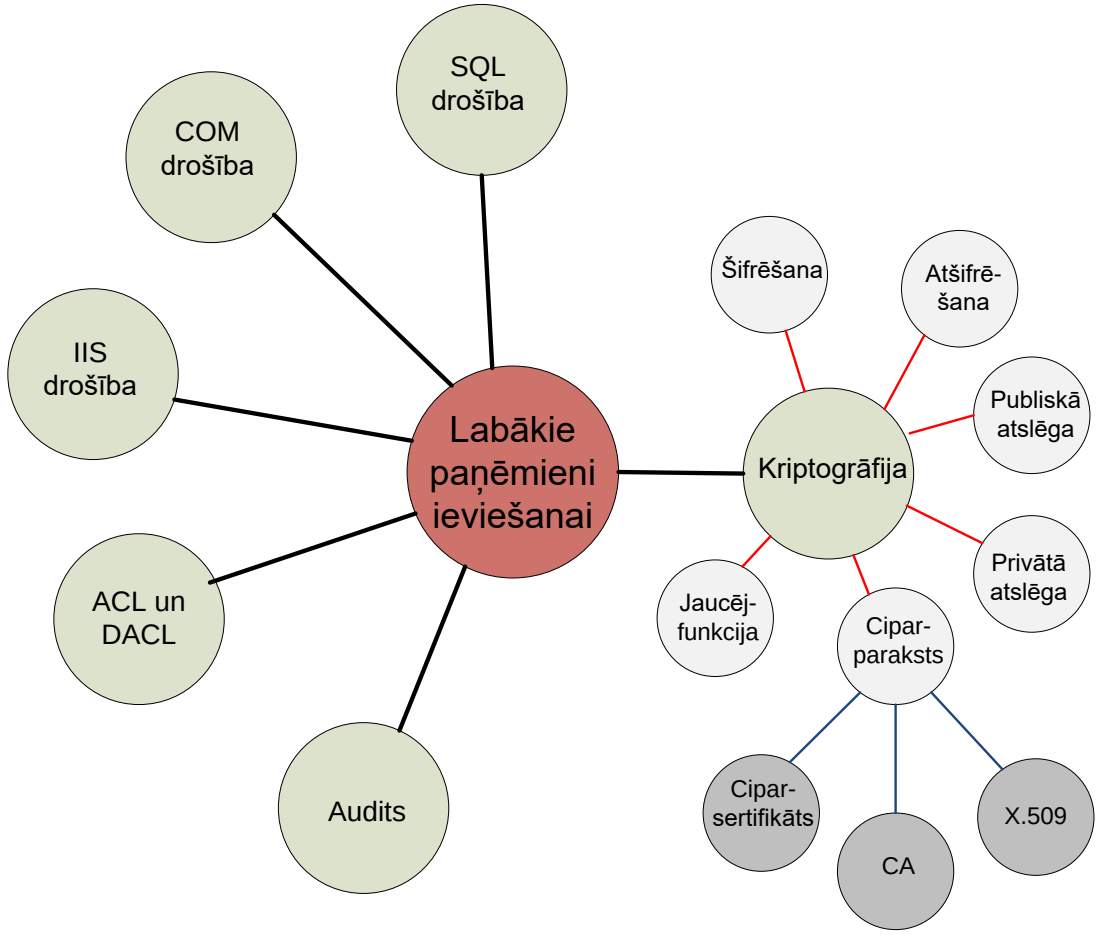
Sertifikāta datnes paplašinājumi var būt šādi: *.CER, .DER, .PEM, .P7B, .P7C, .PFX, .P12.*

2.2. piemērs. Ciparsertifikāta paraugs

```
Certificate:
   Data:
       Version: 1 (0x0)
       Serial Number: 7829 (0x1e95)
       Signature Algorithm: md5WithRSAEncryption
       Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
               OU=Certification Services Division,
               CN=Thawte Server CA/emailAddress=server-certs@thawte.com
       Validity
           Not Before: Jul  9 16:04:02 1998 GMT
           Not After : Jul  9 16:04:02 1999 GMT
       Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
OU=FreeSoft, CN=www.freesoft.org/emailAddress=baccala@freesoft.org

       Subject Public Key Info:
```

```
        Public Key Algorithm: rsaEncryption
        RSA Public Key: (1024 bit)
            Modulus (1024 bit):
                00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
                33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
                66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
                70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
                16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
                c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
                8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
                d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
                e8:35:1c:9e:27:52:7e:41:8f
            Exponent: 65537 (0x10001)
    Signature Algorithm: md5WithRSAEncryption
        93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
        92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
        ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
        d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
        0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
        5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
        8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
        68:9f
```

2.17. attēlā parādīts nodaļā minēto svarīgāko jēdzienu koks.



2.17. att. **Otrajā nodaļā minēto svarīgāko jēdzienu koks**

Pārbaudes testa jautājumi

1. Kurš no dotajiem apgalvojumiem ir patiess par ACL?
    a. Tas lieto matemātisku izteiksmi, lai noteiktu piekļuves atļauju
    b. Tas izveido auditēšanas paziņojumu, kad aizsargātais objekts saņem piekļuves pieprasījumu
    c. Tas pozicionē sarakstu *Authentication Clarification List*
    d. Tas ir tabulas veidā, kura identificē lietotāja piekļuves tiesības objektiem

2. Kurš no dotajiem lietotājiem var atļaut vai atcelt DACL piekļuvi objektiem?
    a. *Owner of the object*
    b. *Power users*
    c. *ASPNET users*
    d. *Backup users*

3. Administrējamās firmas darbinieks pieprasa piekļuvi attālinātam serverim. Kādas nepieciešamās darbības jāveic šādā gadījumā?
    a. Atļaut piekļuvi serverim, jo tas ir administrējamās firmas darbinieks
    b. Piešķirt tiesības un paļauties uz ugunsmūri
    c. Pārbaudīt firmas iesniegto darbinieku piekļuves tiesību sarakstu. Ja darbinieks iekļauts šajā sarakstā, atļaut piekļuvi serverim
    d. Atteikt piekļuvi serverim

4. Kas no uzskaitītā neietilpst datņu sistēmas piekļuves tiesībās?
    a. *Read*
    b. *Write*
    c. *Delete*
    d. *Execute*
    e. *Save*

5. Kādai lietotāju grupai ir vismazākās privilēģijas?
    a. Administrators
    b. *Users*
    c. *Power Users*
    d. *Guest*

6. ASP.NET lietojumprogramma darbojas standarta lietotāja kontekstā. Kādai grupai šis lietotājs pieder?
    a. *Debugger Users*
    b. *Power Users*
    c. *Users*
    d. *Guests*

7. Kādas no dotajām privilēģijām nav nepieciešamas lietotājam, kas izpilda ASP.NET lietojumprogrammu?
    a. *Deny logon locally*
    b. *Log on as batch job*
    c. *Manage auditing and security log*
    d. *Log on as a service*

8. Kādu grupu locekļi var noteikt vai modificēt auditēšanas politiku?
    a. *Power Users*
    b. *Users*
    c. Administrators
    d. *Backup Operators*

9. Kurus no dotajiem žurnāliem var apskatīt jebkurš lietotājs?
    a. *Security log*
    b. *System log*
    c. *Application log*
    d. Visus trīs
    e. Nevienu no trijiem

10. Kurš dotajiem ciparparaksta elementiem nodrošina datu integritātes pārbaudi?
    a. *Time stamp*
    b. *Middle initial*
    c. *Message digest*
    d. *Password hash*

11. Kādu datu modeli uztur X.509 sertifikāta standarts?
    a. *Web-of-Trust*
    b. *Gradational*
    c. *Fragmentary*
    d. *Hierarchical*

3. Labākie paņēmieni programmēšanai

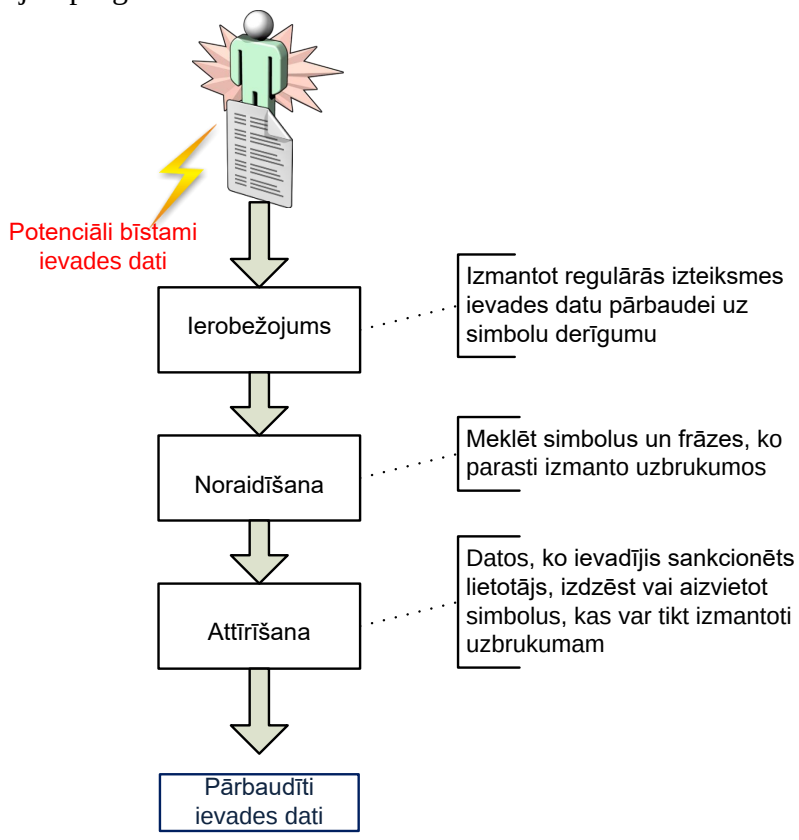
Kaut arī vispusīga lietotņu drošība ir sarežģīta tēma, tomēr var izslēgt daudzas drošības sistēmas ievainojamības, vienkārši izmantojot rekomendācijas par aizsargāta koda izstrādi. Viena no rekomendācijām, piemēram, ir lietotāja ievades datu pārbaude ar regulāro izteiksmju palīdzību.

Drošības nodrošināšana ir nepārtraukts process. Nodaļā apskatīta ievadāmo datu pārbaude katrā aizsardzības līmenī: koda izstrāde rindu pārbaudei ar regulāro izteiksmju palīdzību un datu izmēra pārbaude. Risināta koda izstrādes kārtība, lai novērstu kanonizācijas problēmas. Tāpat šajā nodaļā ir parādīta drošības izņēmumu izmantošana.

3.1. Ievades datu pārbaude

Visi nepārbaudītie ievades dati ir jāuzskata par apdraudējumu. Visiem datiem, kas saņemti no lietotāja vai no citas lietojumprogrammas, ieteicams veikt triju soļu pārbaudes procedūru, pirms tos varēs uzskatīt par drošiem (sk. 3.1. att.):

1) *ierobežojums*. Noraidīt ievades datus, ja tie neatbilst sagaidāmajam tipam, formātam, izmēram un diapazonam;
2) *noraidīšana*. Noraidīt ievades datus, ja tie atbilst noteiktām noraidīšanas prasībām un satur zināmus ļaundabīgus simbolus vai frāzes;
3) *attīrīšana*. Attīrīt ievades datus no simboliem, kas var būt kritiski lietojumprogrammai.



3.1. att. **Ievades datu pārbaudes process**

Ierobežojums

Ievades datu pārbaudes process jāsāk ar ierobežojumu uzstādīšanu datu tipam, formātam, izmēram un diapazonam. Pastāv vairāki atšķirīgi datu pārbaudes veidi, svarīgākie no tiem parādīti 3.1 tabulā.

3.1. tabula

Datu ierobežojuma iespējas

Pārbaudāmie dati	Pārbaudes veids
Tips	• Stingra tipa uzdošana • Regulārās izteiksmes • ASP.NET vadības elementi
Formāts	• Stingra tipu uzdošana • Regulārās izteiksmes
Izmērs	• Regulārās izteiksmes • Īpašība `String.Lenght`
Diapazons	• ASP.NET vadības elementi • Tipizētu datu salīdzināšana

Regulārās izteiksmes (*regular expressions* jeb *patterns*) ir īpašā sintaksē pierakstītas izteiksmes vienkāršāko valodu (t. s. regulāro valodu) aprakstīšanai [19]. Pats sarežģītākais regulāro izteiksmju izmantošanā – saprast to sintaksi. 3.2. tabulā doti svarīgāko metasimbolu apzīmējumi, kurus izmanto regulārajās izteiksmēs.

3.2. tabula

Regulārajās izteiksmēs izmantojamie simboli

Simbols	Apraksts
\	Atzīmē nākamo simbolu kā speciālo simbolu vai burtu
^	Nosaka rindas sākumu
$	Nosaka rindas beigas
*	Atbilst iepriekšējam simbolam vai apakšvirknei, kas ietilpst rindā no 0 līdz vairākām reizēm. Piem., zo* ir ekvivalents *z* un *zoo*
{n}	Nosaka, ka atbilstībai jābūt tieši *n* reizes. Piem., o{2} neatbilst simbolam *o* rindā *Bob*, bet atbilst diviem *o* rindā *food*
{n,m}	Nosaka, ka atbilstībai jāparādās, kā minimums, *n* reizes un, kā maksimums, *m* reizes. Piem., o{1,3} atbilst pirmajiem trijiem *o* rindā *foooood*. (n<=m)
.	Atbilst jebkuram simbolam (izņemot \n)
(šablons)	Atrod atbilstību šablonam un ieraksta to
x|y	Atbilst loģiskajam OR – *x* vai *y*. Piem., *z|food* atbilst *z* vai *food*. Izteiksme *(z|f)ood)* atbilst *zood* vai *food*
[xyz]	Simbolu kopa. Atbilst jebkuram no norādītajiem simboliem. Piem., [abc] atbilst simbolam *a* rindā *plain*
[^xyz]	Izslēdzošā simbolu kopa. Atbilst jebkuram simbolam, kas nav ietverts uzrādītajos. Piem., [^abc] atbilst simbolam *p* rindā *plain*
[a-z]	Simbolu diapazons. Atbilst jebkuram simbolam no uzrādītā diapazona. Piem., [a-z] atbilst jebkuram angļu alfabēta simbolam apakšējā reģistrā
[^a-z]	Izslēdzošais simbolu diapazons. Atbilst jebkuram simbolam, kas neietilpst uzrādītajā diapazonā
\d	Atbilst jebkuram ciparam. Ekvivalents [0-9]
\D	Atbilst jebkuram simbolam, kas nav cipars. Ekvivalents [^0-9]
\n	Atbilst simbolam „Pāreja uz jaunu rindu”. Ekvivalents \x0a un \cJ
\r	Atbilst simbolam „Rakstatgrieze” (CR). Ekvivalents \x0d un \cM
\s	Atbilst jebkuram „Tukšās kopas” simbolam, ieskaitot intervālu un tabulācijas simbolu.
\S	Atbilst jebkuram simbolam, kas nav „Tukšās kopas” elements
\w	Atbilst jebkuram angļu alfabēta simbolam, ciparam vai pasvītrojuma simbolam. Ekvivalents izteiksmei [A-Za-z0-9_]
\W	Atbilst jebkuram simbolam, kas nav angļu alfabēta simbols, cipars vai pasvītrojuma simbols. Ekvivalents izteiksmei [^A-Za-z0-9_]

Regulārās izteiksmes lieto ievades datu salīdzināšanai ar šablonu (*pattern matching*), šablona meklēšanai (*pattern search*) vai tā aizvietošanai (*pattern substitution*). Ar regulārajām izteiksmēm var atrisināt daudzveidīgus datu apstrādes uzdevumus. Regulārās izteiksmes tiek plaši izmantotas *UNIX* vidē. Vairums *Microsoft* rīku tās vispārīgā veidā neatbalsta, izņemot jaunākos, piemēram, .NET *Framework* vai *Internet Explorer*, kas paredz *JavaScript* ar *Perl* stila regulārajām izteiksmēm.

3.1. piemērs. Regulārāro izteiksmju piemēri

^\d{5}$	5 ciparu skaitlis. Pareizās izteiksmes: 25641 21398. Nepareizās: A674 9812
{1,10}	Rindas garumā no 1 līdz 10 simboliem. Pareizās izteiksmes: „Internet” „Elephant”. Nepareizās: „I am invalid”
\d+	Ietver skaitļus, kas >0
^[A-Z]+[a-z]{3,20}$	Vārdi, kas sākas ar lielo burtu (garumā no 3 līdz 20 simboliem). (latviešu simboliem – jāpapildina saraksts [a-z,ā,č,ē,ģ…]. Ar lielajiem burtiem līdzīgi)
^[\+371-]+[0-9]{8}$	LV telefonu numuri
^\(\d{3}\)\d{3}-\d{4}$	ASV telefonu numuri. Pareizās izteiksmes: (234)-521-7654 512-765-2345. Nepareizās: 76-1-2456 (8)-(23)-234
^\d{5}(-\d{4}){0,1}$	ASV ZIP kodi
^LV-+[0-9]{4}$	LV pasta indeksi
\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*	e-pasta adrese. Pareizās izteiksmes: test@contoso.msft mymail@contoso.msft Nepareizās: mymail@contoso mymail.msft@contoso
^([\w-\.]+)@((\[0-9]{1,3}\.[0-9]{1,3}\.)|(([\w-]+\.)+))([a-zA-Z]{1,4}|[0-9]{1,3})(\]?)$	Vēl viens e-pasta pārbaudes variants
http(s)?://([\w-]+\.)+[\w-]+(/[\w- ./?%&=]*)?	Internet URL

Ievades datu pārbaudei ieteicams izmantot regulārās izteiksmes, kas ir ļoti efektīvs līdzeklis lietotņu drošības nodrošināšanai. Ja, piemēram, lietojumprogrammā jāievada piecciparu skaitlis, tad var izmantot regulāro izteiksmi, lai atļautu ievadīt 5 simbolus diapazonā no 0 līdz 9.

.NET *Framework* klases, kas veic regulāro izteiksmju analīzi, atrodas vārdu telpā (*namespace*) ar nosaukumu `System.Text.RegularExpressions`.

3.2. piemērs. Regulāro izteiksmju pārbaude

```
using System;
using System.Text.RegularExpressions;
namespace regular1
{
   class Program
   {
      static void Main(string[] args)
      {
         if (Regex.IsMatch(args[1], args[0]))
         {
```

```
            Console.WriteLine("Ir regulāra izteiksme");
            Console.ReadLine();
        }
        else
        {
            Console.WriteLine("Nav regulāra izteiksme");
            Console.ReadLine();
        }
    }
  }
}
```

Komentāri

Konsoles aplikācijā kā pirmo parametru uzdod regulāro izteiksmi, kā otro – pārbaudāmo datu rindu.

Rezultāts

```
regular1 ^\d{5}$ 1234
Nav regulara izteiksme

regular1 ^\d{5}$ 12345
Ir regulara izteiksme
```

Lietotāji bieži datus uzdod rindu veidā. Piemēram, ASP.NET formā ievadītos datus var iegūt ar rindas `System.Web.UI.WebControls.TextBox.Text` palīdzību, bet datus no *Windows Form* – ar rindas `System.Windows.Forms.TextBox.Text` palīdzību. Ja lietotājs neievada teksta datus, tad jāveic tipu pārveidošana. Ja, piemēram, lietojumprogrammā ir vadības elements **TextBox**, kurā var ievadīt skaitli no 1 līdz 100, tad šo ievades datu pārveidošanu un diapazona pārbaudi var veikt ar 3.3. piemērā dotā koda fragmenta palīdzību. Šis paņēmiens ir ļoti efektīvs nepieļaujamu ievaddatu filtrēšanai.

3.3. piemērs

```
int myNumber = Int16.Parse(TextBox1.Text);
if ( !(( myNumber >=1) && (myNumber <=100)))
Throw New Exception ("Nepareizs ievads: skaitlis ārpus diapazona");
```

Arī ievadāmo rindu garuma pārbaude var būt ļoti lietderīga lietojumprogrammās. Ja ievaddatiem ir lielāks izmērs nekā paredzēts programmā, lietojumprogramma var tikt pakļauta bufera pārpildes situācijai. Lietojumprogrammas uz .NET *Framework* pamata šajā ziņā ir drošas, jo tiek veikta automātiska pārbaude – vai buferis spēj saturēt tajā ierakstāmo informāciju.

Tomēr dažkārt nepieciešams pārbaudīt ievadāmo datu garumu. To var darīt, piemēram, šādā veidā: `if (input.length >20) throw new Exception(„Ievades rinda pārāk gara”).`

Jebkura lietojumprogramma, kurā lietotājam jāievada dati, var būt pakļauta ļaunprātīgām darbībām, un tieši tīmekļa pielietojumi cieš no tā visvairāk. Šī iemesla dēļ ievades datu pārbaudei ASP.NET lietojumprogrammās ir ļoti liela nozīme. Drošības nodrošināšanai var izmantot pieejamos .NET *Framework* līdzekļus, ieskaitot regulārās izteiksmes.

ASP.NET satur papildu līdzekļus – piecus vadības elementus vārdu telpā `System.Web.UI.WebControls`, kurus izmanto datu pārbaudei:

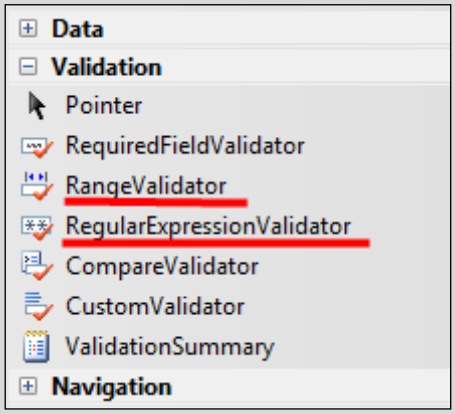
1) `RequiredFieldValidator` – pārbauda laukus, kas satur vismaz vienu simbolu;
2) `CompareValidator` – pārbauda, vai dati vienā laukā pilnībā atbilst datiem citā laukā. Parasti izmanto datu identiskuma pārbaudei laukos *Password* un *Confirm Password*;
3) `RangeValidator` – pārbauda, vai dati ietilpst noteiktā simbolu diapazonā. Var pārbaudīt skaitļus, datumus un simbolus;

4) `RegularExpressValidator` – pārbauda ievades datu atbilstību uzdotajai regulārajai izteiksmei;
5) `Customvalidator` – ļauj veidot savas ievades datu pārbaudes metodes.

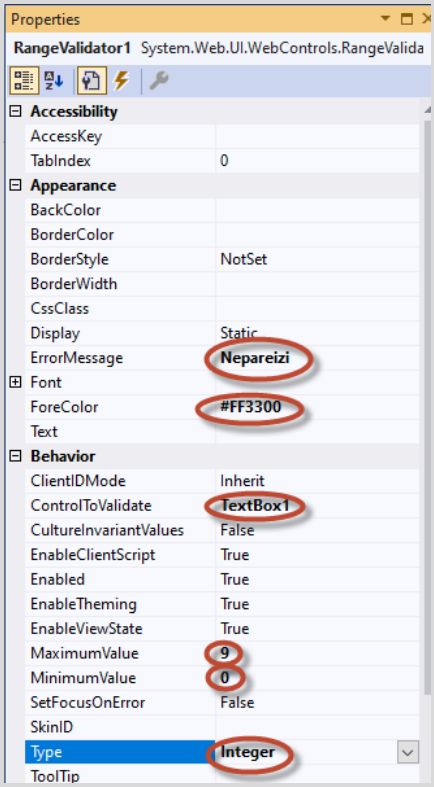
No pieciem vadības elementiem potenciāli ļaunprātīgo datu filtrēšanai visnoderīgākie elementi ir `RangeValidator, RegularExpressionValidator un CustomValidator`.

3.4. piemērs. *`RangeValidator`* un *`RegularExpressionValidator`* izmantošana (forma, kurā tiek pārbaudīta ievadītā skaitļa, e-pasta un vārda pareizība)

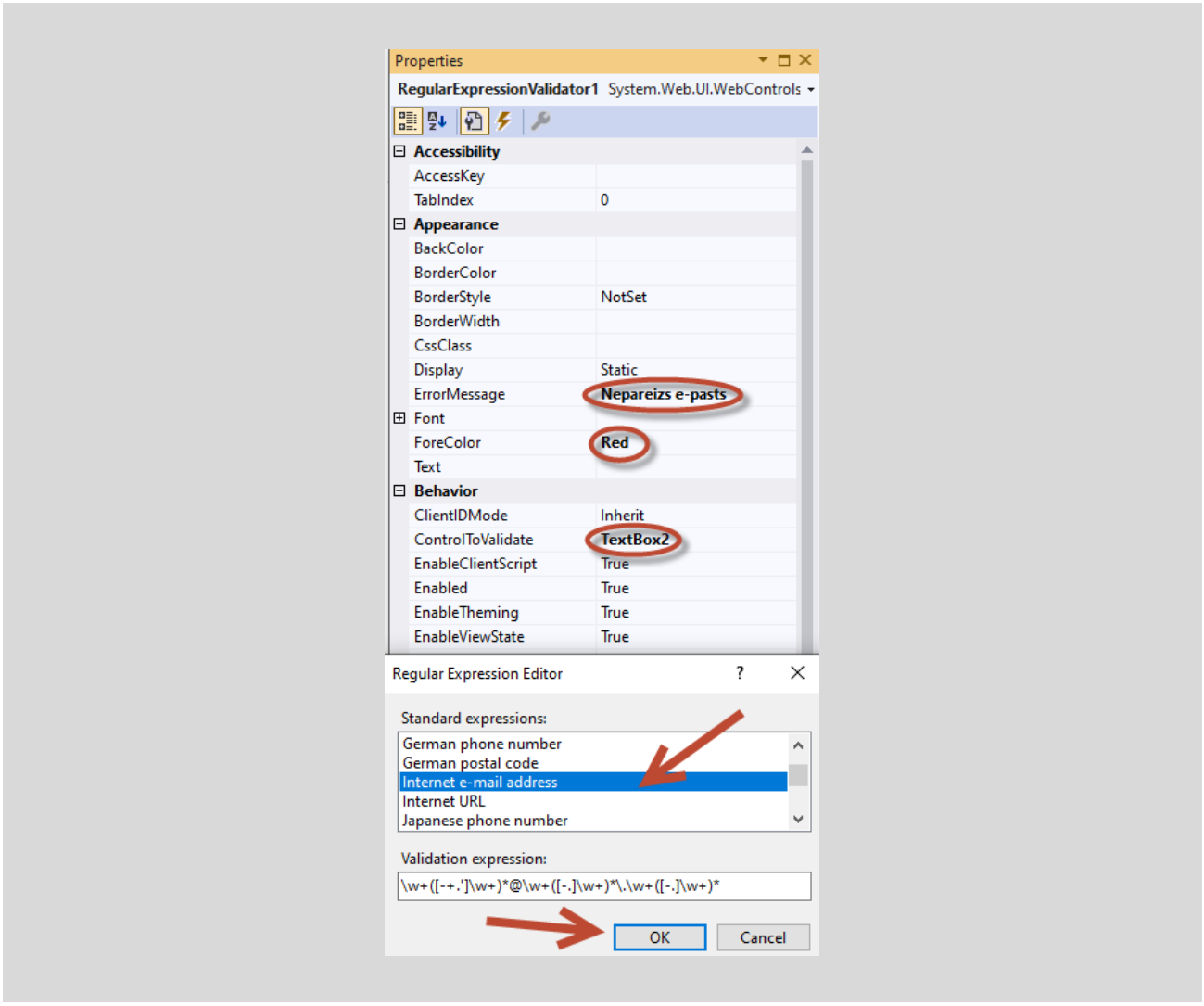
1. ***Visual Studio*** **vidē izveido jaunu** ***ASP.NET Web Application*** **(*.NET Framework*).**
2. Izvēlas ***Web Forms,*** **tad** ***Default.*****aspx. Logā** ***Source*** **izdzēš šablona formu un pāriet uz** ***Design*** **logu. Formā ievieto trīs vadības elementus:** ***TextBox*** **(vienu – skaitļu 0–9 ievadei, otru – e-pasta ievadei, trešo – vārda ievadei).**
3.Elementam ***TextBox1*** **pievieno vadības elementu** ***RangeValidator*** **un** ***TextBox2, TextBox3*** **–** ***RegularExpressionValidator*****.**

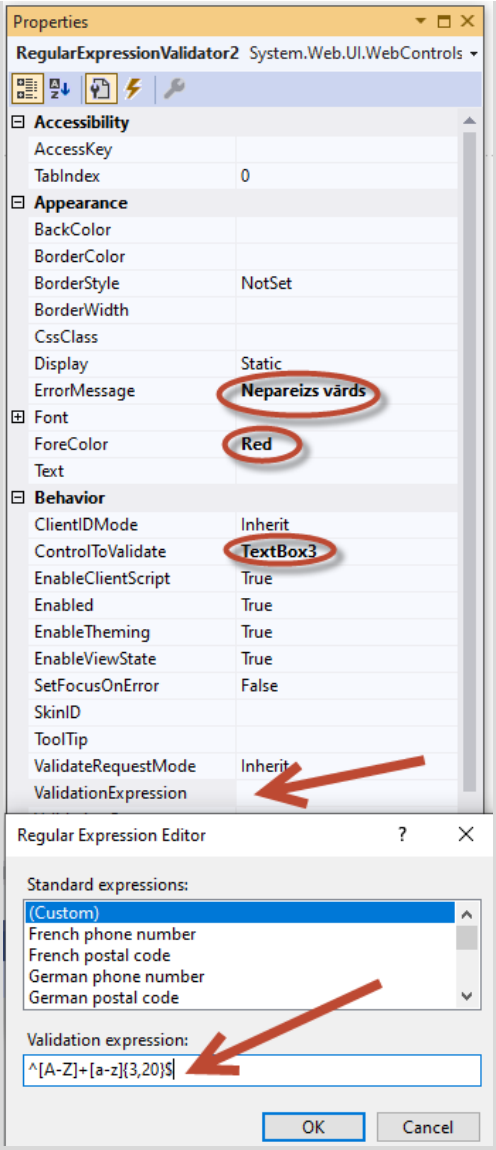


4. Pamaina ***RangeValidator*** **īpašības:**

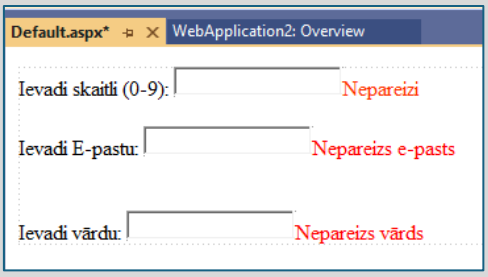


5. Pamaina ***RegularExpressionValidator*** **īpašības** ***TextBox2*** **un** ***TextBox3*****:**

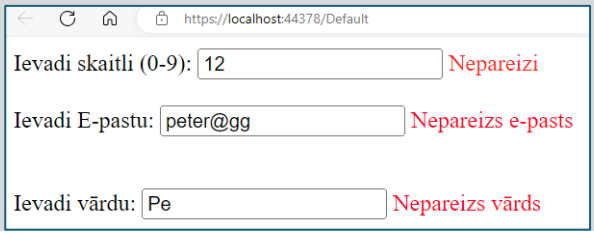

Properties
RegularExpressionValidator1 System.Web.UI.WebControls
Accessibility
AccessKey
TabIndex 0
Appearance
BackColor
BorderColor
BorderStyle NotSet
BorderWidth
CssClass
Display Static
ErrorMessage Nepareizs e-pasts
Font
ForeColor Red
Text
Behavior
ClientIDMode Inherit
ControlToValidate TextBox2
EnableClientScript True
Enabled True
EnableTheming True
EnableViewState True
Regular Expression Editor
Standard expressions:
German phone number
German postal code
Internet e-mail address
Internet URL
Japanese phone number
Validation expression:
\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*
OK
Cancel



6. Iegūta šāda tīmekļa aplikācijas forma, kur pirmajā laukā tiek pārbaudīts ievadīto skaitļu diapazons (no 0 līdz 9), otrajā – e-pasta ievades pareizība, trešajā – vārda pareizība:



7. Rezultāts:



Katrs no šiem vadības elementiem ietver gan klienta, gan servera komponentu. Klienta komponents darbojas lietotāja pārlūkprogrammā *JavaScript* koda veidā. Klienta scenārija izmantošana nepalielina lietojumprogrammas aizsardzības līmeni. Pārbaude klienta pusē dod tūlītēju atgriezenisko saiti ar lietotāju, kurš nejauši ievadījis nepareizus datus. Taču ļaunprātīgie lietotāji var vienkārši izlaist klienta scenāriju izpildi.

Savukārt servera pārbaudes elementa komponents paaugstina aizsardzības līmeni. Ja datus, kas tiek ievadīti tīmekļa lapā, pārbauda vadības elements, tad ASP.NET pārbauda ievadītos datus pirms vadības atgriešanas programmas kodam. Tomēr ASP.NET neapturēs ievades datu apstrādi, ja tie neatbildīs vadības elementā uzdotajiem nosacījumiem. Tāpēc programmētājam papildus jāveic šī vadības elementa īpašības `IsValid` pārbaude. Ja tās vērtība ir *true*, tad ievades dati veiksmīgi izturējuši pārbaudi, ja vērtība ir *false* – nevajadzētu uzticēties ievadītajiem datiem.

Noraidīšana

Kā tika noskaidrots, labākais veids ievaddatu pārbaudē ir pārbaudīt to tipu, formātu, izmēru un diapazonu. Tomēr šis paņēmiens neļauj pilnībā atklāt visus potenciāli bīstamos datus. Tāpēc pēc datu ierobežošanas nākas meklēt noteikta tipa apdraudošos datus. Piemēram, šī regulārā izteiksme ļauj atklāt scenāriju HTML ievaddatos: *<\s*script\s*>*.

Datu, kuri atbilst šādai regulārajai izteiksmei, noraidīšana apgrūtinās uzbrukumus, kas izmanto starpvietņu skriptošanas paņēmienus. Ļaunprātīgiem lietotājiem nāksies atrast tādu scenārija formātu, kas neatbildīs izmantotajai regulārajai izteiksmei. Piemēram, minētā regulārā izteiksme atklās tekstus *<script>*, *< script >* un *<script language="JavaScript">*, bet palaidīs garām tekstu *<SCRIPT>*. Ļaunprātīgs lietotājs var ātri konstatēt, ka filtrēšanas metode ir jūtīga attiecībā uz simbolu reģistru, un izmantot šo nepilnību uzbrukumos.

Vispārīgā gadījumā ievades datu ar saturu, kuru var identificēt kā ļaundabīgu, noraidīšana dod iespēju mazināt uzbrukuma draudus, taču par pilnībā drošu to uzskatīt nevar. Jebkurā gadījumā noraidīšana jāpielieto tikai pēc stingras ievaddatu ierobežošanas.

Attīrīšana

Daudzos gadījumos pieļaujamie simboli var tikt izmantoti uzbrukumu veikšanā. Piemēram, ļaunprātīgi lietotāji var izmantot apostrofa simbolu uzbrukumos ar SQL injekciju izmantošanu, jo šis simbols ir atdalītājs SQL komandās. Tomēr, mēģinot no tā izvairīties, nedrīkst noraidīt visus apostrofa simbolus saturošos datus, jo tie var būt, piemēram, dažādos uzvārdos vai ģeogrāfiskajos nosaukumos.

Tādos gadījumos nepieciešams izslēgt šāda tipa simbolus – nomainot tos ar speciāliem drošiem simboliem. Piemēram, SQL divi apostrofa simboli pēc kārtas tiek interpretēti kā viens, kas nav atdalītājs. Tādējādi, nomainot vienu apostrofa simbolu uz diviem, būs iespējams ielikt vārdus dinamiski ģenerējamos SQL pieprasījumos – nepieļaujot apostrofa simbola izmantošanu kā atdalītāju. Nomaiņu var veikt šādā veidā: *output = input.Replace("'", "'')*. Mainīt vai dzēst bīstamos simbolus var arī ar regulāro izteiksmju palīdzību. Piemēram, komanda *Regex.Replace(input,"[^\w\.@-]","")* nodzēsīs visus simbolus, kas nav burti vai cipari, izņemot simbolus @,- un . (punkts).

Analoģiski komandas *comment=comment.replace("<","&lt")* un

comment=comment.replace(">","&gt")

nomainīs simbolu < uz *&lt* (> simbolu uz *&gt*). Pārlūkprogrammas interpretē simbolus < un > kā HTML koda vai klienta scenāriju atdalītājus un tos neattēlo. Lai attēlotus simbolus „mazāks" un 'lielāks", tiek izmantoti simboli *&lt* un *&gt*. Tādējādi šīs komandas apgrūtinās starpvietņu skriptošanas veikšanu.

3.2. Kanonizēšanas problēmu novērtēšana

Uzbrukumi, kuros izmanto kanonizēšanas kļūdas tīmekļa lietojumprogrammās, ir un būs viena no lielākajām problēmām izstrādātājiem. Tomēr kanonizēšanas problēmas var apdraudēt arī *Windows Forms* lietojumprogrammas, jo lokālo datņu vārdi var tikt uzdoti dažādos veidos.

Ja datnes vārds tiek pārbaudīts, pirms to ir pārbaudījusi operētājsistēma, ļaunprātīgs lietotājs var uzdot datnes vārdu, ko lietojumprogramma neuzskatīs par neatļautu. Aizsargāties pret uzbrukumiem, kas izmanto kanonizācijas kļūdas, var palīdzēt ceļa kanonizācija pirms tā pārbaudes. Turklāt ir jāparūpējas par to, lai ceļš saturētu tikai atļautos simbolus.

Par kanonizēšanu jau tika minēts 1.1. apakšnodaļā. Kanonizēšana ir ceļa vienkāršošanas process līdz vienkāršai absolūtai formai. Jebkuru datni var identificēt vairākos veidos, piemēram, *C:\boot.ini, ..\..\..\BOOT.INI* vai *C:/boot%2eini*, tāpēc operētājsistēmai ir nepieciešams kanonizēt vārdu, pirms tiek izdarīta piekļuve datnei.

Pieņem, ka ir izstrādāta *Windows Forms* lietojumprogramma, kas ļauj apskatīt teksta datnes. 3.5. piemērā parādītā metode nodrošina šī vienkāršā uzdevuma izpildi, attēlojot datnes saturu laukā *outputBox* (*TextBox* komponents). Ja rodas kļūda, tiek izvadīts attiecīgs paziņojums par kļūdu.

3.5. piemērs

```
private void showFile(string fileName)
{
   try
   {
      System.IO.StreamReader sr = new System.IO.StreamReader(fileName);
      outputBox.Text = sr.ReadToEnd();
      sr.Close();
   }
catch (Exception ex)
   {
      outputBox.Text = "ERROR: " + ex.Message;
   }
}
```

Problēma var rasties brīdī, kad ļaunprātīgs lietotājs grib izmantot programmu datnes *C:\boot.ini* apskatīšanai, kas tam ļautu iegūt informāciju par datoru. Tādu iespēju var novērst, ievietojot metodē nosacījumu:

```
if (fileName == @"C:boot.ini")
{
  throw new Exception („Piekļuve liegta - tā ir sistēmas datne!"); }
```

Šis nosacījums nostrādās, ja tiks ievadīts *C:\boot.ini*. Tomēr to var apiet, ievadot, piemēram, vienu no dotajiem vārdiem:

- *C:\boot.ini*
- *..\..\..\..\..\..\..\..\..\..\..\..\boot.ini*
- *\boot.ini*

Problēmu var atrisināt divējādi:

1) izmantojot regulāro izteiksmi teksta *boot.ini* meklēšanai. Šis paņēmiens nostrādās, bet tas nav pārāk drošs, jo ļaunprātīgs lietotājs atradīs citu datnes uzdošanas veidu, piemēram, *c:\boot%2eini* (tīmekļa lietojumprogramma var interpretēt simbolus *%2e* kā punktu);

2) kanonizēt datnes vārdu un pēc tam veikt pārbaudi. Tas ir daudz drošāks veids, jo .NET *Framework* nodod lietojumprogrammai absolūto datnes vārdu, uz kā pamata arī var veikt pārbaudi.

Lai varētu pielietot šo paņēmienu, jāizmanto metode `System.IO.Path.GetFullPath`, kā parādīts 3.6. piemērā.


3.6. piemērs

```
try
{
   fileName = Path.GetFullPath(fileName);
   if (fileName == @"C:\boot.ini")
     {
         throw new Exception ("No, that's a system file!");
     }
  System.IO.StreamReader sr = new System.IO.StreamReader(fileName);
 outputBox.Text = sr.ReadToEnd();
 sr.Close();
}
catch (Exception ex)
   {
     outputBox.Text = "ERROR: " + ex.Message;
   }
```

 Paliek, kā minimums, viena iespēja atvērt datni *boot.ini*. Ja uzdot datni *\\localhost\c$\boot.ini* veidā, tad metode `Path.GetFullPath` nevarēs pārveidot šo tīkla ceļu lokālajā. Tomēr tāda iespēja pieejama tikai lietotājiem, kas var pieslēgties pie kopējā resursa *\\localhost\c$*, bet tādas tiesības dotas tikai administratoram.

Klase `System.IO.Path` vēl satur vairākas metodes, kuras var izmantot kanonizēšanas kļūdu novēršanai:

- `Combine` – kombinē divas ceļu saturošas rindas, ļaujot pašam izveidot ceļu līdz datnei, tādējādi minimizējot kanonizēšanas kļūdas rašanās iespēju;
- `GetDirectoryName` – atgriež absolūtā ceļa nosaukumu līdz mapei, kur atrodas datne;
- `GetExtension` – atgriež tikai datnes paplašinājumu;
- `GetFileName` – atgriež datnes vārdu un paplašinājumu (bez informācijas par mapi);
- `IsPathRooted` – atgriež vērtību *true*, ja ceļš līdz datnei ir absolūts, vai vērtību *false*, ja ceļš ir relatīvs.

Lai izvairītos no kanonizēšanas problēmām ASP.NET lietojumprogrammās, tiek ieteikts izmantot metodi `Request.MapPath` ievades datu, kas satur ceļu līdz datnei, kanonizēšanai. Šīs metodes izpratnei jāzina, ka tīmekļa pieprasījumi tiek sūtīti pa virtuālo ceļu, kas pēc tam tiek pārvērsti atbilstoši fiziskajam ceļam uz serveri. Piemēram, ja tīmekļa vietne pēc noklusējuma saņem virtuālā ceļa */index.html* HTTP pieprasījumu, tad tīmekļa serveris kanonizē šo ceļu, iegūstot fizisko ceļu *C:\Inetpub\Wwwroot\index.html*. Tomēr ne visus ceļus var pārveidot tik vienkārši.

IIS serveris ļauj ļoti elastīgi izveidot virtuālos ceļus. Jebkuram ceļam tādam kā */images/* var uzdot atbilstību pēc noklusējuma, piemēram, mapē *c:\Inetpub*. Tādējādi, lai veiktu virtuālā ceļa kanonizāciju, pārveidojot to fiziskajā, ir nepieciešams izanalizēt IIS virtuālo ceļu struktūru. Turklāt virtuālos ceļus var uzdot gan absolūtā, gan relatīvā formātā. Relatīvie virtuālie ceļi tādi kā *logo.gif* vai *../../images/logo.gif* norāda objekta atrašanās vietu attiecībā pret citu objektu. Absolūtajam ceļam jāsatur atbilstošās tīmekļa vietas saknes katalogu.

Vēl viens faktors, kas jāņem vērā virtuālo ceļu pārveidošanā fiziskajos ceļos, ir simbolu atbilstība. Pārlūkprogramma nepieņem tukšumzīmes (*blank character*) datņu vārdos un uzdod tās kā simbolu *%20* secību. Analoģiski – punkts tiek uzdots ar *%2e* palīdzību, slīpā svītra – ar *%2f* .

Metode `Request.MapPath` pārveido virtuālo ceļu fiziskajā ceļā. Tas ne tikai atbrīvo no nepieciešamības analizēt virtuālos ceļus, bet arī samazina kļūdu rašanās iespēju. Metodi izmanto šādā veidā: `string mappedPath = Request.MapPath(inputPath.Text)`. Lai pārliecinātos par to, ka ceļš patiešām noved pie objekta, kas atrodas lietojumprogrammas virtuālajā mapē, un tas ir atbilstošs fiziskajam ceļam līdz šim objektam, metodei `Request.MapPath` tiek nodoti trīs parametri: virtuālais ceļš, lietojumprogrammas saknes ceļš un vērtība *false* (sk. 3.7. piemēru).

3.7. piemērs

```
try
{
   string mappedPath = Request.MapPath(inputPath.Text,OE
   Request.ApplicationPath, false);
}
   catch (HttpException ex)
         {
          // Iespējama neatbilstība starp lietojumprogrammām
          }
```

Ja pieprasītais ceļš atrodas ārpus lietojumprogrammas mapes, nostrādā izņēmuma gadījums `HttpException`, kas praktiski izslēdz iespēju izmantot kanonizēšanas kļūdas, t. i., lietojumprogrammas datnes „neiziet” ārpus savas virtuālās mapes.

Kanonizācijas problēmas var radīt šādi iemesli:

- gari datņu vārdi – *MyLongFile.txt* var atbilst *MyLong~1.txt*;
- NTFS alternatīvās datu straumes – *MylongFile.txt::$DATA*;
- simboli „.” (*dot*) un „\” (*backslash*) – *MyLongFile.txt.*;
- \\?\ formāts - \\?\d:\temp\file.txt var atbilst *d:\temp\file.txt*;
- ceļi – *C:\boot.ini* var atbilst ..\..\..\..\..\..\..\..\..\..*boot.ini*;
- relatīvie datņu vārdi – ja netiek uzdota mape, kurā datnei jāatrodas;
- ierīču vārdi un rezervētie vārdi.

Ekspertu ieteikumi kanonizēšanas problēmu risku mazināšanai:

- izmantot regulārās izteiksmes datņu vārdu ierobežošanai.
  Piemēram, ja tiek izmantota regulārā izteiksme ^*[cd]:(?:\\\w+)+\\\w{1,32}\.(txt|gif)$,*
  tad *d:\dir1\dir2\gfxfile.gif* – derīgs datnes vārds;
  d:\dir1\dir2\datafile.txt – derīgs datnes vārds;
  c:*datafile.txt* – nederīgs (nav uzdota mape);
  f:\dir1\dir2\datafile.txt – nederīgs (nepareizs diska apzīmējums);
  d:*dir1\dir2\setup.exe* – nederīgs (nepareizs paplašinājums);
  c:*dir1\di~1\datafile.txt* – nederīgs (simbols ~);
  c:\dir1\dir2\datafile.txt. – nederīgs (punkts beigās);
  ?\d:\dir1\datafile.txt – nederīgs (nepareizs diska apzīmējums);
- 8 zīmju datņu vārdu (8.3. formāts) ģenerēšanas apturēšana.
  Reģistrā *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem* iestatīt vērtību: *NtfsDisable8dot3NameCreation : REG_DWORD : 1;*
- lietot pilno ceļu līdz datnei;
- veikt datņu vārdu kanonizēšanu (kā tas tika demonstrēts 3.7. piemērā).

3.3. Drošības izņēmumu izmantošana

Drošības izņēmumi (*Security Exceptions*) norāda uz to, ka .NET kodā ir atklātas drošības kļūdas un tiek ģenerēti paziņojumi par kļūdām. Protams, ir labāk saņemt paziņojumu par kļūdu nekā ciest no drošības viedokļa. Tomēr pati kļūdu paziņojumu izvade var radīt jaunas problēmas. Lai nodrošinātu aizsardzību pēc negaidīta izņēmuma situācijas rašanās, jāierobežo lietotāja tiesības (ja nepieciešams), jāieraksta ziņas par kļūdu drošā vietā un jāsniedz lietotājam pēc iespējas mazāk informācijas par kļūdu, lai tas šo informāciju nevarētu izmantot ļaunprātīgos nolūkos.

Neapstrādātās kļūdas nerada īpašas problēmas, jo .NET *Framework* neatklāj lietotājiem izsmeļošu informāciju par kļūdu avotu. Tomēr lietojumprogrammām ir jāpārtver vairākums izņēmuma situāciju un jāizvada attiecīgi paziņojumi par kļūdām, pēc kuriem var spriest par kļūdu cēloni.

Informācijas uzglabāšana notikumu žurnālā serverī – ideāls paņēmiens fiksēt paziņojumus par kļūdām, jo piekļuve žurnālam ir tikai sistēmas administratoram. Lai varētu pievienot notikumu žurnālam jaunu notikumu avotu, piemēram, savu programmu, nepieciešams reģistrā *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog\Application* ar komandas *New/Key* palīdzību pievienot programmas nosaukumu (piemēram, *MyApplication*).

Lai iegūtu piekļuvi *EventLog*, tiek izmantota vārdu telpa `System.Diagnostics`. Ir jāizveido metode, ar kuras palīdzību žurnālā tiks ierakstīti paziņojumi par kļūdām (metodes paraugs dots 3.8. piemērā).


3.8. piemērs

```
private void Report_Error(Exception ex, EventLogEntryType type, int eventID,  short category)
{
    EventLog myLog = new EventLog("Application");
    myLog.Source = "MyApplication";
    myLog.WriteEntry("An exception occurred: "+ ex.Message, type, eventID, category);
}
```

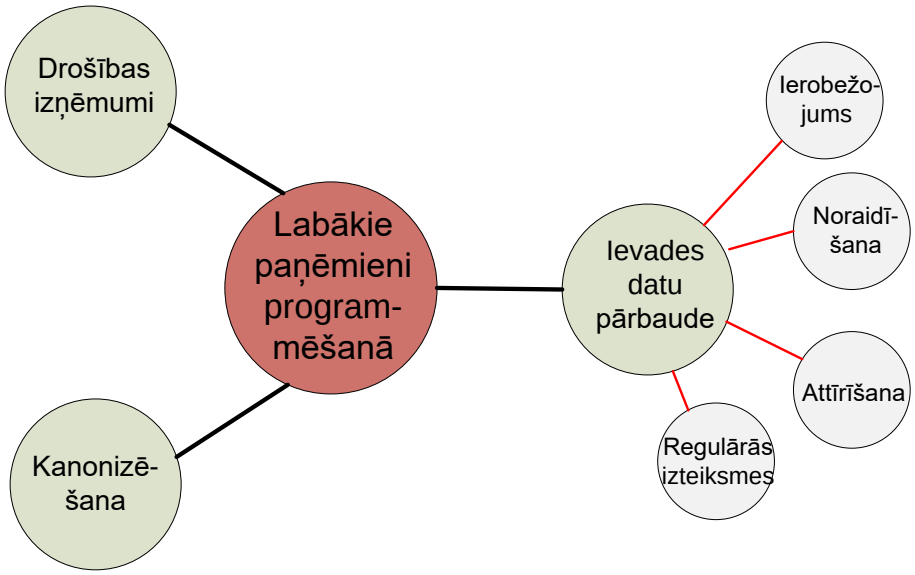
Šī metode jāizsauc katru reizi, kad lietojumprogrammā notiek situācija, kas varētu ieinteresēt administratoru. 3.9. piemērā uzrādītais koda fragments pārtver izņēmuma gadījumu un nodod to metodei `Report_Error`.


3.9. piemērs

```
try
{
   // Kods, kas var ģenerēt izņēmumu
}
catch (Exception ex)
{
    Report_Error(ex, EventLogEntryType.Error, 100, 200);
}
```

Pēc šādas metodoloģijas izmantošanas paziņojumi par kļūdām uzkrāsies notikumu žurnālā.

3.2. attēlā parādīts nodaļā minēto svarīgāko jēdzienu koks.



3.2. att. **Trešajā nodaļā minēto svarīgāko jēdzienu koks**

Pārbaudes testa jautājumi

1. Tīmekļa lietojumprogramma satur formu, kurā tiek ievadīts lietotāja vārds (tikai alfabēta simboli) un parole (var saturēt burtus un ciparus, bet jāsākas ar burtu). Kāda kontrole jāizmanto ievades datu pārbaudei?
    a. *RequiredFieldValidator*
    b. *RangeValidator*
    c. *CompareValidator*
    d. *RegularExpressionValidator*

2. Tīmekļa formā tiek ievadīti lietotāju kredītkaršu numuri. Kāds vadības elements ir vislabāk piemērots šo numuru pārbaudei?
    a. *RequiredFieldValidator*
    b. *RegularExpressionValidator*
    c. *CompareValidator*
    d. *RangeValidator*

3. Kāpēc no servera puses ir nepieciešams veikt tīmekļa formas ievades datu pārbaudi, ja klienta pusē tāda pārbaude tiek veikta?
    a. *Tas samazina servera pieprasījumu skaitu*
    b. *Lietotājs var apiet pārbaudi klienta pusē*
    c. *Tas ir intuitīvi pareizi*
    d. *Viss minētais*

4. Dota regulārā izteiksme ^[cd]:(?:\\\w+)+\\\w{1,32}\.(jpg¦mp3 | zip)$ datņu vārdu pārbaudei. Kādas datnes nav derīgas pēc šīs regulārās izteiksmes izmantošanas?
    a. *e:\courses\2840A\MySong.mp3*
    b. *c:\courses\2840A\Module3\MyPicture.jpg::$DATA*
    c. *c:\courses\2840A\MyProfile.zip*
    d. *c:\courses\2840A\Title.mp3*
    e. *d:\courses\2840A\..\2840A\MyPrivateFile.zip*

4. .NET Framework drošības iespējas

Microsoft Visual Studio ir vispusīgs daudzvalodu izstrādes rīks Microsoft Windows, tīmekļa un mobilo ierīču programmatūras ātrai izveidei un izvietošanai. Visual Studio ļauj izstrādātājiem risināt programmēšanas uzdevumus, izmantojot jau uzkrātās prasmes un zināšanas. Microsoft Visual Studio paceļ lietojumprogrammu izstrādi augstākā pakāpē. Iebūvētās projektu veidnes ļauj izveidot visdažādāko profesionālo programmatūru – gan Windows, tīmekļa un mobilajai lietošanai, gan XML tīmekļa pakalpojumiem, klašu bibliotēkām, Windows pakalpojumiem un kontroles bibliotēkām.

.NET Framework ir augsti produktīva, standartizēta daudzvalodu lietojumprogrammu izpildes vide, kura veic svarīgus tipveida darbus, pārvalda atmiņu, risina versiju problēmas, kā arī uzlabo lietojumprogrammas uzticamību, mērogojamību un drošību.

Nodaļā apskatītas .NET Framework iespējas, kas palīdz lietojumprogrammās iekļaut drošības nodrošināšanas mehānismus.

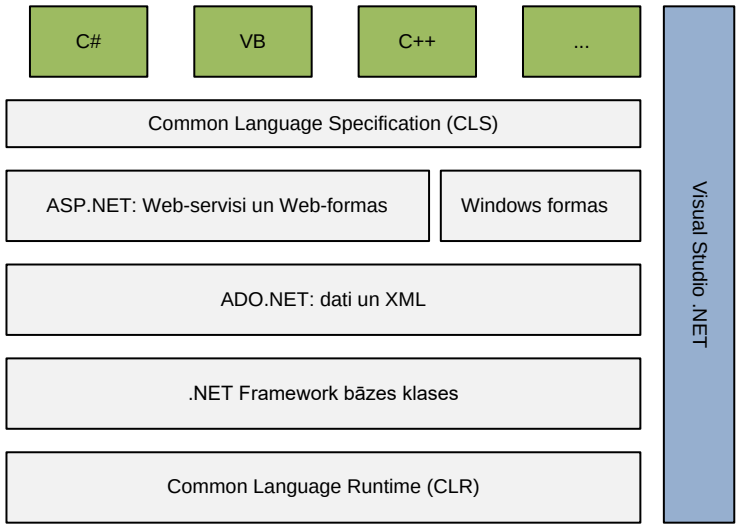
4.1. CLR drošības mehānismu izmantošana

Turpmākajam izklāstam nepieciešamas priekšzināšanas par .NET vidi [20],[21]. *Microsoft*.NET pamatā ir revolucionārā ideja par programmatūras un tīmekļa pielietojumu izstrādes tehnoloģiju pārnešanu uz vidi, kas nodrošina augsta līmeņa lietotņu izstrādi, apvienojot dažādu ierīču, dienestu un datoru darbību. Faktiski .NET sastāda četri pamata komponenti:

1) .NET *Building Block Services*- programmu līdzekļi piekļuvei pie dienestiem tādiem kā datņu glabātava (*file storage*), kalendārs (*calendar*), *Passport.Net* autentifikācijas dienests;
2) programmnodrošinājums jaunu ierīču darbībai Internet tīkla vidē;
3) .NET līdzekļi lietotāju darbam, kas ietver dabisku saskarni (*natural interface*), informācijas aģentus (*information agents*) un intelektuālos tagus (*smart tags*);
4) .NET infrastruktūra, kas sastāv no .NET *Framework, MS Visual Studio.NET,* .NET *Enterprise Servers*.

 Lielākā lietotāju daļa ar .NET saprot tieši infrastruktūru. Tāpēc turpmākajā izklāstā termins .NET tiks lietots infrastruktūras kontekstā.

.NET sastāvdaļu, ar kuras palīdzību tiek izstrādātas lietotnes, sauc par .NET *Framework*. .NET *Framework* un *Visual Studio* arhitektūras shēma parādīta 4.1. attēlā.



4.1. att. **.NET *Framework* un *Visual Studio* arhitektūras shēma**

.NET *Framework* nodrošina ātru savienotu lietojumprogrammu izveidi, kuras ļauj paplašināt lietotāju iespējas, piedāvājot veidošanas blokus tipisku programmēšanas uzdevumu veikšanai. Uz

.NET *Framework* modeļa pamata veidotās lietojumprogrammas darbojas efektīvi un sekmē sistēmu integrāciju neviendabīgās vidēs.

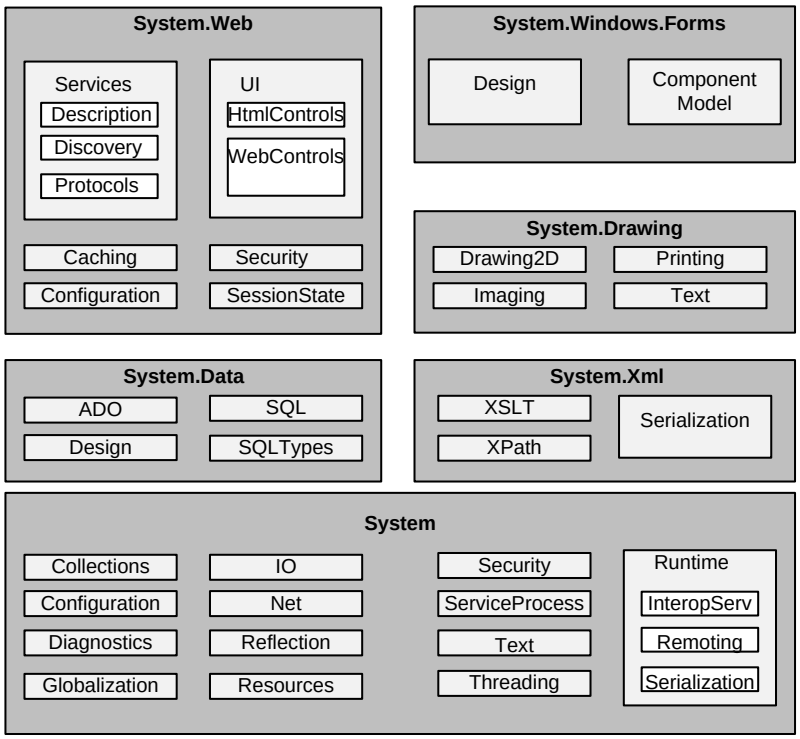
.NET *Framework* būtiskākās versijas parādītas 4.1. tabulā.

4.1. tabula

.NET *Framework* versiju uzskaitījums

Versija	Izdošanas datums	Visual Studio versija
1.0	2002-01-15	Visual Studio .NET (2002)
2.0	2005-10-27	Visual Studio 2005
3.0	2006-11-06	Visual Studio 2008
4.0	2010-04-12	Visual Studio 2010
4.5	2012-08-15	Visual Studio 2012
4.5.1	2013-10-17	Visual Studio 2013
4.5.2	2014-05-05	Visual Studio 2015
4.6	2015-07-20	Visual Studio 2015
4.6.1	2015-11-30	Visual Studio 2015 Update 1
4.6.2	2016-08-02	Visual Studio 2017
4.7	2017-04-05	Visual Studio 2017
4.7.1	2017-10-17	Visual Studio 2017
4.7.2	2018-04-30	Visual Studio 2017
4.8	2019-04-18	Visual Studio 2019
4.8.1	2022-08-09	Visual Studio 2022

.NET *Framework* sastāv no kopējā valodu izpildlaika CLR (*Common Language Runtime*) un .NET *Framework* bibliotēkām (sk. 4.2. att.). CLR pēc būtības ir virtuāla mašīna, kurā funkcionē .NET lietotnes. Visas .NET valodas izmanto .NET *Framework* klašu bibliotēkas, kas uztur praktiski visas tehnoloģijas, sākot ar ievadizvadi un apmaiņu ar datu bāzēm līdz pat *XML* un *SOAP*.



4.2. att. **.NET *Framework* bibliotēkas**

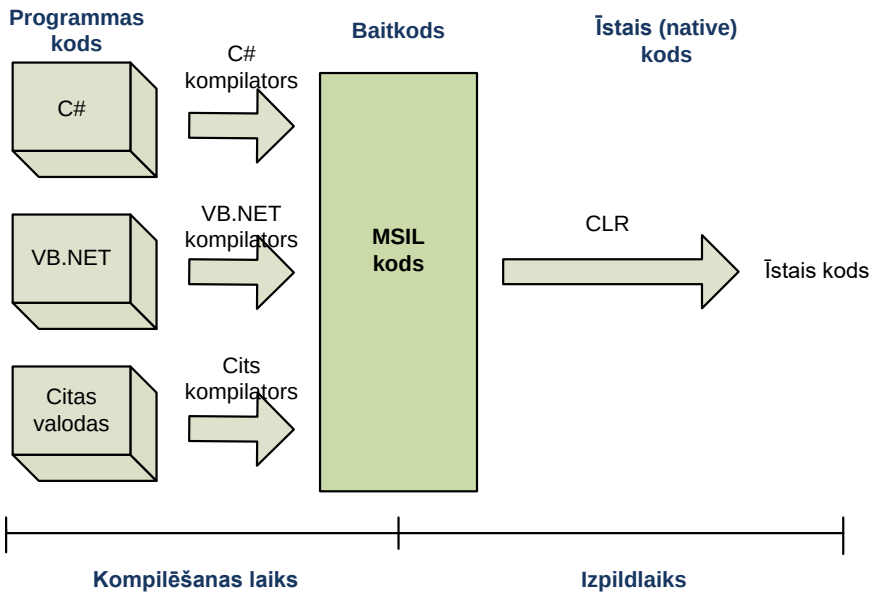
XML (*eXtensible Markup Language*) – programmēšanas valoda, kas speciāli izstrādāta darbam ar tīmekļa dokumentiem. Valoda XML nodrošina globālā tīmekļa izstrādātajiem iespēju radīt savas personiskās birkas (kodus), lai nodrošinātu tādu funkciju izpildi, ko nevar nodrošināt ar valodas HTML starpniecību. Atšķirībā no valodas HTML, kuras saites norāda tikai uz vienu dokumentu, valoda XML nodrošina saites, kurās ir atsauces uz vairākiem dokumentiem.

SOAP (*Simple Object Access Protocol*) – vienkāršais objektu piekļuves protokols. Protokols, ar kura palīdzību *Internet* tīklā un citās izkliedētās skaitļošanas vidēs var sazināties un kopīgi darboties dažādu platformu un operētājsistēmu programmatūra, sūtot cita citai XML formātā strukturētus datus. Ir specificēts, kā šāda apmaiņa var notikt ar HTTP palīdzību. SOAP ir saistīts ar tīmekļa pakalpēm, un tā izstrādē nozīmīga loma ir globālā tīmekļa konsorcijam.

Kopējais valodu izpildlaiks (CLR) ir programma .NET *Framework* vidē. Tā piedāvā pārvaldītu izpildes vidi, kuru aizsargā nozares standartiem atbilstošas tehnoloģijas un ir paredzēta, lai izstrādātājiem lietojumprogrammu izveidē ļautu izmantot dažādas programmēšanas valodas.

Tātad CLR ir koda izpildes laika vide, kurā tiek nodrošināta efektīva lietotņu savstarpēja mijiedarbība, kas iziet ārpus konkrētās programmēšanas valodas robežām (*cross-language interoperability*). Kādā veidā tiek panākta šī mijiedarbība? Ar CLS palīdzību. CLS ir kopējā valodu specifikācija (*Common Language Specification*), kas nosaka to, ka pilnīgai mijiedarbībai starp dažādiem objektiem ir nepieciešams piešķirt tiem tikai tās funkcionālās iespējas, kuras ir kopīgas dažādām programmēšanas valodām. CLS tādējādi nosaka minimālo funkciju kopu, kas jānodrošina programmēšanas valodai, lai tā varētu tikt izpildīta CLR vidē. *Microsoft* dokumentācijā var atrast pilnu CLS prasību specifikāciju. Kā piemēru var minēt prasības objektam „tips”. .NET *Framework* klašu bibliotēka satur tipus, kas atbilst vienkāršiem datu tipiem, kurus izmanto kompilatori. Ar CLS savietojami ir *Byte, Int16, Int32, Int64, Single, Double, Boolean, Char, Decimal, IntPtr* un *String*.

Lai programmas kods būtu kompilējams .NET vidē, programmēšanas valodai ir jābūt ar CLR savietojams kompilators. Šāds kompilators translē programmas kodu *Microsoft* izstrādātās starpvalodas MSIL (*Microsoft Intermediate Language*) kodā (sk. 4.3. att.).



4.3. att. **Vispārēja kompilēšanas procedūra**

Kompilēšanas rezultātā tiek iegūts MSIL baitkods (*bytecode*) un lietotnes pirmajā palaišanas reizē CLR vidē MSIL kods tiek kompilēts īstajā – noteiktā procesora mašīnkodā (*native code)*.

Baitkods – kompilējot programmas kodu, tas tiek pārveidots nevis procesora instrukcijās, bet gan no platformas neatkarīgā objektkodā jeb baitkodā, kas līdzīgs mašīnkodam un sastāv no instrukcijām, aiz kurām var sekot operandi (sintakse ļoti līdzīga *Asambler* valodai).

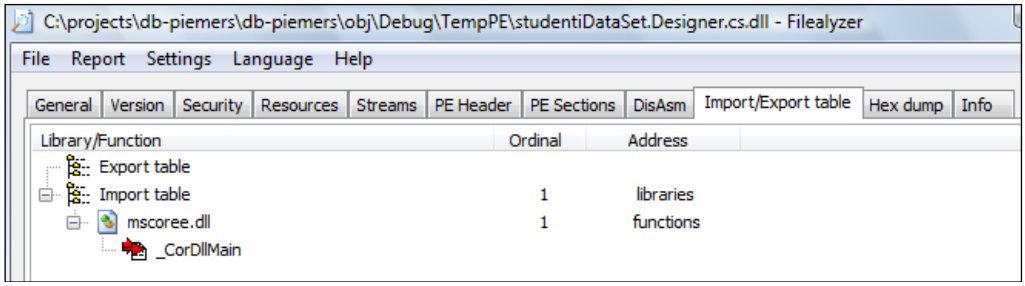
Īstais (dzimtais, izpildāmais) kods – kods, kas tiek iegūts kompilēšanas procesā izpildei noteiktā procesorā ar attiecīgo instrukciju kopu.

MSIL valodā iekļautas instrukcijas metožu ielādei, saglabāšanai, inicializācijai un izsaukšanai, kā arī instrukcijas aritmētiskajām un loģiskajām operācijām, plūsmu vadībai, tiešajai piekļuvei atmiņai u. c. Baitkods fiziski izvietots *.EXE* vai *.DLL* datņu veidā. Tādas datnes sauc par PE datnēm (*Portable Executable*), jo tās bez kompilēšanas var pārnest uz datoru ar citu arhitektūru.

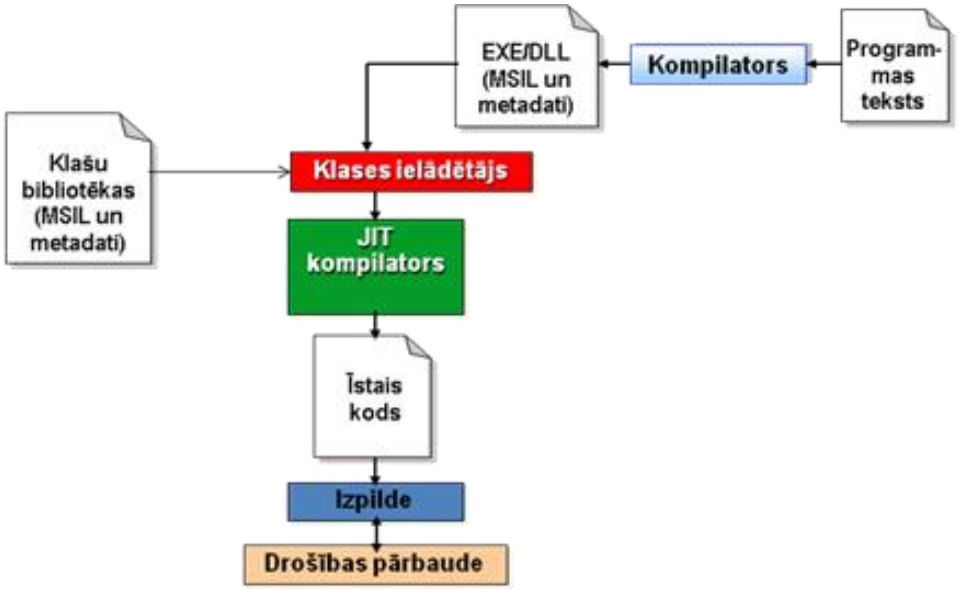
Kā jau tika minēts, MSIL kods nevar tikt izpildīts tiešā veidā (tas taču nav mašīnkods!), tāpēc CLR kompilē to ar operatīvā kompilatora JIT (*Just-In-Time*) palīdzību un iegūst noteiktam procesoram atbilstošu mašīnkodu – izpildāmo kodu (sk. 4.4. att.). Nokompilētais izpildāmais kods tiek saglabāts un pārkompilēts tikai sākotnējā koda izmaiņu gadījumā.

Vispārīgā gadījumā kompilēšanas process notiek vairāku soļu veidā.

1. Tiek uzrakstīta programma, piemēram, C# valodā.
2. Programma tiek kompilēta ar C# kompilatora palīdzību par *.EXE* datni.
3. Kompilators izveido MSIL kodu un ievieto to mapē *\TempPE* PE datnes veidā ar *.DLL* paplašinājumu. Jāņem vērā svarīga detaļa, ka, veidojot sākotnējo datni, kompilators no CLR importē funkciju *_CorExeMain.*



4. Kad lietotne sāk izpildīties, operētājsistēma ielādē šo PE un visas nepieciešamās *.DLL* (tai skaitā bibliotēku, kas importē funkciju *_CorExeMain – mscoree.dll*).
5. Operētājsistēmas ielādētājs veic pāreju uz PE ieejas punktu, ko nosaka kompilators. Tā kā operētājsistēma nevar izpildīt MSIL kodu, tad ieejas punktā notiek pāreja uz funkciju *jCorExeMain* no mscoree.dll.
6. Funkcija *jCorExeMain* sāk izpildīt MSIL kodu no PE.
7. CLR kompilē to ar JIT kompilatora palīdzību.
8. Kompilētā lietotne var tikt izpildīta.



4.4. att. **JIT kompilatora darbības shēma**

Kompilēšanas rezultātā visas iegūtās datnes tiek savāktas vienkopus – to sauc par asambleju (*assembly*). Asambleja ir kolekcija no vairākām datnēm. Pārsvarā šīs datnes satur programmas kodu, bet var saturēt arī citus resursus, kas asociējas ar doto programmu. Saka, ka asambleja sastāv no viena vai vairākiem moduļiem. Katra asambleja satur speciālu metadatu datni *.XML* formātā, ko sauc par deklarāciju jeb manifestu (*manifest*), ar kura palīdzību tiek aprakstīta asambleja. Manifests satur, kā minimums, šādu informāciju par asambleju:

- asamblejas vārds;
- versijas numurs;
- datnes un resursi dotajā asamblejā;
- asamblejas, ko izmanto dotā asambleja.

.NET uztur divu veidu asamblejas – privātās (*private*) un kopējās (*shared*). Privātās izmanto tikai viena lietotne un kontrole par versiju numuriem jāuzņemas programmas autoram. Kopējās asamblejas var izmantot dažādas lietotnes, tām raksturīgs unikāls „stingrs" (*strong name*) vārds, kas izmanto publiskās un privātās atslēgu pāri. Kopējo asambleju gadījumā kontroli par versiju numuriem veic CLR.

Tādējādi ir iespējama situācija, ka datorā var atrasties vienas un tās pašas asamblejas dažādas versijas un viens process tās vienlaicīgi izmanto. Tieši asambleju un versiju numuru izmantošana dod iespēju atrisināt DLL dažādu versiju savietojamības problēmu, zināmu kā *DLL Hell*. Tagad .NET lietotnes meklē asamblejas lokālajā katalogā, kas ļauj bez problēmām izmantot vienas un tās pašas asamblejas vairākas versijas.

 *DLL Hell* – strupceļa situācija, kas bija saistīta ar *Microsoft Windows* dinamisko DLL bibliotēku izmantošanu – versiju konflikts. *Microsoft* apgalvo, ka ar .NET ieviešanu šī problēma ir atrisināta un vairs nav aktuāla.

Tomēr izstrādātājiem jebkurā gadījumā jābūt uzmanīgiem, lai nepieļautu, piemēram, situāciju, ka divas dažādas DLL versijas pūlas ierakstīt datus vienā datnē, bet dažādos formātos.

CLR uztur divu veidu asambleju glabātavas – *Download Cache* un *Global Assembly Cache* (GAC). Asambleja, kuru izmanto vairākas lietotnes, tiek novietota GAC. Ja asambleja netiek atrasta lokālajā katalogā vai GAC, tad CLR pūlas nolasīt konfigurācijas datni. Šajā datnē var uzdot asamblejas atrašanās vietu (*code base*), tad CLR ielādēs asambleju un novietos to glabātavā *Download Cache*.

Ar CLR saistīta vēl viena svarīga .NET koncepcija – pārvaldīts kods (*managed code*). Tas ir kods, kuru vada CLR un kurš izpildās CLR vidē.

Pēc iepazīšanās ar .NET *Framework* darbības principiem var rasties jautājums – kurā brīdī sākas lietotnes drošības nodrošināšana?

Mūsdienu operētājsistēmu drošības modeļi ir saistīti ar lietotājiem un grupām. Tas nozīmē, ka lietotājiem un to izpildāmajām lietotnēm ir vai nav piešķirtas tiesības operācijām ar noteiktiem resursiem. .NET struktūra nodrošina izstrādātāju noteikto (*developer defined*) uzdotu drošības modeli, ko sauc par uz lomām balstītu drošību (*role-based security*). Turklāt .NET nodrošina koda piekļuves drošību (*code-access, evidence-based*). Šie līdzekļi ir koda drošas izpildes pamatā – ja lietotājam, piemēram, ir tiesības piekļuvei resursiem, bet lietotnes izpilde tam ir liegta, tad lietotāja pieprasījums piekļuvei resursam tiks noraidīts.

.NET drošības sistēma balstās uz CLR, kuras funkcijas pārsniedz operētājsistēmas iespējas, tādējādi papildinot tās. Faktiski CLR drošības iespējas ir vairāk detalizētākas, tāpēc pārvaldīta koda drošības nodrošināšana tiek uzticēta tieši CLR un drošības sistēmas darbs sākas tajā brīdī, kad CLR ielādē klasi (sk. 4.4. att.). Klases ielādētājs ir .NET drošības sistēmas sastāvdaļa, jo klases ielādes laikā jau tiek pārbaudītas pieejas tiesības resursiem un tiek garantēta noteiktu lomu un identifikācijas datu pārbaude.

 Saka, ka CLR aizsargā nevis kodu no lietotāja, bet lietotāju no koda, t. i., nevis nedod lietotājam izpildīt kodu, bet gan pārbauda, vai lietotājam ir atļauts izpildīt kodu.

Lietotņu tiesības uz noteikta veida resursiem tiek pārbaudītas procesa laikā, ko sauc par steka pārstaigāšanu (*stack walk*). Tā būtība ir šāda: metodes izsaukšanas brīdī stekā tiek ievietots ieraksts, kas satur atpakaļadresi, lokālos mainīgos un metodei nodoto parametru sarakstu. Metodes darbības beigās šis ieraksts tiek izņemts no steka – tādējādi lietotņu izpildes procesā steka izmērs palielinās vai samazinās. Pirms metodei atļaut piekļuvi, aizsargātais resurss var pieprasīt steka pārstaigāšanu, kas izsauc visu ierakstu pārbaudi ar nolūku noteikt to piekļuves tiesības aizsargātajam resursam. Faktiski notiek pārbaude: vai visām funkcijām, kas izsauc kodu, kurš savukārt vēršas pie resursa, ir tiesības to darīt. Šī procedūra garantē daudz augstāku drošības līmeni, salīdzinot ar izsaucošās lietotnes konkrēto tiesību pārbaudi.

Steka pārstaigāšana tiek izmantota, lai novērstu tā saucamos uzbrukumus ar aizvietošanu (*luring attack*), kad „mazāk ticams" kods izsauc „vairāk ticamu" kodu, lai veiktu „mazāk ticamam" kodam neatļautas darbības. Lai nodrošinātos pret šāda tipa apdraudējumiem, asambleju drošības nolūkā izmanto dažādas pārbaudes, piemēram, pieprasījumus par metodes darbības atļauju (sk. 4.2. tabulu). Šīs metožu atļaujas vairāk tiks apskatītas 6. nodaļā.

4.2. tabula

Asamblejās biežāk izmantotie metožu atļaujas pieprasījumi

Metode	Ietekme uz steka pārstaigāšanu
Pieprasījums (*Demand*)	Norāda, ka tiek veikta pilna steka pārbaude – visiem izsaukumiem jābūt identificētiem un ar noteiktām tiesībām. Pieprasījuma pārbaude *Demand* tiek izpildīta katrā izsaukuma reizē, jo izsaukumus var veikt dažādi kodi. *Demand* ir noturīga pret uzbrukumiem ar aizvietošanu – neautorizēts kods, kas pūlas iziet šo pārbaudi, tiek veiksmīgi pārķerts.
Apgalvojums (*Assert*)	Paaugstina izsaucošā koda privilēģijas. Apgalvojums palīdz nodrošināt metodes piekļuvi konkrētajam resursam arī gadījumos, ja objektam, kas izsauc metodi, nav nepieciešamo tiesību. Ja steka pārstaigāšanas laikā atrasts ieraksts ar šādu apgalvojumu, tad šīs atļaujas pārbaude dod pozitīvu rezultātu (tiek uzskatīts, ka programmētājs zina, ko dara).
Noraidījums (*Deny*)	Izsaucošajam kodam noraida visas tiesības.
Atļaujas (*PermitOnly*)	Izsaucošajam kodam noraida visas tiesības, izņemot uzrādītās atļaujas.
Saites pieprasījums (*LinkDemand*)	Saites pieprasījuma pārbaude ļauj programmētājam izmainīt steka pārstaigāšanas procedūru tā, lai notiktu tikai izsaucošā koda pārbaude, nevis visa steka pārbaude.
Mantošanas pieprasījums (*InheritanceDemand*)	Līdzīgi saites pieprasījuma pārbaudei, šajā gadījumā tiek kontrolēta klašu mantošana – piešķir tiesības mantotām klasēm.

Aizsardzība pret uzbrukumiem ar aizvietošanu ietver divus pretpasākumus (sk. 4.3. tabulu).

4.3. tabula

Pretpasākumi uzbrukumiem ar aizvietošanu

Pretpasākumi	Piešķirtā piekļuve resursiem:
Lietotāja identitātes priekšraksti (*user-identity statements*)	Lietotāja autentificēšanas informācijai
Koda identitātes priekšraksti (*code- identity statements*)	Informācijai par koda izcelsmi

Priekšraksts (*statement*) – teikums programmēšanas valodā, kas apraksta izpildāmās operācijas un ir sintaktiski un semantiski pilnīgi noformēts. Pēc kompilācijas vai interpretācijas priekšraksts tiek pārveidots vienā vai vairākās datora mašīnvalodas instrukcijās.

Pastāv divas pieejas steka pārstaigāšanas modificēšanai (sk. 4.4. tabulu).

4.4. tabula

Divas pieejas steka pārstaigāšanas pārveidošanai

Pieeja	Veids
Deklaratīvā (atribūti)	• Atribūti definē atļauju nosacījumus pirms izpildes • Nodrošina vispusīgus drošības iestatījumus
Imperatīvā (kods)	• Nodrošina daļējus drošības iestatījumus • Atļaujas var attiekties uz izpildes vidi • Tiek īstenoti izpildlaikā

Deklaratīvie priekšraksti nosaka .NET *Framework* videi veikt drošības pasākumu pārbaudi pirms metodes izpildes. Tas ir visdrošākais veids ierobežot piekļuvi kodam, jo izpildes vide veic drošības pārbaudi pirms koda izpildes.

Imperatīvie priekšraksti tiek noteikti kodā un var tikt izmantoti detalizētākam piekļuves ierobežojumam koda sastāvdaļām, t. i., ierobežo piekļuvi atsevišķiem metodes komponentiem.

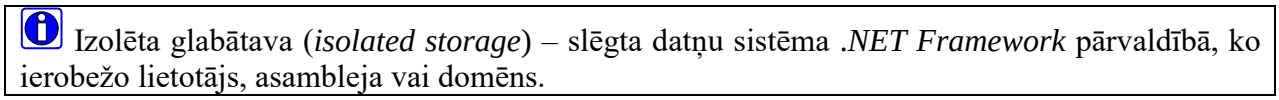
Vispārīgā gadījumā .NET drošība sastāv no daudziem komponentiem un svarīgākie no tiem ir:

- drošība lietotņu līmenī (*application domain*) – tiks apskatīta 4.2. apakšnodaļā;
- drošība lomu līmenī (*role-based security*) – tiks apskatīta 5. nodaļā;
- drošība koda līmenī (*code access security*) – tiks apskatīta 6. nodaļā;
- drošība apliecinājumu līmenī (*evidence-based security*) – tiks apskatīta 6. nodaļā;
- kriptogrāfija – tiks apskatīta 7. nodaļā.

4.2. Aizsardzības ieviešana ar lietotņu domēnu izmantošanu

Tradicionāli lietotnes glabā patstāvīgos datus cietajā diskā un tieši mijiedarbojas ar datņu sistēmu. Tomēr koda piekļuves aizsardzības mehānismi, kas iebūvēti .NET *Framework*, neļauj asamblejai pa tiešo ierakstīt datus datņu sistēmā, ja lietotājam nav uz to tiesību. Lietotājam var nebūt tiesību piekļuvei tai datņu sistēmas daļai, kurā tas pūlas ierakstīt datus. Ja, piemēram, dati glabājas mapē *System*, bet lietotājs nav administrators, tad viņam, visticamāk, nebūs piekļuves tiesību ierakstīt šeit datus. Ja arī lietotājam nav dotas tiesības datņu ierakstīšanai diskā, izstrādātājam jārūpējas arī par to, ka dati var tikt izmantoti citos nolūkos. Piemēram, pirmo pārlūkprogrammu izstrādātāji neaizsargāja kešatmiņu un vietņu apmeklētības vēsturi. Varētu likties, ka šī informācija nav jāaizsargā, jo ļaunprātīgs lietotājs nevar to izmantot paaugstinātu privilēģiju iegūšanai. Tomēr no tās daudzos gadījumos var noteikt veiktās darbības, kas pārkāpj lietotāja konfidencialitāti.

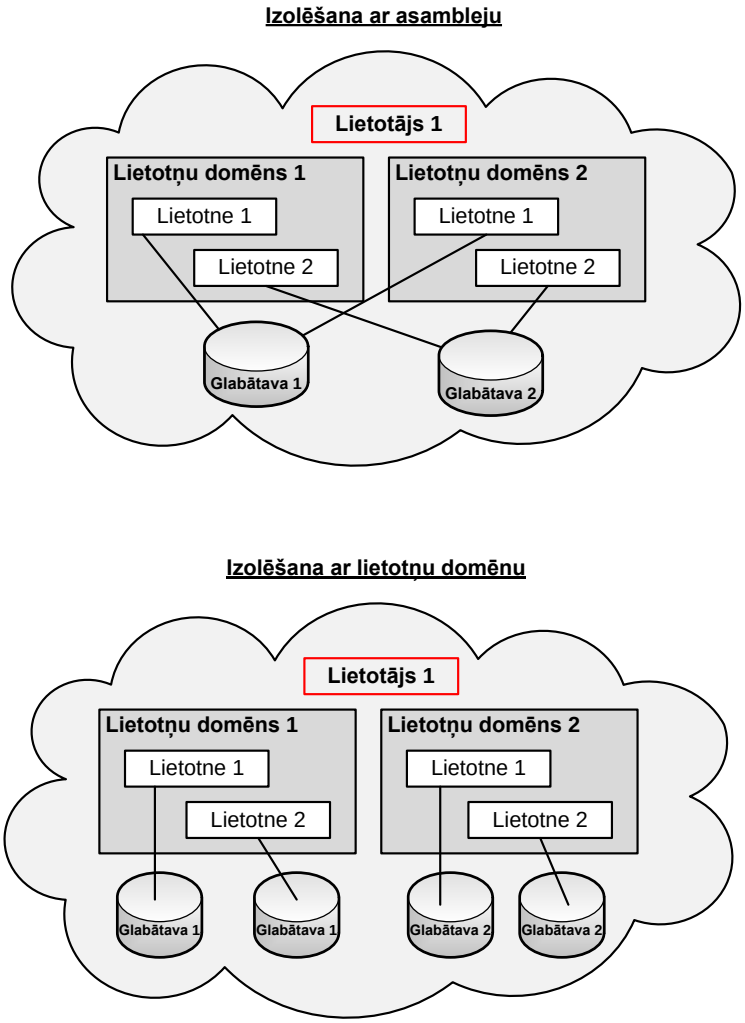
Datņu konfidencialitātes aizsardzības nodrošināšanai var izmantot izolētas glabātavas, kurām nav nepieciešama tieša mijiedarbība ar datņu sistēmu. .NET *Framework* nodrošina asamblejām piekļuvi glabātavai, kas ir specificēta konkrētai asamblejai vai lietotājam, un automātiski šifrē šo glabātavu, lai aizsargātu to no nesankcionētas piekļuves.


Izolēta glabātava (*isolated storage*) – slēgta datņu sistēma *.NET Framework* pārvaldībā, ko ierobežo lietotājs, asambleja vai domēns.


Saka, ka CLR lietotņu atdalīšanas (izolēšanas) nolūkā izveido lietotņu domēnus (nejaukt ar citu domēna apzīmējumu - sistēmu, ko izmanto, lai identificētu konkrētu datoru *Internet* tīklā).

Izolētās glabātavas ne vienmēr ir labākais risinājums patstāvīgu datu glabāšanai. Tās nevajadzētu izmantot konfigurācijas parametru uzglabāšanai, ko pārrauga administrators. Taču tas ir labs veids dažādu lietotāja iestatījumu uzglabāšanai, jo tie nav konfigurācijas parametri un tos nepārrauga administrators.

Piekļuve izolētai glabātavai tiek ierobežota ar lietotāju, kurš izveidojis šo glabātavu. Papildu tam piekļuvi izolētai glabātavai ierobežo ar noteiktu asambleju. Citiem vārdiem sakot, asambleja B nevar iegūt piekļuvi datnēm, kas atrodas asamblejas A izveidotajā izolētajā glabātavā. Vēl viens papildu ierobežojums ir ar lietotņu domēna palīdzību. Ja glabātava ierobežota ar lietotņu domēnu, tad vienas asamblejas kopijas, kas palaistas dažādos lietotņu domēnos, nevarēs kopīgi izmantot vienu glabātavu. Šie visi trīs glabātavu ierobežojumu tipi attēloti 4.5. attēlā.



4.5. att. **Izolētas glabātavas ierobežotas ar lietotāju un asambleju vai domēnu**

Kad lietotne izmanto izolētu glabātavu, tā saglabā datus savā unikālajā glabātavē. Fiziski glabātava ir datne datora datņu sistēmā. Parasti glabātava izvietojas klienta datņu sistēmā, bet serveru lietotnes var izveidot izolētas glabātavas serverī.

Dažādās *Windows* versijās izolētas glabātavas atrodas dažādās vietās. 4.5. tabulā uzrādītas datņu sistēmas atrašanās vietas, kurās tiek veidotas izolētas glabātavas pārvietojamu un nepārvietojamu lietotāja profilu gadījumos. Piemēram, konkrētajos datoros izolētas glabātavas tika izveidotas šādās mapēs: *C:\Documents and Settings\Peter\Local Settings\ Application Data\ IsolatedStorage\ kdoaqme0.pqt\nwcqbld5.450\Url.qas5r1wwrccrqucwmpb3sq1dohb0frc5\ Url.qas5r1wwrccrqucwmpb3sq1dohb0frc5\ Files\ mansfails.txt* un *C:\Users\ Peter\ AppData\ Local\ IsolatedStorage\ cjbujpvg.ezu\ j213jpbs.me5\ Url.wdhij2oi4zznasthgusixn5zq1ue4fya\ AssemFiles\ mansfails.txt*.

4.5. tabula

Izolēto glabātavu atrašanās vietas dažādās *Windows* versijās

Operētājsistēma	Datņu sistēmas atrašanās vieta
Windows 98, *Windows Me* – lietotāju profili aizliegti	Pārvietojamā glabātava = <SYSTEMROOT>\Application Data Nepārvietojamā glabātava = WINDOWS\Local Settings\Application Data
Windows 98, *Windows Me* – lietotāju profili atļauti	Pārvietojamā glabātava = <SYSTEMROOT>\Profiles\<user>\Application Data Nepārvietojamā glabātava = Windows\Local Settings\Application Data
Windows NT 4.0	<SYSTEMROOT>\Profiles\<user>\Application Data
Windows NT 4.0 – Service Pack 4	Pārvietojamā glabātava = <SYSTEMROOT>\Profiles\<user>\Application Data Nepārvietojamā glabātava = <SYSTEMROOT>\Profiles\<user>\Local Settings\Application Data

Windows 2000, *Windows* XP, *Windows Server* 2003 – jauninašana no NT 4.0	Pārvietojamā glabātava = <SYSTEMROOT>\Profiles\<user>\Application Data Nepārvietojamā glabātava = <SYSTEMROOT>\Profiles\<user>\Local Settings\Application Data
Windows 2000 – tīra instalēšana (un jauninašana no *Windows* 98 un NT 3.51)	Pārvietojamā glabātava = <SYSTEMDRIVE>\Documents and Settings\<user>\Application Data Nepārvietojamā glabātava = <SYSTEMDRIVE>\Documents and Settings\<user>\Local Settings\Application Data
Windows XP, *Windows* Server 2003 - tīra instalēšana (un jauninašana no *Windows* 2000 un *Windows* 98)	Pārvietojamā glabātava = <SYSTEMDRIVE>\Documents and Settings\<user>\Application Data Nepārvietojamā glabātava = <SYSTEMDRIVE>\Documents and Settings\<user>\Local Settings\Application Data
Windows Vista un 7	Pārvietojamā glabātava = <SYSTEMDRIVE>\Users\<user>\AppData\Roaming Nepārvietojamā glabātava = <SYSTEMDRIVE>\Users\<user>\AppData\Local
Windows 10, 11	Pārvietojamā glabātava = <SYSTEMDRIVE>\Users\<user>\AppData\Roaming Nepārvietojamā glabātava = <SYSTEMDRIVE>\Users\<user>\AppData\Local

 Nav vajadzības atcerēties visas atrašanās vietas konkrētā operētājsistēmā, pietiek zināt to, ka izolētas glabātavas tiek izvietotas lietotāja profilā.

Darbam ar izolētām glabātavām izmanto vārdu telpu `System.IO.IsolatedStorage`, kurā ir trīs klases:

1) `IsolatedStorageException` – nodrošina izņēmuma situāciju apstrādi;
2) `IsolatedStorageFile` – nodrošina izolētas glabātavas datņu vadību;
3) `IsolatedStorageFileStream` – nodrošina izolētas glabātavas datņu nolasīšanu un ierakstīšanu. Darbs ar šādām datnēm ir analoģisks darbam ar parastajām datņu sistēmas datnēm.

Atšķirība starp `IsolatedStorageFile` un `IsolatedStorageFileStream` ir tajā apstāklī, ka klase `IsolatedStorageFile` tiek izmantota piekļuvei atsevišķām glabātavām, bet klase `IsolatedStorageFileStream` dod iespēju darboties ar atsevišķām glabātavas datnēm.

Darbs ar izolētām glabātavām ir ļoti līdzīgs darbam ar parastām datnēm. Galvenās atšķirības ir tādas, ka izstrādātājam nepieciešams:

- izmantot vārdu telpas `System.IO` un `System.IO.IsolatedStorage`;
- uzdot objektu `IsolatedStorageFile`;
- konstruēt datņu sistēmas objektus `StreamWriter` vai `StreamReader`.

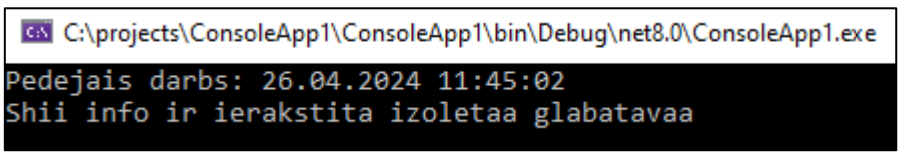
4.1. piemērā izolētā glabātavā tiek izveidota datne *mansfails.txt*, kurā ar objekta `StreamWriter` palīdzību tiek ierakstīts teksts un pēc tam ar *StreamReader* palīdzību nolasīts.

 4.1. piemērs. Darbs ar izolētu glabātavu

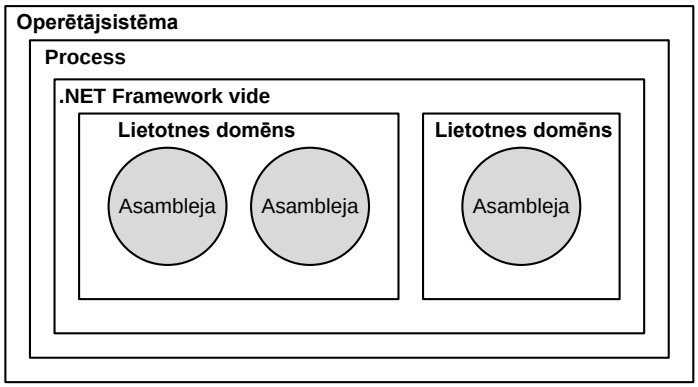
```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.IO.IsolatedStorage;
namespace iso
{
   class Program
   {
      static void Main(string[] args)
      {
         Program.WriteToIso();
         Program.ReadFromIso();
```

```
    }
    public static void WriteToIso()
    {  // Datu ierakstīšana izolētā glabātavā
       // Apraksta izolētu glabātavu
       IsolatedStorageFile isoStore = IsolatedStorageFile.GetUserStoreForAssembly();
       // Veido izolētu glabātavu
       IsolatedStorageFileStream configFile = new IsolatedStorageFileStream("mansfails.txt",
FileMode.Create, isoStore);
       // Veido objektu StreamWriter izolētai glabātavai
       StreamWriter writer = new StreamWriter(configFile);
       // Ieraksta kaut ko
       String output;
       System.DateTime currentTime = System.DateTime.Now;
       output = "Pedejais darbs: " + currentTime.ToString();
       writer.WriteLine(output);
       output = "Shii info ir ierakstita izoletaa glabatavaa";
       writer.WriteLine(output);
       writer.Flush();
       writer.Close();
       configFile.Close();
    }

    public static void ReadFromIso()
    {  // Datu nolasīšana no izolētas glabātavas
       // Apraksta glabātavu, izolētu ar asambleju
       IsolatedStorageFile isoStore = IsolatedStorageFile.GetUserStoreForAssembly();
       // Atver izolētu glabātavu tikko izveidotajā asamblejā
       IsolatedStorageFileStream configFile = new IsolatedStorageFileStream("mansfails.txt",
FileMode.Open, isoStore);
       // Veido objektu StreamReader izolētai glabātavai
       StreamReader reader = new StreamReader(configFile);
       // Nolasa tekstu
       string theEntry;
       do
       {  theEntry = reader.ReadLine();
          Console.WriteLine(theEntry);
       } while (theEntry != null);
       Console.ReadLine();
       reader.Close();
       configFile.Close();
    }
  }
}
```

Rezultāts



Lietotņu domēns faktiski ir „konteiners", kas ļauj izpildīt vairākas asamblejas vienā procesā, taču aizliedz tām piekļuvi citu asambleju atmiņai. Lietotņu domēniem ir daudzas procesu iespējas tādas kā atsevišķi atmiņas apgabali un piekļuve resursiem. Tomēr lietotņu domēni ir daudz efektīvāki nekā procesi, jo ļauj izpildīt asamblejas atsevišķos lietotņu domēnos, netērējot resursus atsevišķu procesu izpildei. 4.6. attēlā parādīts, kādā veidā viens process var saturēt vairākus lietotņu domēnus.

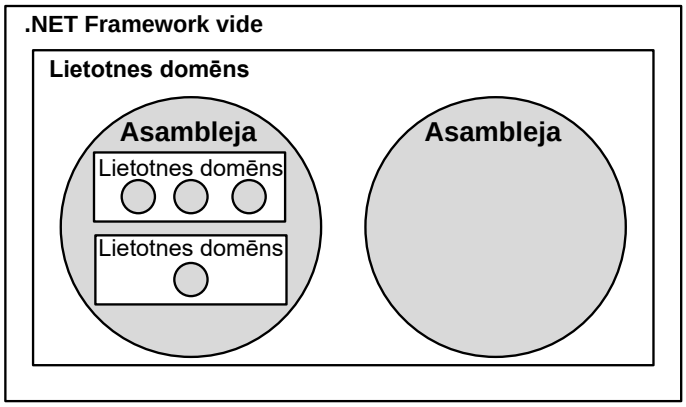


4.6. att. **Lietotņu domēni nodala asamblejas viena procesa ietvaros**

.NET *Framework* pārvalda lietotņu domēnus, bet operētājsistēma pārvalda procesus.

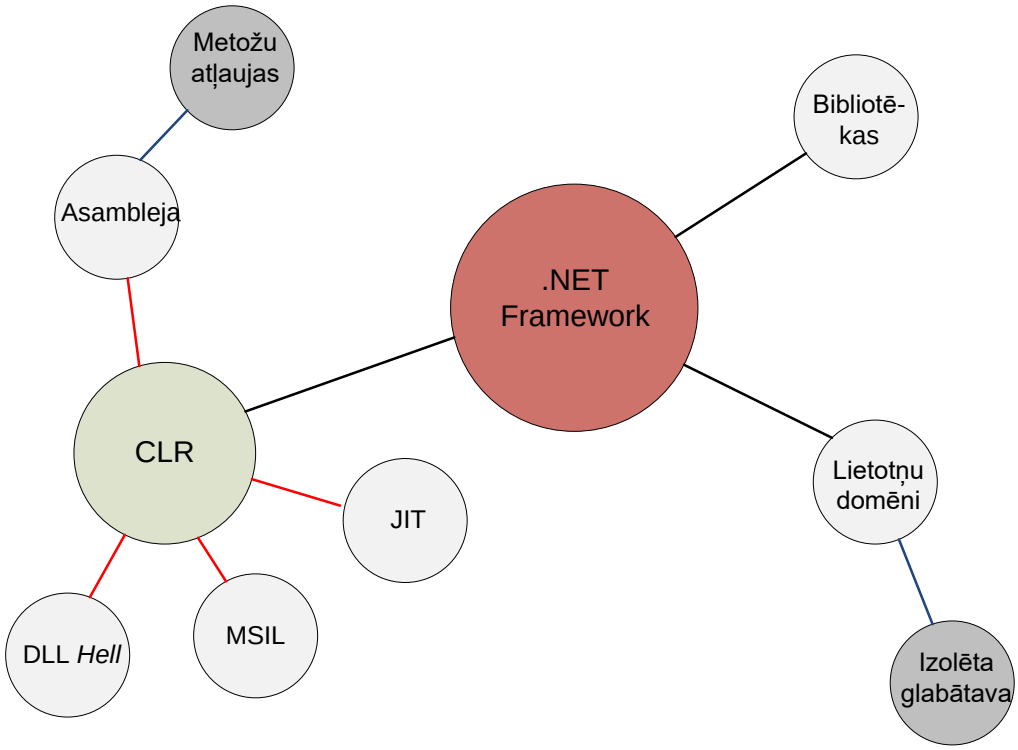
Labs lietotņu domēna piemērs ir ASP.NET darba procesa realizācija IIS dienestos. Ja 10 lietotāji vienlaicīgi apmeklē ASP.NET vietni, tad ASP.NET izveido atsevišķu lietotņu domēnu katram lietotājam. Faktiski ASP.NET izpilda 10 atsevišķus asamblejas eksemplārus. To pašu efektu var panākt, izpildot vienu asambleju 10 atsevišķos procesos. Taču šajā gadījumā tiks patērēts procesora laiks, pārslēdzoties starp procesiem, kas samazina veiktspēju. *Internet Explorer* arī veido vienu lietotņu domēnu visām asamblejām no vienas vietnes.

Izstrādātājs asambleju izpildei var veidot savus lietotņu domēnus, minimizējot risku, ka asamblejas vērsīsies pie resursiem vai veiks nesankcionētas darbības. Piemēram, var izmantot lietotņu domēnus, lai izpildīt citu asambleju ar ierobežotām tiesībām. 4.7. attēlā parādīts, kā asambleja var „izmitināt" (*host*) lietotņu domēnus.



4.7. att**. Lietotņu domēnu „izmitināšana" asamblejā**

Lietotņu domēna izmantošana koda piekļuves aizsardzības nodrošināšanai tiks apskatīta 6. nodaļā. 4.8. attēlā parādīts nodaļā minēto svarīgāko jēdzienu koks.



4.8. att. **Ceturtajā nodaļā minēto svarīgāko jēdzienu koks**

Pārbaudes testa jautājumi

1. Kurš no atļauju pieprasījumiem var tikt izmantots *Assert* metodes *Permission* objektam, bet nedrīkst būt par cēloni uzbrukumam ar aizvietošanu?
   a. Piekļuves datnei pieprasījums
   b. Pieprasījums ierakstīšanas atļaujai sistēmas reģistrā
   c. Pieprasījums atļaujai izsaukt *Win32 API* pašreizējā sistēmas datuma un laika pārbaudei
   d. Pieprasījums atļaujai izpildīt attālinātu procedūru

2. Kura no CLR drošības iespējām aizsargās Jūsu lietotni no citu izstrādātāju *.dll* datnes izmantošanas?
   a. Drošība koda līmenī
   b. Drošība lomu līmenī
   c. Lietotņu domēns
   d. Autentificēšana

3. Tika izveidota asambleja, kas ļauj izpildīt darbības ar datnēm. Kura no drošības iespējām pasargās pret uzbrukumu ar aizvietošanu?
   a. *Windows* autentificēšana
   b. Drošība koda līmenī
   c. Lietotņu domēns
   d. *Microsoft Passport*

4. Tika izveidota droša klase ar augstu atļauju pakāpes līmeni. Kura no atļaujām liegs citai klasei mantot šīs drošības iespējas?
   a. *LinkDemand*
   b. *InheritanceDemand*
   c. *AccessDemand*
   d. *InstanceDemand*

5. Lomās balstītas aizsardzības ieviešana

Uz lomām balstīta aizsardzība – tas ir lietotāju autentificēšanas un pilnvarošanas process uz noteiktu atļauju pamata. Izstrādātājiem ir jābūt iespējai izmantot lomās balstītu aizsardzību, lai ierobežotu piekļuvi dažādām lietotņu daļām.

Piektajā nodaļā apskatīta lomās balstītas aizsardzības realizācija ar .NET Framework. Tiks paskaidrota autentificēšanas saistība ar pilnvarošanu un lietotāja konta pārbaudes veidi, kā arī lomās balstītas drošības ieviešana ar deklaratīvajiem un imperatīvajiem priekšrakstiem.

5.1. Lomās balstītas drošības pamati

Lomās balstīta aizsardzība (*Role-Based Security - RBS*) – lietotāju autentificēšanas un pilnvarošanas process, kas nosaka, kādus resursus lietotājs ir pilnvarots izmantot [22], [23].

Loma (*role*) – definē darba funkcijas, kas tiek noteiktas pilnvarošanas procesā.

Datorsistēmu funkcionēšanas drošības nodrošināšanā svarīgs ir lomu jēdziens. Piekļuves vadība uz lomu pamata – tā ir tāda drošības politika, kurā sistēmas subjektu piekļuves tiesības pie noteiktiem objektiem tiek grupētas saskaņā ar to pielietošanas specifiku, tādējādi veidojot lomas. Privilēģijas netiek piešķirtas lietotājam tiešā veidā, tās tiek iegūtas caur savām lomām, t. i., lietotājam tiek nozīmētas lomas.

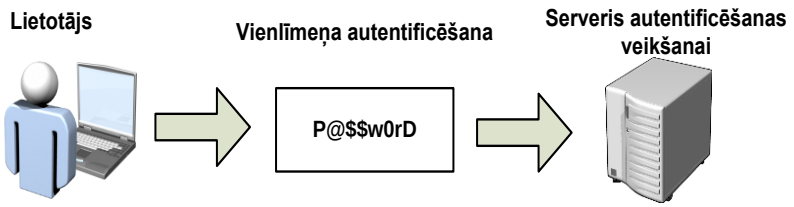
Lomas tiek veidotas dažādu darba funkciju veikšanai. Noteiktām lomām tiek piešķirtas atļaujas veikt tās vai citas operācijas. Piemēram, sistēmas lietotājam nozīmētas fiksētas lomas, saskaņā ar kurām tas iegūst attiecīgas privilēģijas noteiktu sistēmas funkciju veikšanai.

– Vienam subjektam var būt vairākas lomas.
– Vairākiem subjektiem var būt viena loma.
– Vienai lomai var būt vairākas atļaujas.
– Vairākām lomām var būt viena atļauja.

Autentificēšana (*authentication*) – ziņojuma, datu avota un datu saņēmēja autentiskuma apstiprināšana, kā arī lietotāja identitātes pārbaudes procedūra.

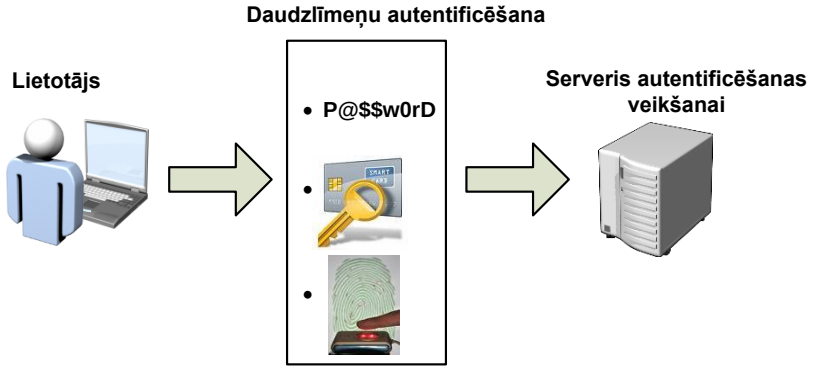
Pilnvarošana *(authorization)* – pilnvaru piešķiršana kādai personai vai personu grupai noteiktu darbību izpildei un resursu izmantošanai.

Autentificēšana ir lietotāja personības noteikšanas process, pati redzamākā un fundamentālākā drošības koncepcija. Tradicionāli lietotāja personības apstiprināšanai tiek lietotas paroles. Tiek uzskatīts, ka tikai lietotājs zina paroli un tā apstiprina viņa personību. Citi personas autentificēšanas paņēmieni pamatojas uz to, ka lietotājam ir kāds personību apstiprinošs priekšmets (viedkarte) vai raksturiezīme (biometrija). Vienlīmeņa autentificēšanas piemērs parādīts 5.1. attēlā.



5.1. att. **Autentificēšana ar paroles palīdzību**

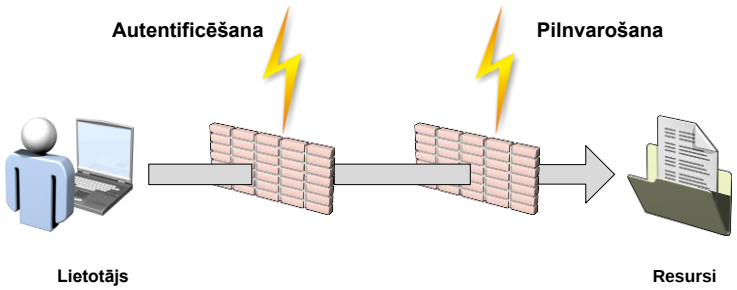
Vairāklīmeņu autentificēšana (sk. 5.2. att.) ir divu un vairāku autentificēšanas metožu kombinācija drošības palielināšanai. Visbiežāk izplatītā kombinācija ir parole un viedkarte.



5.2. att. **Daudzlīmeņu autentificēšana ar paroles, viedkartes un/vai pirksta nospieduma palīdzību**

Kā parasti, paaugstinātas drošības pasākumiem ir savi trūkumi – lietotājiem nepatīk veikt daudzlīmeņu autentificēšanu, jo tas prasa vairāk laika, bet biometrijas gadījumā vēl papildu aprīkojumu. Tomēr daudzlīmeņu autentificēšana ir viens no labākajiem pasākumiem drošības risku minimizēšanai.

Pēc autentificēšanas parasti tiek veikta pilnvarošana (sk. 5.3. att.). Pilnvarošanas procesā notiek pārbaude, vai lietotājam ir tiesības piekļūt pie noteiktiem resursiem. Autentificēšana ir cieši saistīta ar pilnvarošanu, un bieži šos terminus jauc. Atšķirība tiek demonstrēta ar iekāpšanas lidmašīnā piemēru. Pirms iekāpšanas lidmašīnā tiek pieprasīta pase un biļete. Pases pārbaude ir autentificēšana, jo ļauj noskaidrot cilvēka personību. Tāpat tiek pārbaudīta biļete, lai noteiktu, uz kādu reisu grib tikt šis cilvēks. Tādējādi notiek pilnvarošana – pārbaude, vai cilvēkam ir atļauja iesēsties lidmašīnā.



5.3. att. **Autentificēšanas un pilnvarošanas procesu secība**

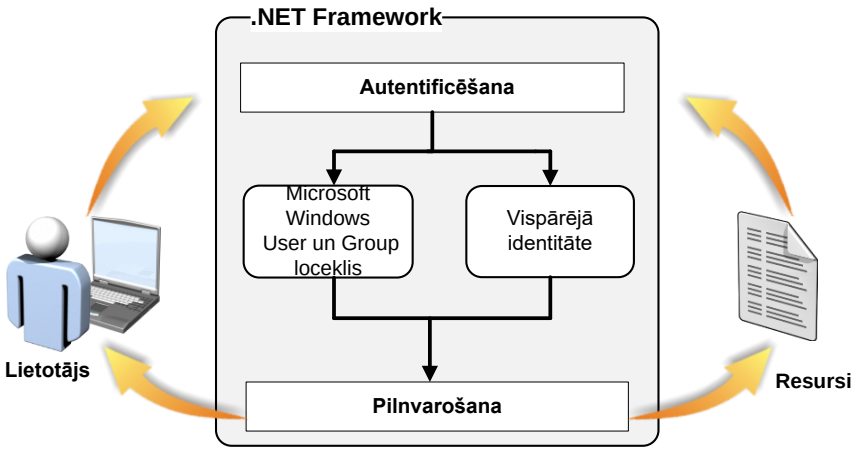
Datorsistēmās autentificēšana parasti tiek veikta, ievadot lietotāja vārdu un paroli. Vārds identificē lietotāju, bet parole apstiprina to, ka lietotājs patiešām ir tas, par ko uzdodas. Tomēr datorsistēma vēl nezina, vai lietotājam ir piekļuves tiesības pie noteiktiem resursiem, tāpēc pilnvarošanas nolūkā tiek pārbaudīts ACL saraksts.

Uz lomām balstīta drošība .NET *Framework* vidē ļauj izstrādātājiem izmantot informāciju par *Microsoft Windows* lietotājiem un grupām, kā arī izveidot savas autentificēšanas un pilnvarošanas procedūras, veicot lietotāja identificēšanas pārbaudi (sk. 5.4. att.).

.NET *Framework* vidē tiek izmantotas divas koncepcijas:

- identitāte (*identity*) – iekapsulē lietotāja vārdu;
- pilnvaras (*principal*) – iekapsulē informāciju par lietotāja lomām.

CLR nodrošina funkcionalitāti, tiešā veidā pārbaudot lietotāja identitāti un pilnvaras, vai arī veicot deklaratīvās vai imperatīvās (kodā) atļauju pārbaudes.



5.4. att. **Autentificēšana un pilnvarošana *.NET Framework* vidē**

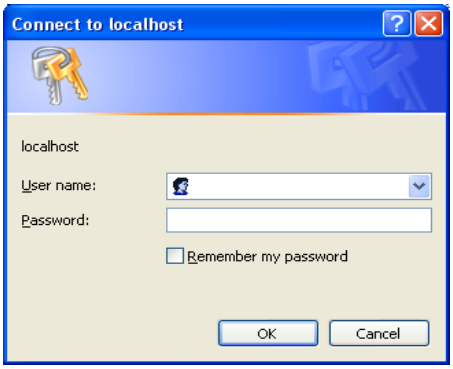
Iekapsulēšana – datu struktūru un atsevišķu funkciju apvienošana vienā objektā, lai novērstu nevēlamus blakusefektus sistēmas darbībā. Izmantojot iekapsulēšanu, lielas objektorientētas programmas kļūst labāk lasāmas, jo visi dati un ar tiem saistītie kodi ir atrodami vienā vietā.

Pastāv vairākas populāras autentificēšanas metodes:

- *Basic*;
- *Digest*;
- ciparparaksti;
- integrētās:
  - *Kerberos*
  - NTLM
- *Microsoft Passport*;
- biometrija.

Īsumā tiks apskatītas dažas metodes – *Basic*, *Digest* un *Kerberos*.

Basic

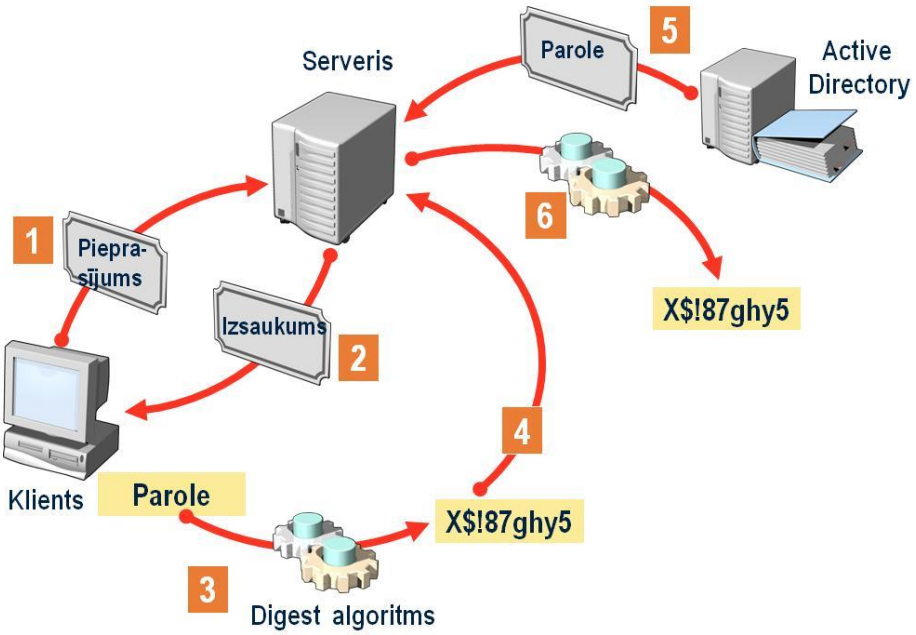


- vienkārša un efektīva (HTTP 1.0 protokola RFC 261 sastāvdaļa);
- uztur populārākie tīmekļa serveri un pārlūkprogrammas;
- viegli realizēt;
- strādā ar piekļuves vadības sarakstiem;
- nepieciešams izmantot SSL/TLS.

Digest

- līdzīga *Basic*, tiek uzskatīta par drošāku;
- parole netiek nodota atklāti, bet jaucējfunkcijas veidā, kas izskaitļota ar MD5 algoritma palīdzību;
- vairāk informācijas par metodi: *Microsoft MSDN® Knowledge Base article KB222208 "Setting Up Digest Authentication for Use with Internet Information Services 5.0"*;
- autentificēšanas laikā tiek veiktas šādas darbības (sk. 5.5. att.):
  1) klienta pārlūkprogramma veic seansa pieprasījumu;
  2) serveris pieprasījuma iesākumā neatrod lauku *Authorization*, tāpēc nosūta atpakaļ izsaukumu – autentificēšanās pieprasījumu;

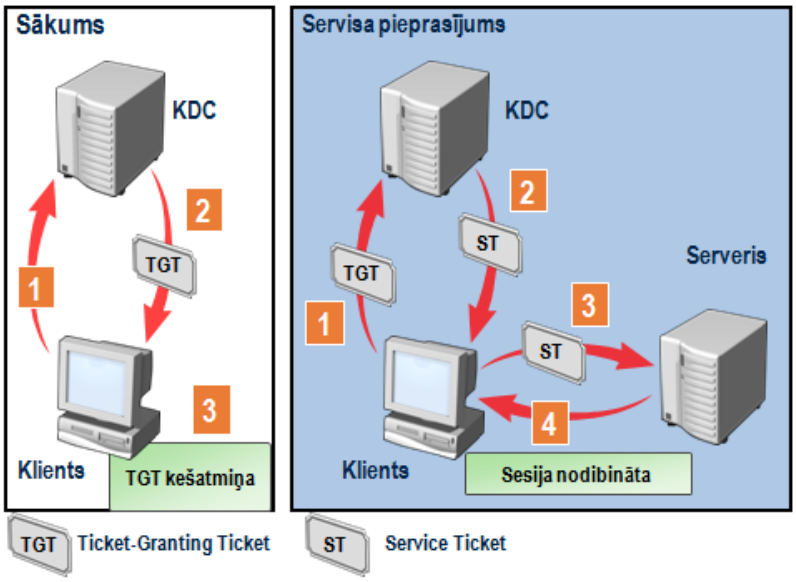
3) pēc lietotāja vārda un paroles ievadīšanas tiek formēta rinda, kas sastāv no lietotāja vārda, paroles un citiem klientam un serverim zināmiem komponentiem. Tiek izskaitļota jaucējfunkcijas vērtība pēc MD5 algoritma;
4) iegūtā jaucējfunkcijas vērtība tiek nosūtīta serverim;
5) serveris veic līdzīgu procedūru, izmantojot lietotāja paroles kopiju no *Active Directory*;
6) serveris salīdzina klienta jaucējfunkcijas vērtību ar servera izskaitļoto, un pozitīva rezultāta gadījumā IIS autentificē klientu un veic tā pieprasījuma apstrādi.



5.5. att. **Autentificēšanas metode *Digest***

Kerberos

- klienta un servera savstarpējās autentificēšanās protokols pirms savienojuma;
- trīs dalībnieki: klients, serveris un starpnieks - atslēgu izdalīšanas centrs (*Key Distribution Center – KDC*);
- Shematiski *Kerberos* protokola darbības shēma ir šāda (sk. 5.6. att.):
  1) sākumā, kad klients pirmo reizi reģistrējas sistēmā, parole tiek apstrādāta ar jaucējfunkcijas algoritmu DES-CBC-MD5, kā rezultātā tiek ģenerēta ilglaicīga kriptogrāfiskā atslēga, kas tiek ievietota KDC datu bāzē;
  2) KDC atbild klientam ar tā saucamo TGT mandātu (*Ticket-Granting Ticket*), kas satur laikspiedogu (*timestamp*) un klienta autentificēšanās informāciju;
  3) klients ar savu privāto atslēgu atšifrē TGT un ievieto to kešatmiņā. TGT satur unikālu sesijas atslēgu (*session key*), kas nepieciešama transakciju nodrošināšanai ar KDC.
- Servera servisa pieprasījuma gadījumā:
  1) kad ir nepieciešamība savienoties ar serveri, klienta TGT ar sesijas atslēgu tiek sūtīts uz KDC, kur notiek klienta mandāta pārbaude;
  2) KDC ģenerē seansa mandātu (*service ticket*), kurš satur sesijas atslēgu un informāciju par klientu. Seansa mandāts tiek šifrēts ar ilglaicīgu kriptogrāfisko atslēgu, kuru zina tikai KDC dienests un serveris. Seansa mandāts tiek nogādāts klientam;
  3) klients seansa mandātu sūta serverim sava pieprasījuma izpildei. Serveris atšifrē klienta seansa mandātu un iegūst sesijas atslēgu, kuru izmanto klienta atļauju pārbaudei;
  4) Ja viss ir kārtībā, serveris konstatē, ka seansa mandātu ir izdevis likumīgs starpnieks – KDC, un sesija ar klientu var tikt nodibināta.



5.6. att. **Autentificēšanas metode *Kerberos***

Līdzīgi autentificēšanai arī pilnvarošanā pastāv vairākas vispārīgas metodes:

- IIS tīmekļa atļaujas (un IP/DNS ierobežojumi);
- .NET drošība lomu līmenī;
- .NET drošība koda līmenī;
- NTFS piekļuves vadības saraksti;
- *Windows Server Authorization Manager*;
- SQL *Server* atļaujas.

5.2. Lomās balstītas drošības ieviešana ar *Identity* un *Principal* objektiem

Resursu (mapes, datnes) pārvaldība tiek veikta ar ACL sarakstu palīdzību. Taču operētājsistēmas objekti nav vienīgie resursi, kas būtu jāaizsargā. Bieži nepieciešams aizsargāt arī pašu lietotnes izpildes loģiku. Pilnvarošanas veikšanai var izmantot .NET *Framework* klases `WindowsIdentity` un `WindowsPrincipal`, kas ļauj realizēt piekļuvi atsevišķām lietotnes funkcijām tikai noteiktiem lietotājiem vai grupām.

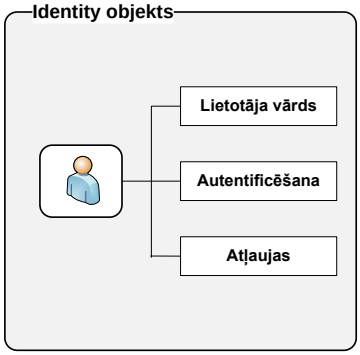
Identitāte (*identity*) – .NET *Framework* objekts, kas iekapsulē informāciju par lietotāju.

Identitātes objekti:

- *Windows* identitāte;
- vispārējā identitāte;
- individuālā identitāte.

Visas identitātes klases īsteno identitātes saskarni (*Identity Interface*):

- `Name`
- `isAuthenticated`
- `Authentication Type`



Klase `System.Security.Principal.WindowsIdentity` pārstāv *Windows* lietotāja kontu. Šī klase netiek izmantota lietotāja autentificēšanai, jo tā jau ir veikta ar *Windows* līdzekļiem.

Klase ļauj piekļūt lietotāja vārdam, loģiskajam mainīgajam, kurš uzdod autentificēšanās pazīmi, un autentificēšanās tipam, t. i., klase tikai glabā autentificēšanas rezultātu.

Vispārīgā gadījumā `WindowsIdentity` klases eksemplāra izveidošanai tiek izsaukta viena no trim metodēm:

1) `GetAnonymous` – atgriež objektu `WindowsIdentity` anonīmam autentificēšanu neizgājušam *Windows* lietotājam. Šo metodi izmanto, lai kodu varētu izpildīt anonīms lietotājs;
2) `GetCurrent` – atgriež objektu `WindowsIdentity` pašreizējam lietotājam. Metodi izmanto, lai pārbaudītu lietotāja vārdu un dalību lietotāju grupās;
3) `Impersonate` – atgriež objektu `WindowsImpersonationContext` noteiktam lietotājam.

`WindowsIdentity` objektu pašreizējam lietotājam var uzdot, piemēram, šādā veidā: `WindowsIdentity currentIdentity = WindowsIdentity.GetCurrent();`

Pēc mainīgā vērtības piešķiršanas var iegūt šādu informāciju par lietotāju:

- `AuthenticationType` – autentificēšanās metodes tipa vērtība;
- `IsAnonymous` – vērtība ‘true’, ja lietotājs anonīms;
- `IsAuthenticated` – vērtība ‘true’, ja lietotājs ir autentificējies;
- `IsGuest` – vērtība ‘true’, ja lietotājs ir viesa statusā;
- `IsSystem` – vērtība ‘true’, ja lietotājs ir sistēmas sastāvdaļa;
- `Name` – domēna un lietotāja vārds formā *DOMĒNS\Lietotāja_vārds*. Ja lietotāja konts atrodas lokālajā datorā, tad kā domēns tiks uzrādīts datora vārds, pretējā gadījumā tiks uzrādīts *Active Directory* domēna vārds;
- `Token` – vesels skaitlis, kas raksturo speciālu lietotāja autentificēšanās marķieri. To iestata autentificēšanu veikušais dators.

5.1. piemērs. Informācijas attēlošana par pašreizējo lietotāju

```
using System;
using System.Security.Principal;

namespace autentificesana
{
   class Program
   {
      static void Main(string[] args)
      {
         // Iegūst info par pašreizējo lietotāju
         WindowsIdentity currentIdentity = WindowsIdentity.GetCurrent();
         Console.WriteLine("Vārds: " + currentIdentity.Name);
         Console.WriteLine("Autentificēšanas tips: " + currentIdentity.AuthenticationType);
         Console.WriteLine("Marķieris: " + currentIdentity.Token.ToString());

         // Lietotāja statuss
         if (currentIdentity.IsAnonymous)
            Console.WriteLine("Anonīms lietotājs");
         if (currentIdentity.IsAuthenticated)
            Console.WriteLine("Autentificēts lietotājs");
         if (currentIdentity.IsGuest)
            Console.WriteLine("Viesis lietotājs");
         if (currentIdentity.IsSystem)
            Console.WriteLine("Sistēmas lietotājs");
         Console.ReadLine();
      }
   }
}
```

Rezultāts

```
C:\projects\ConsoleApp4\ConsoleApp4\bin\Debug\net8.0\ConsoleApp4.exe
Vārds: K3-123861279\Peteris_Grabusts
Autentificēšanas tips: NTLM
Markieris: 908
Autentificēts lietotājs
```

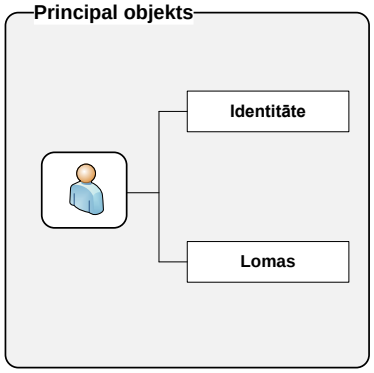
Pilnvaras (*principal*) – .NET *Framework* objekts, kas iekapsulē informāciju par lietotāja lomām. Pilnvaras veido identitātes un lomu kompozīcija.

.NET *Framework* uz lomām bāzētas drošības pilnvaras ir:

- *Windows* pilnvaras;
- vispārējās pilnvaras;
- individuālās pilnvaras.

Pilnvaras saskarni īsteno *Identity Interface*:

- `IsInRole`.



Klase `System.Security.Principal.WindowsPrincipal` nodrošina piekļuvi ziņām par lietotāja dalību grupās. Šī klase jāveido ar klases `WindowsIdentity` eksemplāra palīdzību šādā veidā:

```
WindowsIdentity currentIdentity = WindowsIdentity.GetCurrent();
WindowsPrincipal currentPrincipal = new WindowsPrincipal(currentIdentity);
```

Drošības sistēmas dalībnieku politika ir shēma, ar kuras palīdzību .NET *Framework* nosaka, par kādu dalībnieku pēc noklusējuma būs jādod informācija lietotnes pieprasījuma gadījumā. `System.Security.Principal.PrincipalPolicy` nosaka trīs politikas shēmas:

1) `NoPrincipal` – shēma bez iebūvētās funkcionalitātes;
2) `UnauthenticatedPrincipal` – politikas shēma pēc noklusējuma, kurā tiek pieņemts, ka visi drošības dalībnieki nav autentificēti;
3) `WindowsPrincipal` – politikas shēma, kas nepieciešama darbam ar *Windows* drošības sistēmu.

Tā kā shēma `WindowsPrincipal` nav shēma pēc noklusējuma, darba sākumā to nepieciešams padarīt aktīvu šādā veidā:

```
AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal);
```

Klase `WindowsPrincipal` satur vienu metodi `IsInRole`, kas ļauj noteikt, vai lietotājs ir noteiktas grupas loceklis. Šim nolūkam nepieciešams veikt piederības pārbaudi konkrētai grupai, nododot metodei vienu no trim parametriem:

- grupas `WindowsBuiltInRole` objekts;
- autentificēšanas domēns un grupas nosaukums (teksta rindas veidā);
- grupas identifikators (vesela skaitļa veidā).

Ar klases `WindowsPrincipal` palīdzību var noteikt, kādās grupās iekļauts lietotājs. Lai varētu veikt piederības pārbaudi, metodei `WindowsPrincipal.IsInRole` jānodod klases `System.Security.Principal.WindowsBuiltInRole` loceklis. Katrs klases `WindowsBuiltInRole` loceklis pārstāv grupu, kas pastāv vai nu lokālajā datorā, vai arī domēnā

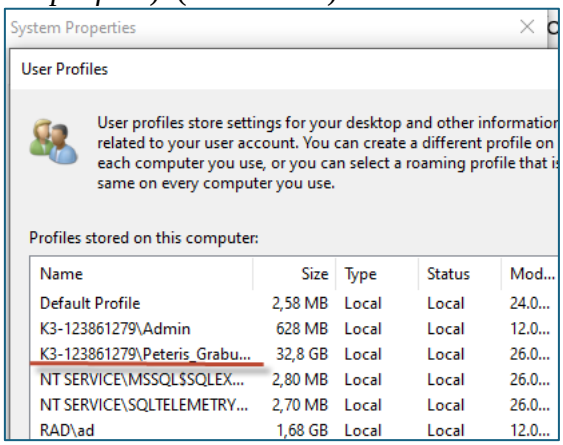
Active Directory. Atkarībā no tā, kādā datorā palaista asambleja, var gadīties, ka atsevišķas lomas var nepastāvēt vispār, t. i., dažādās operētājsistēmās grupas atšķiras, kas var izsaukt izņēmuma situāciju `System.ArgumentException`. 5.1. tabulā uzskaitīti klases `WindowsBuiltInRole` locekļi un atbilstošās grupas un to piederības apgabals.

5.1. tabula

Klases `WindowsBuiltInRole` locekļi

Loceklis	Grupa	Apgabals
AccountOperator	*Account Operators*	Domēns
Administrator	*Administrators*	Lokāls
BackupOperator	*Backup Operators*	Lokāls
Guest	*Guests*	Lokāls
PowerUser	*Power Users*	Lokāls
PrintOperator	*Print Operators*	Lokāls
Replicator	*Replicator*	Lokāls
SystemOperator	*Server Operators*	Domēns
User	*Users*	Lokāls

Nākamā piemēra izpildei nepieciešams zināt lietotāja vārdu un domēnu. To var noskaidrot ar 5.1. piemēra palīdzību vai apskatot, piemēram, savu lietotāja profilu (*Control Panel /System/ Advanced system settings/ User profiles*) (sk. 5.7. att.).



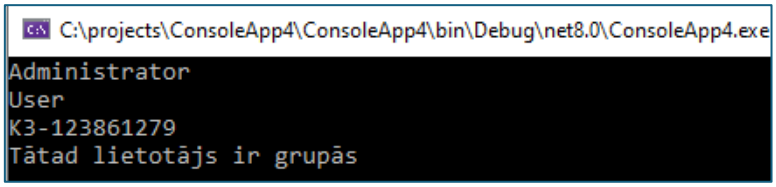
5.7. att. **Lietotāja profils konkrētajā datorā**

5.2. piemērs. Lietotāja piederības grupai pārbaude

```
using System;
using System.Security.Principal;
namespace principal
{
   class Program
   {
      static void Main(string[] args)
      {
         WindowsIdentity currentIdentity = WindowsIdentity.GetCurrent();
         WindowsPrincipal currentPrincipal = new WindowsPrincipal(currentIdentity);

         // Pārbauda dalību divās bieži izmantotās grupās
         if (currentPrincipal.IsInRole(WindowsBuiltInRole.Administrator))
         {
           Console.WriteLine(WindowsBuiltInRole.Administrator.ToString());
         }
```

```
        if (currentPrincipal.IsInRole(WindowsBuiltInRole.User))
        {
           Console.WriteLine(WindowsBuiltInRole.User.ToString());
        }
        // Domēna vārds
        Console.WriteLine(System.Environment.UserDomainName.ToString());
         Console.WriteLine("Tātad lietotājs ir grupās");
         Console.ReadLine();
      }
   }
}
```

Rezultāts:



Piemērā tika konstatēta konkrētā datora lietotāja piederība *Administrator* un *User* grupai. Lai varētu pārbaudīt piederību administratora izveidotajās vai domēna grupās, metodei `IsInRole` jānodod rinda veidā *DOMĒNS\Grupas_vārds*.

Daudzos gadījumos datora vārds un domēna vārds nav zināms, tad var izmantot `System.Environment.MachineName` un `System.Environment.UserDomainName`.

5.3. Lomās balstītas drošības ieviešana ar *Permission* objektu palīdzību

Lomās balstītas drošības pasākumu ieviešanā bieži izmanto RBS deklaratīvos un imperatīvos priekšrakstus.

> Deklaratīvie priekšraksti (*declarative*) – nosaka .NET *Framework* videi veikt drošības pasākumu pārbaudi pirms metodes izpildes. Tas ir visdrošākais veids ierobežot piekļuvi kodam, jo izpildes vide veic drošības pārbaudi pirms koda izpildes.

Klase `System.Security.Permissions.PrincipalPermission` un ar to saistītā klase `PrincipalPermissionAttribute` ļauj pārbaudīt aktīvo drošības sistēmas dalībnieku deklaratīvajām un imperatīvajām darbībām. Šīs klases, ko vispārinot sauc par `PrincipalPermission` klasi, parasti izmanto deklaratīvo priekšrakstu izveidošanai, kuriem nepieciešama lietotāju autentificēšana vai iekļaušana noteiktā lomā. Tādā gadījumā ar klases `PrincipalPermission` palīdzību var pieprasīt, lai lietotājs atbilstu šiem nosacījumiem.

Var izmantot jebkuru kombināciju no trim klases īpašībām:

- `Authenticated` – vērtība ‘true’, ja lietotājam vajag veikt autentificēšanu;
- `Name` – lietotāja vārds;
- `Role` – atbilstošā lietotāja loma.

Klasei `PrincipalPermission` ir vairākas metodes, taču turpmākajā izklāstā no RBS priekšrakstiem tiks izmantota tikai metode `PrincipalPermission.Demand`. Tā pārbauda, vai aktīvais drošības sistēmas dalībnieks atbilst prasībām, kas norādītas īpašībās `Authenticated`, `Name` un `Role`. Ja dalībnieks neatbilst kādai no īpašībām, tiek ģenerēts izņēmuma stāvoklis.

Lai varētu izmantot RBS deklaratīvos priekšrakstus, programmas kodā jābūt trim elementiem:

1) metodei `System.AppDomain.CurrentDomain.SetPrincipalPolicy` – norāda drošības sistēmas dalībnieku politiku;
2) blokam `try/catch` – piekļuves bez attiecīgām atļaujām pārtveršanai;
3) atribūtam `PrincipalPermission` – nosaka piekļuves prasības.

Nākamajā koda fragmenta piemērā tiek izsaukta metode `PetersOnlyMethod`, kas aizsargāta ar RBS deklaratīvo priekšrakstu un izdod paziņojumu, ja lietotājs šajā gadījumā nav *peter*.

5.3. piemērs. Deklaratīvā priekšraksta izmantošana

```
[PrincipalPermission(SecurityAction.Demand, Role = @"peter-PC\peter")]

static void PetersOnlyMethod()
{
  // Kods, kuru var izpildīt tikai peter, piemēram,
   Console.WriteLine("Jums ir tiesibas veikt sho darbibu!");
   Console.ReadLine();
}

static void Main(string[] args)
{

System.AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal);

   try
   {
      PetersOnlyMethod();
   }
   catch (System.Security.SecurityException ex)
   {
      Console.WriteLine("Jums nav tiesību veikt šo funkciju");
      Console.ReadLine();
   }
 }
```

Ja būs cits domēna vai lietotāja vārds, tad, protams, koda fragments *PetersOnlyMethod* netiks izpildīts.

Var izmantot vairākus deklaratīvos priekšrakstus, lai kodu varētu izpildīt lietotājs, kas atbilst vismaz vienai no prasībām. Nākamo koda fragmentu varētu izpildīt tikai šie lietotāji:

- lokālās grupas *Administrators* locekļi;
- lietotājs *CONTOSO\User1*, kas ir grupas *CONTOSO\Managers* loceklis;
- jebkurš autentificējies lietotājs.

5.4. piemērs. Vairāku deklaratīvo priekšrakstu izmantošana

```
[PrincipalPermission(SecurityAction.Demand, Name = @"CONTOSO\Administrator")]
[PrincipalPermission(SecurityAction.Demand, Name = @"CONTOSO\User1", Role =
@"CONTOSO\Managers")]
[PrincipalPermission(SecurityAction.Demand, Authenticated = true)]

private void AdministratorsOnlyMethod()
    {
      // Kods, kuru var izpildīt tikai, piemēram,  lietotājs
      // CONTOSO\Administrator
    }
```

Imperatīvie priekšraksti (*imperative*) – tiek noteikti kodā un var tikt izmantoti detalizētākam piekļuves ierobežojumam koda sastāvdaļām, t. i., ierobežo piekļuvi atsevišķiem metodes komponentiem.

Lai varētu izmantot RBS imperatīvos priekšrakstus, programmas kodā jābūt šādiem četriem elementiem:

1) metodei `System.AppDomain.CurrentDomain.SetPrincipalPolicy` – norāda drošības sistēmas dalībnieku politiku;
2) blokam `try/catch` – piekļuves bez attiecīgām atļaujām pārtveršanai;
3) objektam `PrincipalPermission` – kura īpašībām dotas noteiktas vērtības, atbilstošas ierobežojumiem;
4) metodei `PrincipalPermission.Demand` – nosaka piekļuves prasības.

Pirmie divi elementi pilnībā sakrīt ar RBS deklaratīvo priekšrakstu elementiem, taču klases `PrincipalPermission` izmantošana radikāli atšķiras. Tagad vajag izveidot jaunu klases `PrincipalPermission` objektu ar trim konstruktoriem:

- `PrincipalPermission(PermissionState)` – ļauj norādīt klases `PrincipalPermission` objekta īpašības, izmantojot objektu `System.Security.Permissions.PermissionState`;
- `PrincipalPermission(Name, Role)` – norāda jaunā objekta īpašību `Name` un `Role` vērtības. Ja nepieciešams norādīt tikai lietotāja vārdu vai lomu, otrajam operandam uzdod vērtību `null`;
- `PrincipalPermission(Name, Role, Authenticated)` – norāda jaunā objekta īpašību `Name`, `Role` un `Authenticated` vērtības. Ja kāda no īpašībām netiek izmantota, tās vērtību aizstāj ar `null`.

Ja lietotājs, piemēram, ir grupas *peter* loceklis, tad ar imperatīvo priekšrakstu palīdzību tam tiks atļauts izpildīt noteiktu koda fragmentu, kā tas parādīts 5.5. piemērā.

5.5. piemērs. Imperatīvā priekšraksta izmantošana

```
System.AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal);
    string requiredRole = System.Environment.MachineName + @"\peter";
    // vai: string requiredRole = @"peter-PC\peter";
    try
    {
      PrincipalPermission administratorPermission =
            new PrincipalPermission(null, requiredRole, true);
      administratorPermission.Demand();
      Console.WriteLine("Pieeja atlauta");
      // Te varētu būt programmas kods
      Console.ReadLine();
    }
    catch (System.Security.SecurityException ex)
    {
      Console.WriteLine("Pieeja liegta");
      // Te varētu būt kļūdu apstrādes modulis
      Console.ReadLine();
    }
```

Kā redzams no piemēriem, pastāv trīs dažādas iespējas pārbaudīt lietotāja piederību noteiktai grupai:

1) metode `WindowsPrincipal.IsInRole`;

2) RBS deklaratīvie priekšraksti;
3) RBS imperatīvie priekšraksti.

Katra no šīm iespējām kalpo vienam mērķim – programmas izpildes secības izmainīšana atkarībā no lietotāja piederības noteiktai grupai. Tomēr katra no tām tiek izmantota saviem mērķiem, kā tas parādīts 5.2. tabulā.

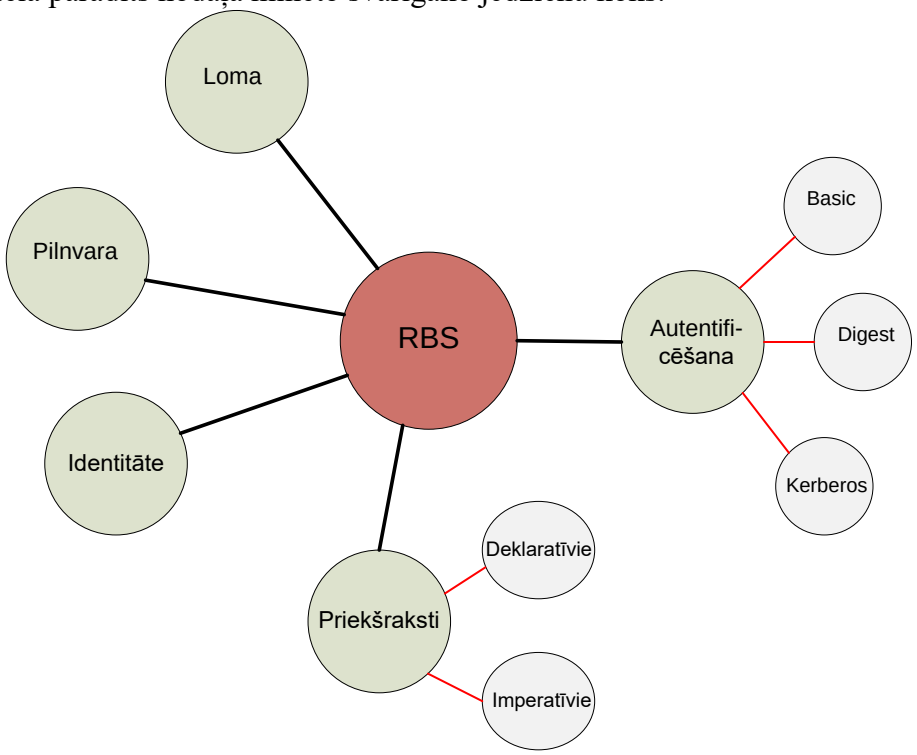
5.2. tabula

RBS pārbaudes iespējas

Nosacījums	Rekomendējamais pārbaudes veids
Ierobežot piekļuvi metodei pilnībā (pie statiski noteiktiem nosacījumiem)	RBS deklaratīvie priekšraksti
Ierobežot piekļuvi metodei pilnībā vai daļēji (pie statiski vai dinamiski noteiktiem nosacījumiem)	RBS imperatīvie priekšraksti
Pamainīt programmas izpildes secību (atkarībā no lietotāja piederības grupai)	Metode `WindowsPrincipal.IsInRole`

Ja iespējami vairāki RBS ieviešanas scenāriji, no sākuma ieteicams pielietot deklaratīvos priekšrakstus, tad imperatīvos, bet metodi `WindowsPrincipal.IsInRole` ieteicams izmantot tikai galējas nepieciešamības gadījumos. Tas saistīts ar to, ka izņēmuma gadījumu apstrāde ir pats drošākais veids neļaut izmantot metodi lietotājam, kam nav pietiekamu pilnvaru.

5.8. attēlā parādīts nodaļā minēto svarīgāko jēdzienu koks.



5.8. att. **Piektajā nodaļā minēto svarīgāko jēdzienu koks**

Pārbaudes testa jautājumi

1. Kāda veida atribūts jāpievieno kodam, lai deklarētu prasību, ka lietotājs ir iekļauts īpašā lomā vai arī tas ir autentificējies?
   a. *PermissionAttribute*
   b. *PrincipalAttribute*
   c. *PermissionPrincipalAttribute*
   d. *PrincipalPermissionAttribute*

2. Kurš no apgalvojumiem vislabāk apraksta iespēju veikt imperatīvo RBS pārbaudi ar objekta atļaujām?
    a. Izveidot un inicializēt *PrincipalPermission* objektu, un tad izsaukt *Demand* metodi
    b. Izveidot un inicializēt vispārēju *Permission* objektu, un tad izsaukt *Demand* metodi
    c. Izveidot un inicializēt vispārēju *Permission* objektu un tad izsaukt *PrincipalDemand* metodi

3. Kurš no apgalvojumiem par pilnvaru objektiem ir patiess?
    a. *WindowsPrincipal* klase pārstāv *Windows* lietotājus un to lomas.
    b. *GenericPrincipal* pārstāv tos lietotājus un lomas, kas pastāv neatkarīgi no *Windows* lietotājiem un to lomām.

4. .NET lietotne satur šādu procedūru:

```
01 public void TestProc(string U1, string R1,
02  string U2, string R2) {
03 PrincipalPermission Perm1 =
04 new PrincipalPermission(U1, R1);
05 PrincipalPermission Perm2 =
06 new PrincipalPermission(U2, R2);
07 // Insert new code.
08 // Additional procedure code goes here.
09 }
```

   Kurš no koda fragmentiem varētu būt ielikts 9. rindā, lai nodrošinātu abiem lietotājiem *User1* un *User2* kopīgas drošības lomas?
    a. *Perm1*.*IsUnrestricted*; *Perm2*.*IsUnrestricted*
    b. *Perm1*.*IsSubSetOf*(*Perm2*)
    c. *Perm1*.*Intersect*(Perm2).*Demand*()
    d. *Perm1*.*Union*(*Perm2*).*Demand*()

5. Ar .NET un SQL *Server* tiek izstrādāta datubāzes lietotne. Lietotnē drošības pārbaudei bieži nākas izmantot *Assert*, *Deny* un *PermitOnly*. Ir nepieciešamība optimizēt lietotnes izpildes ātrumu.
   Kādā veidā vislabāk iekļaut drošības pārbaudi?
    a. Pielietot deklaratīvo drošības pārbaudi
    b. Pielietot imperatīvo drošības pārbaudi
    c. Izmantot SQL *Server* drošības pārbaudes nodrošināšanai
    d. Iegūt drošības informāciju no SQL *Server* datubāzes, izmantojot klienta rakstītās drošības klases

6. Koda piekļuves aizsardzības ieviešana

Koda piekļuves aizsardzība ir jauna drošības koncepcija un ļauj lietotājam norādīt kodam atļautās darbības. Koda piekļuves aizsardzība iespējama tikai pārvaldītam kodam, t. i., kodam, kas uzrakstīts atbilstoši CLR specifikācijām. Tādas aizsardzības arhitektūra konceptuāli balstās uz diviem galvenajiem komponentiem: atļauju kopas un koda grupas.

Nodaļā apskatīti koda piekļuves drošības pamati un vienkāršu aizsardzības darbību veikšana. Tiek izskaidrota imperatīvo un deklaratīvo aizsardzības operāciju pielietošana koda piekļuves aizsardzības īstenošanā.

6.1. Koda piekļuves drošības pamati

Koda piekļuves aizsardzība (*Code Access Security – CAS*) ir drošības sistēma, kas ļauj administratoram un izstrādātājam veikt lietotnes pilnvarošanu. Tā balstās ne uz lietotāju identificēšanu, bet uz izpildāmā koda identificēšanu (kodam atļautās operācijas un pieejamie resursi ir ierobežoti saskaņā ar drošības apliecinājumiem) [24], [25].
Diemžēl, .NET *Core* vairs pilnā mērā neuztur CAS [26].

Jebkurā drošības sistēmā jābūt iespējai identificēt lietotājus un noteikt lietotāju tiesības. Taču CAS identificē un piešķir atļaujas lietotnēm, nevis cilvēkiem, tādējādi nevar izmantot pierastos lietotāja vārdus, paroles un ACL sarakstus. Tā vietā CAS identificē asamblejas ar drošības apliecinājumu palīdzību. Par asamblejas identificēšanas apliecinājumu var kalpot mape, kurā uzglabājas asambleja, vai, piemēram, stingrs vārds. Asamblejas apliecinājums nosaka to, kādai koda grupai asambleja pieder.

CAS aizsardzības elementi ir šādi:

- drošības apliecinājums (*evidence*);
- atļauja (*permission*);
- atļauju kopas (*permission sets*);
- koda grupas (*code groups*).

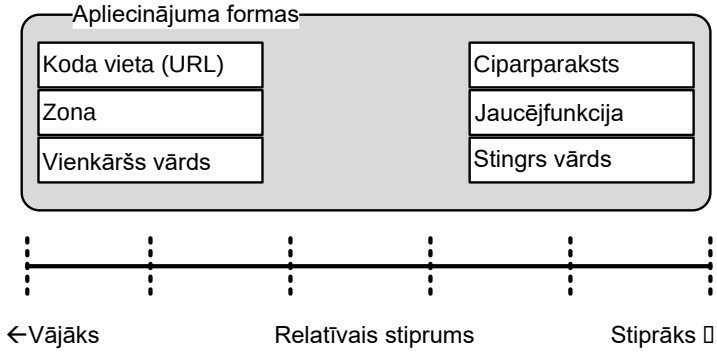
Drošības apliecinājums ir ziņas par asambleju, ko vāc CLR, lai noteiktu, kādām koda grupām pieder asambleja. Parasti par apliecinājumu kalpo ciparparaksts un atrašanās vieta (mape vai *Internet* tīkls), no kurienes palaista asambleja. 6.1. tabulā uzskaitīti drošības apliecinājumu tipi un to apraksts.

6.1. tabula

Drošības apliecinājuma tipi

Drošības apliecinājums	Apraksts
Lietotnes mape	Mape, kurā atrodas asambleja
Jaucējfunkcija	Konsistences pārbaude. Asamblejas kriptogrāfiskā jaucējfunkcija, kas viennozīmīgi identificē asamblejas versiju. Jebkuras izmaiņas asamblejā padara jaucējfunkciju par nederīgu
Izdevējs	Asamblejas izdevēja ciparparaksts
Vietne	Vietne, no kurienes tiek ielādēta asambleja
Stingrs vārds	Kriptogrāfiski stingrs asamblejas nosaukums (nosaukums, versija, publiskā atslēga)
URL	URL adrese, no kurienes tiek ielādēta asambleja
Zona	Zona, kurā palaista asambleja, piem., *Internet* vai *Local Intranet*

Drošības apliecinājuma formu relatīvais stiprums attēlots 6.1. attēlā.



6.1. att. **Apliecinājuma formu relatīvais stiprums**

Izšķir divu veidu drošības apliecinājumus:

- īpašnieka apliecinājumu;
- asamblejas apliecinājumu.

Īpašnieks jeb saimnieks (*host*) ir dators vai programma, kas nodrošina informatīvos pakalpojumus citiem tīkla datoriem vai programmām. Īpašnieka apliecinājums nosaka asamblejas izcelsmi, piem., mape, vietne vai URL. Tas var saturēt arī asamblejas identifikācijas datus – konsistenci, izdevēju vai stingro vārdu.

Asamblejas apliecinājums ir īpašs apliecinājums, ko piešķir izstrādātājs.

Atļauja – tas ir CAS piekļuves vadības elements. Piemēram, atļauja *File Dialog* nosaka, vai asambleja var atļaut lietotājam atvērt dialoga logus **Open** un **Save**.

.NET *Framework* uztur daudzas iebūvētas koda piekļuves atļauju klases sistēmas resursu aizsardzībai (sk. 6.2. tabulu).

6.2. tabula

Koda piekļuves atļauju klases

Koda piekļuves atļauju klase	Aizsargātie resursi
`DirectoryServicesPermission`	Nodrošina asamblejai piekļuvi pie *Active Directory*. Var norādīt ceļu un atļaut/aizliegt piekļuvi apskatei un ierakstam
`DnsPermission`	Atļauj vai aizliedz asamblejai izdarīt DNS pieprasījumus
`EnvironmentPermission`	Dod iespēju asamblejām piekļūt pie mainīgajiem *Path*, *Username* un *Number_Of_Processors*
`EventLogPermission`	Nodrošina asamblejai piekļuvi notikumu žurnāliem.
`FileDialogPermission`	Nosaka, vai asamblejai ir tiesības atļaut lietotājam dialoga logus **Open** un **Save**
`FileIOPermission`	Ierobežo piekļuvi datnēm un mapēm
`IsolatedStorageFilePermission`	Nodrošina asamblejām piekļuvi izolētām glabātavām
`MessageQueuePermission`	Dod iespēju asamblejām vērsties pie ziņojumu rindām
`OleDbPermission`	Uzskaita asamblejai pieejamos OLE DB pakalpojumu sniedzējus
`PerformanceCounterPermission`	Nosaka, vai asambleja var nolasīt vai ierakstīt datus veiktspējas skaitītājos
`PrintingPermission`	Ierobežo asamblejas drukāšanas iespējas
`ReflectionPermission`	Nosaka, vai asambleja var izmantot informāciju no citām asamblejām
`RegistryPermission`	Ierobežo piekļuvi reģistra sadaļām

Atļauju kopas faktiski ir CAS ACL saraksts. Piemēram, *Internet* atļauju kopa ietver šādas atļaujas: *File Dialog*, *Isolated Storage File*, *Security*, *User Interface* un *Printing*.

Zona *LocalIntranet* ietver vairāk atļauju, pieņemot, ka kods lokālajā tīklā ir vairāk uzticams nekā kods *Internet* tīklā:

- *DNS*	- *Environment Variables*	- *Event Log*
- *File Dialog*	- *Isolated Storage File*	- *Printing*
- *Reflection*	- *Security*	- *User Interface*

.NET *Framework* iekļautas septiņas standarta atļauju kopas (sk. 6.3. tabulu).

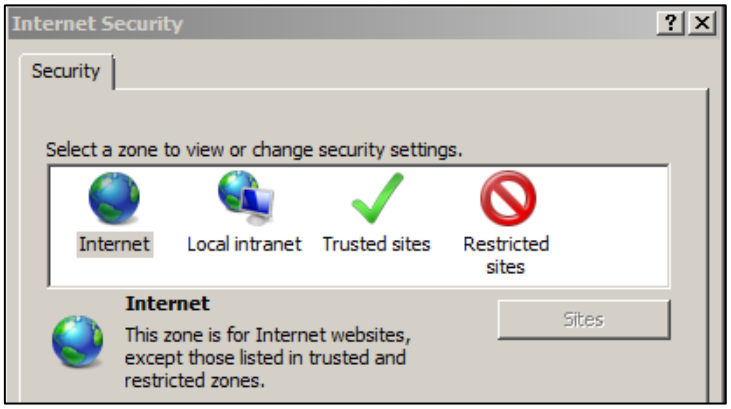
6.3. tabula

Standarta atļauju kopas

Atļauju kopa	Apraksts
FullTrust	Atbrīvo asambleju no CAS atļauju pārbaudes.
SkipVerification	Atļauj asamblejai izlaist atļauju pārbaudi (var palielināt veiktspēju uz drošības rēķina).
Execution	Atļauj asamblejas izpildi, bet nepiešķir citas atļaujas.
Nothing	Nepiešķir asamblejai nekādas atļaujas (nav atļauta pat asamblejas izpilde).
LocalIntranet	Piešķir asamblejai daudz atļauju, taču piekļuve datņu sistēmai atļauta tikai ar dialoga logu **Open** un **Save** palīdzību.
Internet	Piešķir asamblejai ierobežotu atļauju skaitu. Parasti šādu asambleju var palaist ar minimālu risku.
Everything	Piešķir asamblejai visas atļaujas. Atšķirībā no *FullTrust* – iziet CAS pārbaudi.

Koda grupas ir pilnvarošanas līdzeklis, kas saista asamblejas ar atļauju kopām, t. i., koda grupa nodrošina saviem dalībniekiem pieeju kādai noteiktai atļauju kopai. CAS koda grupas kalpo tādam pašam nolūkam kā RBS lietotāju grupas. Piemēram, ja administrators grib vairākiem lietotājiem atļaut piekļuvi mapei, viņš izveido grupu, pievieno šai grupai lietotājus un nosaka šai grupai atļaujas. Koda grupas darbojas tāpat, tikai ar to izņēmumu, ka grupai nav nepieciešams manuāli pievienot atsevišķas asamblejas. Tā vietā dalība grupās tiek noteikta ar drošības apliecinājumu.

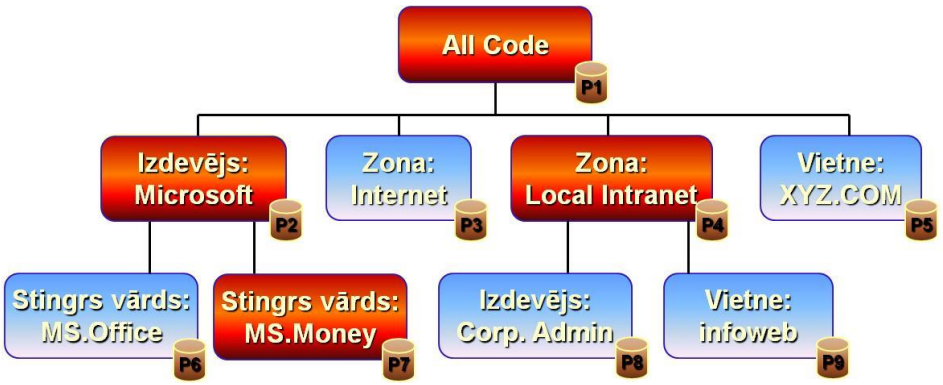
Piemēram, jebkuram *Internet* tīklā palaistajam kodam vajag ietilpt koda grupā *Internet_Zone*. Nosacījums piederībai kodu grupai *Internet_Zone* ir zonas apliecinājums, kas identificē asambleju kā palaistu *Internet* tīklā (sk. 6.2. att.).



6.2. att. **Apliecinājums piederībai kodu grupai *Internet_Zone***

Lietotnes, kas palaistas *Internet* un *LocalIntranet* zonās, nesaņem atļauju *FileIOPermission* un tādējādi nevar tiešā veidā piekļūt datnēm. Taču tām ir atļauja *FileDialogPermission*, t. i., asamblejas *Internet* zonā var atvērt datnes, piedāvājot lietotājam izvēlēties datni ar objekta *OpenFileDialog* palīdzību. Asamblejas *LocalIntranet* zonā var arī saglabāt datnes, izmantojot objektu *SaveFileDialog*.

Iespēja uzrādīt tikai vienu drošības apliecinājuma tipu un tikai vienu atļauju kopu kodu grupai var likties ierobežojoša. Taču, tāpat kā lietotāja konts var tikt iekļauts vairākās grupās, arī asambleja var tikt iekļauta vairākās koda grupās. Asambleja iegūs visas atļaujas, kas piešķirtas katrai no kodu grupām (to sauc par atļauju kopu apvienojumu). Piemēram, lietotne *MS.Money* atrodas *LocalIntranet* zonā, tai ir zināms izdevējs un stingrs vārds. Piešķirtās atļaujas ir kopu apvienojums P1 ∪ P2 ∪ P7 ∪ P4 (sk. 6.3. att.).



6.3. att. **Četru atļauju kopu apvienojuma piemērs**

6.4. tabulā uzskaitītas standarta kodu grupas, kas ietilpst kodu grupā *All_Code*.

6.4. tabula

Standarta kodu grupas

Koda grupa	Drošības apliecinājums	Atļauju kopa
My_Computer_Zone	*Zone: My Computer*	*FullTrust*
LocalIntranet_Zone	*Zone: Local Intranet*	*LocalIntranet*
Internet_Zone	*Zone: Internet*	*Internet*
Restricted_Zone	*Zone: Untrusted sites*	*Nothing*
Trusted_Zone	*Zone: Trusted sites*	*Internet*

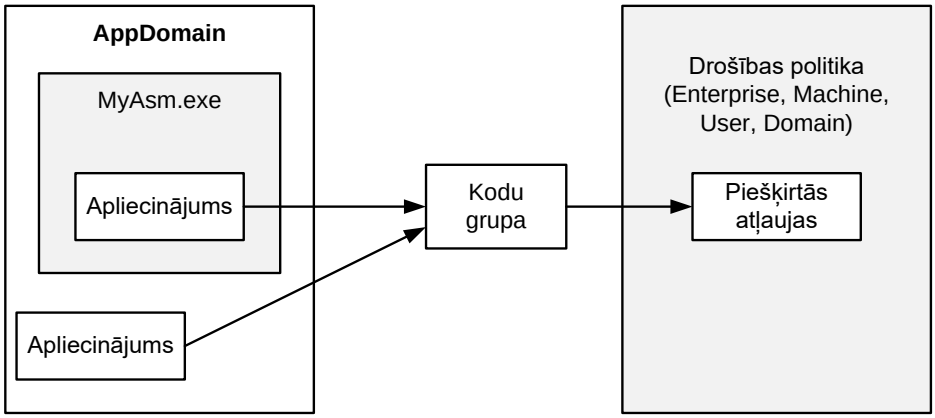
Tādējādi var runāt par noteiktu drošības politiku, kas loģiski apvieno koda grupas un atļauju kopas. Turklāt drošības politika var arī saturēt asamblejas, kas pakļautas citiem politikas tipiem. Drošības politika ļauj administratoriem nodrošināt CAS vairākos līmeņos. Pēc noklusējuma pastāv četri politikas līmeņi:

1) uzņēmuma (*Enterprise*) līmenis – visaugstākais. To var iestatīt ar *Active Directory* dienestu palīdzību;
2) mašīnas jeb datora (*Machine*) līmenis – tiek pielietots visam kodam, kas izpildās konkrētajā datorā;
3) lietotāja (*User*) līmenis – tiek noteiktas atsevišķu lietotāju atļaujas;
4) lietotņu domēna (*Application Domain*) līmenis – viszemākais. Tiek izmantots asambleju pārvaldībai lietotņu domēnos (sk. 4.2. apakšnodaļu).

CLR pārbauda katru līmeni atsevišķi un piešķir asamblejai konkrētam līmenim atbilstošu minimālo atļauju kopu (to sauc par atļauju kopu šķēlumu).

Pēc noklusējuma uzņēmuma un lietotāja līmeņos visam kodam tiek piešķirta pilnīga uzticība *FullTrust*, t. i., netiek veiktas CAS atļauju pārbaudes, tādējādi CAS atļaujas tiek uzturētas datora līmenī.

CAS aizsardzības elementu saistība ar drošības politiku ir parādīta 6.4. attēlā.

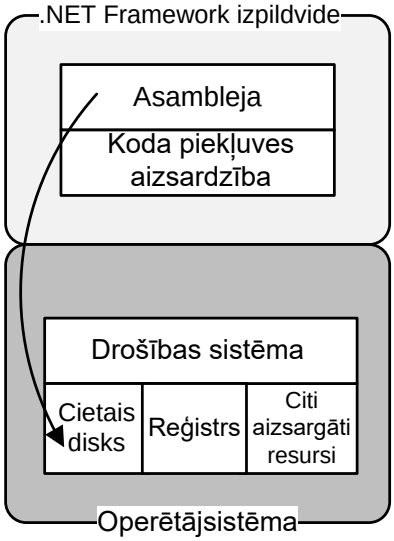


6.4. att. **CAS nodrošināšanas bloki**

Vairāku līmeņu drošības politikas izmantošana tiks paskaidrota ar piemēru.

Pieņem, ka lietotājs ielādē no *Internet* tīkla asambleju. Asambleja tiek lejupielādēta lokālajā datorā, t. i., zonā *My Computer*. Dators ietilpst domēnā *Active Directory*, un šī domēna administrators uzņēmuma līmeņa drošības politikā izveidojis koda grupu, kas piešķir lokāla datora asamblejām atļauju kopu *Everything*.

CAS darbojas virs pastāvošās operētājsistēmas drošības. 6.5. attēlā parādīts, kādā veidā CAS mijiedarbojas ar operētājsistēmas drošības sistēmu. CAS papildina, bet neaizvieto lomās balstīto aizsardzību. Ja, piemēram, CAS atļauj asamblejai ierakstu mapē *C:\Windows*, bet lietotājam, kurš palaida šo asambleju, nav tam atļaujas, tad asambleja nevarēs ierakstīt datus šajā mapē.



6.5. att. **CAS un operētājsistēmas drošības sistēmu saistība**

Tātad CAS ir pilnīgi neatkarīga no operētājsistēmas drošības sistēmas.

6.2. Atļaujas pieprasījumu veikšana

Tā kā CAS ierobežo lietotnēm dotās atļaujas, tad vienmēr nepieciešams plānot lietotņu darbu kontekstā ar ierobežoto atļauju kopu. Atsevišķos gadījumos CAS aizsardzība var būt tik ierobežojoša, ka lietotnei var nebūt atļauju pat vienkāršu darbību veikšanai. Citos gadījumos lietotnei var būt vairāk atļauju nekā nepieciešams, kas pārkāpj minimālo privilēģiju principu un padara lietotni apdraudētāku.

Pastāv vairāki iemesli CAS atļaujas pieprasījumu izmantošanai.

- .NET Framework izpildes vide nekad nepalaidīs lietotni, ja tai nebūs piekļuves tiesību nepieciešamajiem resursiem. Ja izstrādātājs lietotnē nav iekļāvis iespēju apstrādāt izņēmuma gadījumus, kad asamblejai pietrūkst CAS atļauju, tas var izmantot metodi `SecurityAction.RequestMinimum` nepieciešamo atļauju deklarēšanai. Kad lietotājs mēģinās palaist lietotni, kurai CAS drošības politika nav piešķīrusi nepieciešamās atļaujas, tiks ģenerēts izņēmuma gadījums. Lietotājs, iespējams, nevarēs noteikt tā iemeslu, bet administratoram būtu jāzina cēlonis. Jebkurā gadījumā labāk ieteicams izmantot metodi `SecurityAction.RequestMinimum` nekā pieļaut neapstrādājamus izņēmuma gadījumus.
- Minimālo privilēģiju principa izmantošana samazinās iespēju, ka ļaunprātīgs lietotājs varēs izmantot lietotni nesankcionētu darbību veikšanai, piem., datu izdzēšanai, piekļuvei aizsargātām datnēm vai ļaunprātīgu programmu izplatīšanai. Izmantojot CAS atļaujas pieprasījumus asamblejas atļauju ierobežošanai, var novērst iespēju, ka ļaunprātīgs lietotājs izmantos lietotni piekļuvei tai neparedzētiem datiem.
- Lietotni var palaist ar CAS atļauju ierobežojumiem, tātad arī mazāk uzticamās zonās. Lietotnes izstrādes laikā var izmantot metodi `SecurityAction.RequestOptional`, kura asamblejai piešķirs tikai izstrādātāja norādītās atļaujas.

CAS var ierobežot piekļuvi daudziem resursu tipiem, katram no tiem paredzēta sava klase. 6.5. tabulā uzskaitītas dažas klases, kas tiek izmantotas CAS pieprasījumos (pilnu informāciju par klašu īpašībām var atrast MSDN dokumentācijā).

6.5. tabula

Klases darbam ar CAS pieprasījumiem

Klase	Atļauja
System.Net.WebPermission	Piekļuve *Internet* tīkla resursiem (*Web Access*)
System.Security.Permissions.FileDialogPermission	Dialoga logi piekļuvei datnēm (*File Dialog*)
System.Security.Permissions.FileIOPermission	Datņu ievadizvade (*FileIO*)
System.Security.Permissions.PermissionSet	Drošība (*Security*)
System.Security.Permissions.RegistryPermission	Reģistri (*Registry*)
System.Security.Permissions.SecurityPermission	Drošība (*Security*)
System.Security.Permissions.SiteIdentityPermission	Drošība (*Security*)
System.Security.Permissions.UIPermission	Lietotāja saskarne (*User Interface*)
System.Web.AspNetHostingPermission	Piekļuve *Internet* tīkla resursiem (*Web Access*)

Katrai klasei piemīt savas īpašības un metodes. Parasti tiek izmantotas divas standarta īpašības:

- `Action` – norāda darbības, kuras jāizpilda. Izmanto vienu no trim `SecurityAction` locekļu vērtībām;
- `Unrestricted` – loģiskais mainīgais, kas norāda uz atļauju piekļuvei visām klases atļaujām.

Vispārīgā gadījumā pastāv trīs CAS atļauju pieprasījumu tipi – īpašībai `Action` tiek piešķirta viena no `SecurityAction` locekļu vērtībām:

1) `SecurityAction.RequestMinimum` – pieprasa asamblejas palaišanai nepieciešamās atļaujas. Ja asamblejai nebūs norādītās CAS atļaujas, tiks ģenerēts izņēmuma gadījums `System.Securit.Policy.PolicyException`;

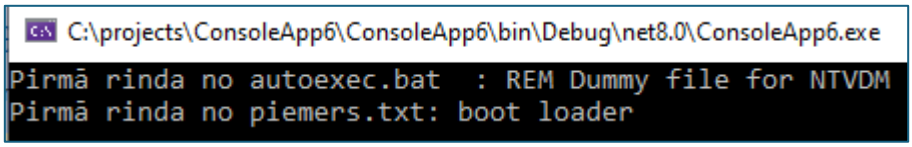
2) `SecurityAction.RequestOptional` – pieprasa tikai uzskaitītās atļaujas, visas pārējās tiek noraidītas. Tiek garantēts, ka lietotnei nebūs vairāk atļauju, kā uzrādījis izstrādātājs. Šo atļaujas pieprasījumu izmanto kopā ar `SecurityAction.RequestMinimum`, ja asambleja nevar strādāt bez atļauju pieprasījuma;
3) `SecurityAction.Refuse` – ierobežo lietotnei piešķirtās atļaujas. Šī tipa atļauju pieprasījums jālieto, piemēram, gadījumos, kad jāizdara tā, lai lietotnei nebūtu piekļuves pie kritiskiem aizsargājamiem resursiem.

Pieņem, ka vēlas nolasīt pirmo rindu no datnes *c:\autoexec.bat* un *c:\Windows\piemers.txt* (datnes *piemers.txt* vietā var izvēlēties patvaļīgu datni no mapes *c:\Windows*). Vienkāršs programmas kods realizē abu datņu pirmo rindu izvadi uz ekrāna.


6.1. piemērs. Informācijas attēlošana bez atļaujas pieprasījuma

```
using System;
using System.IO;
using System.IO.IsolatedStorage;
using System.Security.Permissions;

namespace assembly1
{
   class Program
   {
      static void Main(string[] args)
      {
         StreamReader r1 = new StreamReader(@"c:\autoexec.bat");
         Console.WriteLine("Pirmā rinda no autoexec.bat  : " + r1.ReadLine());
         StreamReader r2 = new StreamReader(@"c:\windows\piemers.txt");
         Console.WriteLine("Pirmā rinda no piemers.txt: " + r2.ReadLine());
         Console.ReadLine();
      }
   }
}
```

Rezultāts

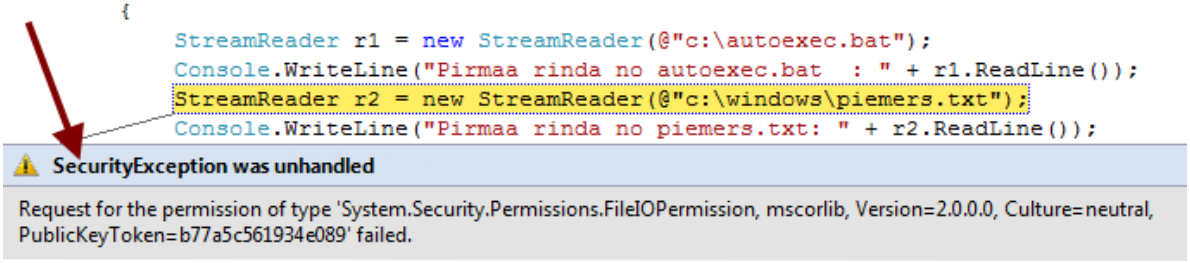


Lietotnes aizsardzībai pievieno divus atļaujas pieprasījumus. Viens no tiem asamblejai atļauj piekļuvi diskam C:\, bet otrs aizliedz nolasīt datus no mapes *c:\Windows*.

```
[assembly:FileIOPermissionAttribute(SecurityAction.RequestMinimum,Read=@"c:\")]
[assembly:FileIOPermissionAttribute(SecurityAction.RequestRefuse,Read=@"c:\windows")]
```

Šajā gadījumā tiek atļauts nolasīt pirmo rindu no datnes *autoexec.bat*, bet, mēģinot nolasīt datus no *C:\Windows\piemers.txt*, vecākajās *Visual Studio* versijās izpildes vide ģenerē izņēmuma gadījumu, kā parādīts 6.6. attēlā.

```
using System;
using System.IO;
using System.IO.IsolatedStorage;
using System.Security.Permissions;

[assembly:FileIOPermissionAttribute(SecurityAction.RequestMinimum,Read=@"c:\")]
[assembly:FileIOPermissionAttribute(SecurityAction.RequestRefuse,Read=@"c:\windows")]

namespace assembly1
{
    class Program
    {
        static void Main(string[] args)
        {
            StreamReader r1 = new StreamReader(@"c:\autoexec.bat");
            Console.WriteLine("Pirmaa rinda no autoexec.bat  : " + r1.ReadLine());
            StreamReader r2 = new StreamReader(@"c:\windows\piemers.txt");
            Console.WriteLine("Pirmaa rinda no piemers.txt: " + r2.ReadLine());
```



6.6. att. **Atļaujas pieprasījuma aizlieguma piemērs**

CAS ietekmē tikai asamblejas ar daļēju uzticību. Izpildes vide pilnībā ignorē CAS atļaujas pieprasījumus asamblejām ar pilnīgu uzticību *FullTrust*.

6.3. Deklaratīvo un imperatīvo aizsardzības operāciju īstenošana

5. nodaļā tika runāts par RBS deklaratīvajiem un imperatīvajiem priekšrakstiem. Arī CAS var izmantot deklaratīvās un imperatīvās operācijas koda pilnvarošanai. CAS atļaujas pieprasījumi tiek izmantoti nepieciešamo atļauju noteikšanai asamblejai, kas izsauc citu asambleju, metodi vai klasi, pirms .NET *Framework* izpildes vide palaiž to.

Iepriekš bija minēts, ka pastāv trīs CAS atļauju pieprasījumu deklarēšanas tipi. Deklaratīvo un imperatīvo aizsardzības operāciju īstenošanai izmanto šādus deklarāciju variantus:

- `Assert` – nosaka izpildes videi ignorēt faktu, ka kodam var arī nebūt norādīto atļauju;
- `Demand` – nosaka izpildes videi ģenerēt izņēmuma gadījumu, ja izsaucošajam kodam un kodam, kas atrodas augstāk stekā, nav norādīto atļauju;
- `Deny` – izraisa to, ka izpildes vide ierobežo metodes piekļuvi, izdzēšot norādītās atļaujas;
- `InheritanceDemand` – nosaka izpildes videi ģenerēt izņēmuma gadījumu, ja asamblejai, kas manto no klases, nav norādīto atļauju;
- `LinkDemand` – izraisa to, ka izpildes vide ģenerē izņēmuma gadījumu, ja kodam, kas tieši izsauc asambleju, nav norādīto atļauju;
- `PermitOnly` – nosaka izpildes videi ierobežot metodes piekļuvi, izdzēšot visas atļaujas, izņemot norādītās.

Lai varētu izprast šo metožu darbības principu, tiek piedāvāts piemērs par grupu no četriem cilvēkiem, kas grib iekļūt slēgtā pasākumā. Saimnieks (metode) ir nolīdzis apsargu (.NET *Framework* izpildes vidi), kas raugās, lai ieiet (izsaukt metodi) varētu tikai tie viesi (izsaucošās asamblejas), kuriem ir ielūgumi (CAS atļaujas).

Ja saimnieks izsauc metodi `InvitedGuests.LinkDemand()`, tad apsargs pārbauda pirmā viesa ielūgumu un tad ielaiž visus pārējos. Tas ir ātri, bet uz pasākumu var tikt cilvēki bez ielūguma. Ja izsaukta metode `InvitedGuests.Demand()`, tad apsargs atsevišķi pārbauda katra viesa ielūgumus. Tas aizņems vairāk laika, bet svešie netiks.

Lai paātrinātu procedūru, pirmais no uzaicinātajiem viesiem var izmantot `InvitedGuests.Assert()`, lai apsargs būtu pārliecināts, ka visi viesi no grupas bija uzaicināti (tiek pieņemts, ka apsargs var paļauties uz pirmā viesa ielūgumu). Turklāt pirmais viesis var atvest cilvēkus bez ielūgumiem. Tas var būt labi gadījumā, ja saimnieks vēlas daudz viesu pasākumā, bet negrib izdāļāt daudz ielūgumu (kas var nokļūt sliktās rokās). No otras puses – būs slikti, ja kāds ļaundaris uzzinās par iespēju iekļūt pasākumā.

Ja saimnieks vēlas, lai viesi pasākumā tikai dejotu un nedarītu neko citu, viņam jāizmanto `Dancing.PermitOnly()`, lai apsargs kontrolētu viesu atrašanos deju laukumā. Ja saimnieks vēlas, lai viesi darītu jebko, tikai ne dejotu, var izmantot `Dancing.Deny()`.

Izstrādātājam ir daudz CAS realizācijas variantu lietotnēs un šīs instrukcijas var palīdzēt konkrētās metodes izvēlē:

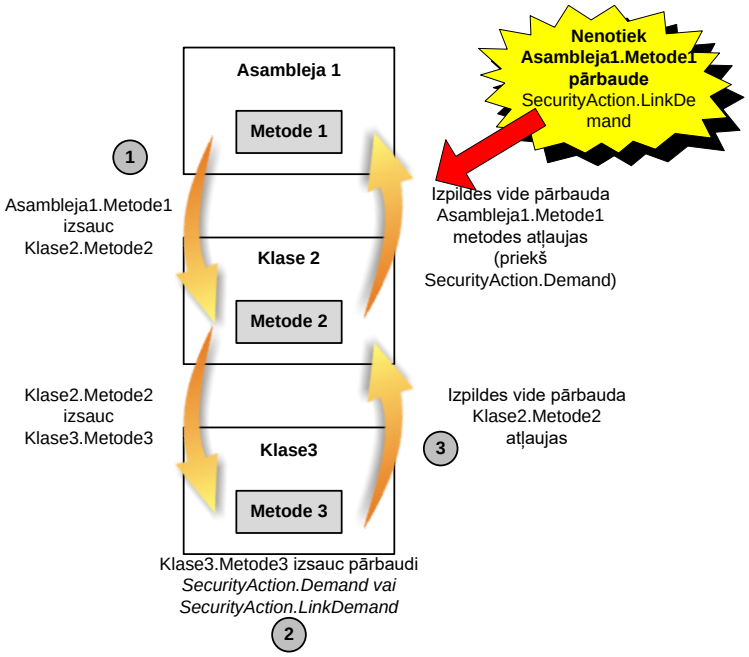
- lietojiet deklarāciju `SecurityAction.PermitOnly`, lai ierobežotu katrai metodei pieejamās atļaujas. Uzrādiet visas atļaujas, kas nepieciešamas metodei;
- lietojiet deklarāciju `SecurityAction.Deny`, lai detalizēti aprakstītu katrai metodei pieejamās atļaujas;
- lietojiet deklarāciju `CodeAccessPermission.PermitOnly`, lai imperatīvi ierobežotu atļaujas, ja kādai metodes daļai nepieciešams mazāk atļauju nekā pārējai daļai. Tas ir svarīgi, kad izsaucošos objektus ir izveidojuši citi izstrādātāji. Atļauju atjaunošanai izmantojiet metodi `CodeAccessPermission.RevertPermitOnly`;
- lietojiet `CodeAccessPermission.Assert`, ja jāatļauj kodam ar daļēju uzticību izsaukt metodi, kam nepieciešamas atļaujas, kādu nevar būt izsaucošajam kodam. Rūpīgi pārbaudiet kodu, piemēram, parametru `Assert` var izmantot ļaunprātīgi lietotāji, lai iegūtu paaugstinātas privilēģijas. Pēc tādu funkciju izpildes, kuras prasa paaugstinātas privilēģijas, atjaunojiet sākotnējās atļaujas ar metodi `CodeAccessPermission.RevertAssert`;
- lietojiet `CodeAccessPermission.Demand` tikai tajos gadījumos, ja asambleja realizē īpašas funkcijas, kas nav .NET *Framework* funkciju pamatā, piemēram, nepārvaldīta koda izsaukšana.

 Pastāv viedoklis, ka deklaratīvie priekšraksti ir mazāk droši nekā imperatīvie, jo deklaratīvos vieglāk analizēt. Tāpat jāņem vērā tas, ka deklaratīvie priekšraksti izpildās ātrāk par imperatīvajiem.

Divas `SecurityAction` deklarācijas un divas `CodeAccessPermission` metodes ļauj izpildes videi ģenerēt izņēmuma gadījumus, ja iztrūkst CAS atļaujas `Demand` un `LinkDemand`. Atšķirība starp deklarēšanu un metodēm ir tajā apstāklī, ka `Demand` veic atļauju pārbaudi, lai apstiprinātu visa izsaucošā koda piekļuvi, kamēr `LinkDemand` pārbauda tikai tiešo izsaucošo kodu. Lai izprastu atšķirību, var salīdzināt procedūras `Demand` un `LinkDemand` (sk. 6.7. att.) izmantošanu.

Kā redzams attēlā, `Demand` nosaka, vai visam izsaucošajam kodam ir pieprasītās atļaujas un ģenerē izņēmuma gadījumu, ja tādu nav. Tas ir daudz drošāk nekā `LinkDemand` izmantošana, kura pārbauda tikai tiešo izsaucošo kodu. Kā jau jebkuram aizsardzības mehānismam, arī šeit ir savi trūkumi. `Demand` pieprasa, lai izpildes vide veiktu vairāk pārbaužu, kas prasa lielākus procesora resursus un attiecīgi samazina veiktspēju. `LinkDemand` gadījumā veiktspēja ir augstāka, bet palielinās risks, ka ļaunprātīgs lietotājs var apiet aizsardzības pārbaudi.

 `Demand` un `LinkDemand` nepārbauda pašreizējās metodes atļaujas – tiek pārbaudīts izsaucošais kods, bet ne asamblejas atļaujas.



6.7. att. **`Demand` un `LinkDemand` izmantošanas procedūra**

Ja nepieciešams noteikt, vai asamblejai ir nepieciešamās atļaujas, nav ieteicams izmantot `Demand`, jo, kā jau tika teikts, tā nepārbauda asamblejas atļaujas. Tā vietā jāizmanto metode `System.Security.SecurityManager.IsGranted`, kā parādīts 6.2. piemērā.

6.2. piemērs. Metodes `IsGranted` izmantošana

```
using System;
using System.IO;
using System.Security;
using System.Security.Permissions;
namespace isgranted
{
   class Program
   {
      static void Main(string[] args)
      {
         FileIOPermission filePermissions = new
                              FileIOPermission(FileIOPermissionAccess.Read,@"c:\windows\");
         if (SecurityManager.IsGranted(filePermissions) == true)
         {  // Asambleja var nolasīt datus no mapes c:\windows
            StreamReader r1 = new StreamReader(@"c:\windows\piemers.txt");
            Console.WriteLine("Pirmaa rinda no piemers.txt: " + r1.ReadLine());
            Console.ReadLine();
         }
         else
         {  // Asambleja nevar nolasīt datus no mapes c:\windows
            Console.WriteLine("Nevar nolasīt info no mapes windows");
            Console.ReadLine();
         }
      }
   }
}
```

Rezultāts

```
C:\projects\ConsoleApp6\ConsoleApp6\bin\Debug\net8.0\ConsoleApp6.exe
Nevar nolasit info no mapes windows
```

Lai varētu veikt CAS atļauju pieprasījumu ***deklaratīvās operācijas***, jāizmanto kāda no 6.5. tabulā dotajām klasēm kopā ar `SecurityAction.Demand` un `SecurityAction.LinkDemand`. 6.3. piemērā parādīta metode, kas izmanto klasi `FileIOPermissionAttribute`, lai deklaratīvi pārbaudītu, vai kodam, kas izsauc doto metodi, ir piekļuve mapei *C:\A*.

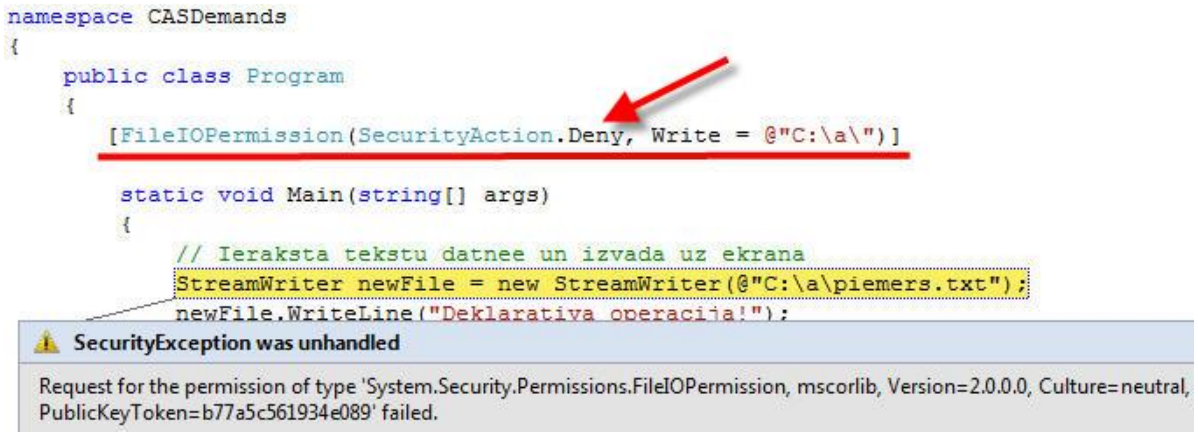

6.3. piemērs. Deklaratīvo operāciju piemērs

```
using System;
using System.IO;
using System.Security;
using System.Security.Permissions;
namespace CASDemands
{
   public class Program
   {
  [FileIOPermission(SecurityAction.Demand, Write = @"C:\a\")]
   static void Main(string[] args)
      {
         // Ieraksta tekstu datnē un izvada ekrānā
         StreamWriter newFile = new StreamWriter(@"C:\a\piemers.txt");
         newFile.WriteLine("Deklaratīvā operacija!");
         newFile.Close();
         StreamReader r1 = new StreamReader(@"c:\a\piemers.txt");
         Console.WriteLine("Pirmā rinda ir: " + r1.ReadLine());
         Console.ReadLine();
      }
   }
}
```

Rezultāts

```
C:\projects\ConsoleApp6\ConsoleApp6\bin\Debug\net8.0\ConsoleApp6.exe
Pirmā rinda ir: Deklaratīvā operacija!
```

Ja `Demand` aizvieto ar `Deny`, tad ieraksts mapes *C:\A* datnē ir ierobežots, kā rezultātā vajadzētu tikt ģenerētam izņēmuma gadījumam, kā parādīts 6.8. attēlā.



6.8. att. **Izņēmuma gadījums, kad kodam nav uzrādītās atļaujas**

Katrai `SecurityAction` deklarācijai, ko izmanto CAS atļauju pieprasījumu veikšanai, pastāv metode `CodeAccessPermission` ar tādu pašu nosaukumu un funkcionalitāti, ko izmanto ***imperatīvo*** operāciju veikšanai.

`SecurityAction` pielieto deklaratīvo operāciju veikšanai, bet metodi `CodeAccessPermission` – imperatīvo operāciju veikšanai.

6.4. piemērā tiek veiktas tās pašas darbības, kas 6.3. piemērā, tikai ar imperatīvajām atļauju pārbaudes operācijām.

6.4. piemērs. Imperatīvo operāciju piemērs

```
using System;
using System.IO;
using System.Security;
using System.Security.Permissions;

namespace CASImperativs
{
  public class CASImperativeClass
  {
    [FileIOPermission(SecurityAction.Assert, Write = @"C:\a\")]

    static void Imperativs()
    {
  FileIOPermission filePermissions = new FileIOPermission(FileIOPermissionAccess.Write,
                                                          @"C:\a\");
  filePermissions.Assert();
        try
        {
          StreamWriter newFile = new StreamWriter(@"C:\a\piemers.txt");
          newFile.WriteLine("Imperatīvā operācija!");
          newFile.Close();
          StreamReader r1 = new StreamReader(@"c:\a\piemers.txt");
          Console.WriteLine("Pirmā rinda ir: " + r1.ReadLine());
        }
        finally
        {
          CodeAccessPermission.RevertAssert();
        }
    }
    static void Main(string[] args)
    {
        Imperativs();
        Console.WriteLine("beigas");
        Console.ReadLine();
    }
  }
}
```

Rezultāts

```
file:///C:/projects/CASImperativs/CASImperativs/bin/Debug/CASImperativs.EXE
Pirmaa rinda ir: Imperativa operacija!
beigas
```

Līdzīgi kā iepriekšējos piemēros, ja `Assert` aizvieto ar `Deny`, tad ieraksts mapes *C:\A* datnē ir ierobežots, kā rezultātā vajadzētu tikt ģenerētam izņēmuma gadījumam, kā parādīts 6.9. attēlā.

Imperatīvo aizsardzības operāciju priekšrocība ir tajā apstāklī, ka var pārtvert un apstrādāt aizsardzības izņēmuma gadījumus, kas parādās metodē. Ja vajag tikai ģenerēt izņēmumu izsaucošajā kodā, tad jālieto deklaratīvās operācijas.

```
static void Imperativs()
{
    FileIOPermission filePermissions = new FileIOPermission(FileIOPermissionAc
    filePermissions.Deny();

    try
    {
        StreamWriter newFile = new StreamWriter(@"C:\a\piemers.txt");
        newFile.WriteLine("Imperativa operacija!");
        newFile.Close();
        StreamReader r1 = new StreamReader(@"c:\a\piemers.txt");
        Console.WriteLine("Pirmaa rinda ir: " + r1.ReadLine());
```

SecurityException was unhandled

Request for the permission of type 'System.Security.Permissions.FileIOPermission, mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089' failed.

6.9. att. **Izņēmuma gadījums, kad kodam nav uzrādītās atļaujas**

CAS atļaujas pieprasījumus jācenšas pielietot tā, lai ierobežotu asamblejas atļaujas līdz minimumam, kas nepieciešams asamblejas funkcionēšanai, t. i., atļaujas var pielāgot rūpīgāk, ierobežojot atsevišķu metožu atļaujas.

`SecurityAction` un atļauju metodes nosaka izpildes videi ierobežot šādas CAS atļaujas: `Deny` un `PermitOnly`. Atšķirība ir tajā, ka `Deny` izdzēš vienu atļauju vai atļauju grupu, bet `PermitOnly` izdzēš visas atļaujas vai atļauju grupas, izņemot norādītās. `Deny` veic funkcijas, kas analoģiskas `RequestRefuse`, bet `PermitOnly` – analoģisks `RequestOptional`.

`RequestRefuse` un `RequestOptional` izmanto asambleju deklarēšanai, bet `Deny` un `PermitOnly` – metožu uzdošanai.

Deklaratīvās metodes atļaujas ierobežojuma piemērs, kā aizliegt metodei piekļuvi mapei *C:\Windows*: **[FileIOPermission(SecurityAction.Deny, All = @"C:\windows\")]**

Imperatīvās metodes atļaujas ierobežojuma piemērs, kā aizliegt metodei piekļuvi mapei *C:\Windows*:

```
FileIOPermission filePermissions = new FileIOPermission(FileIOPermissionAccess.AllAccess,
@"C:\windows\");
 filePermissions.Deny();
```

CAS atļaujas pieprasījumi pastiprina aizsardzību, taču var samazināt lietotnes veiktspēju. Piemēram, metodes `Demand` izsaukums prasa diezgan daudz resursu, jo liek izpildes videi sistemātiski pārbaudīt katra izsaucošā objekta atļaujas. Metode `LinkDemand` ir viens no veidiem paaugstināt veiktspēju, salīdzinot ar metodi `Demand`, taču ar risku samazināt drošību. Cits paņēmiens ir metodes `Assert` izmantošana, kurš nosaka izpildes videi izlaist visas aizsardzības pārbaudes.

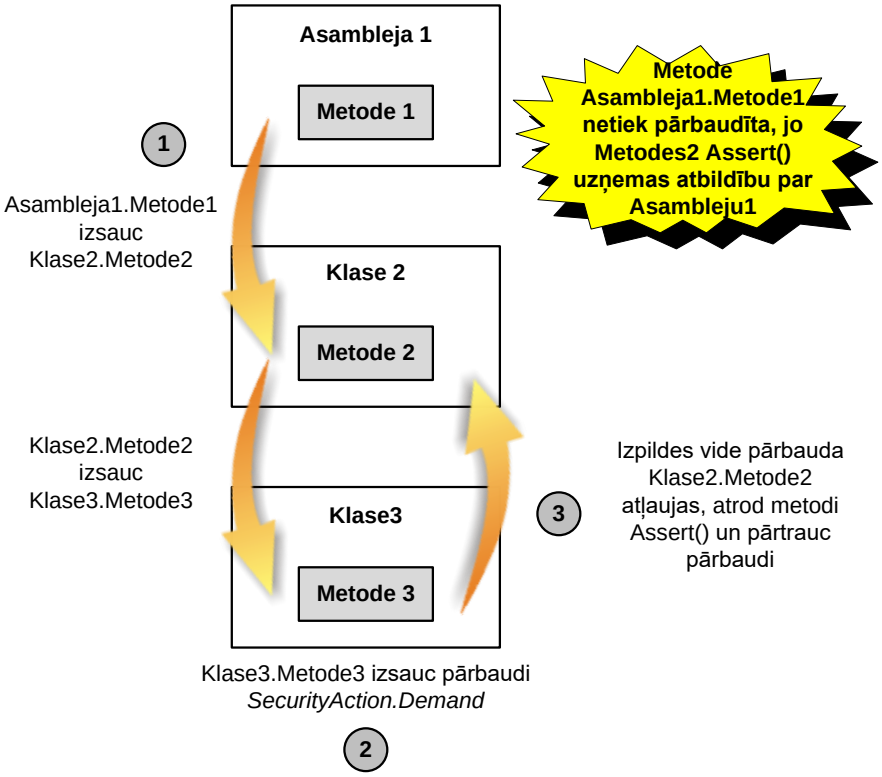
Metodei `CodeAccessPermission.Assert` nav nekā kopīga ar funkciju `assert` programmēšanas valodā C++.

Atļauju objekti satur metodi `Assert`, kas ļauj metodei galvot par visiem izsaucošajiem objektiem. 6.10. attēlā parādīts, kā metodes `Assert` izsaukums aizliedz izpildes videi pārbaudīt CAS atļaujas asamblejām, kas atrodas augstāk stekā.

Tam ir divas sekas:

- veiktspējas palielināšanās, jo samazinās atļauju pārbaužu skaits;
- atļauja nepriviliģētam kodam izsaukt metodes ar augstākām CAS atļauju prasībām.

Piemēram, ja tiek izveidots objekts `RegistryPermission` un izsaukta metode `Assert`, tad asamblejai jābūt piešķirtai atļaujai `RegistryPermission`, taču kodam, kas izsauc šo asambleju, tādas atļaujas var nebūt.



6.10. att. **`Assert` bloķē pārbaudes**

Metodi `Assert` var izmantot gan deklaratīvi, gan arī imperatīvi. Deklaratīvās operācijas piemērs ir šāds: **`[FileIOPermission(SecurityAction.Assert, All = @"C:\windows\")].`** Līdzīgi atļaujas uzdod imperatīvi šādā veidā (tiek bloķētas visas CAS atļauju pārbaudes, vēršoties pie datnēm mapē *C:\windows*):

```
FileIOPermission filePermissions = new FileIOPermission(FileIOPermissionAccess.AllAccess,
@"C:\windows\");
 filePermissions.Assert();
```

6.4. Koda piekļuves aizsardzības darbību pielietošana

Izstrādātājiem ieteicams izmanto CAS priekšrocības lietotņu aizsardzības nodrošināšanā. Šajā apakšnodaļā tiks doti praktiski ieteikumi un rekomendācijas savu aizsardzības metožu izstrādāšanai, lai izstrādātājam būtu priekšstats, kā sākt veidot savas atļaujas vai noteikt dalību grupās, kodu grupās un lietotņu domēnos.

Aizsardzības ieviešana ar lietotņu domēnu palīdzību jau tika apskatīta 4.2. apakšnodaļā. No šīs nodaļas ir zināms, ka lietotņu domēns ir ceturtais drošības politikas līmenis. Tā kā tas ir atsevišķs politikas līmenis, tad, protams, lietotņu domēnam var pielāgot tos pašus aizsardzības elementus, kādus izmanto uzņēmuma, datora vai lietotāja līmenī.

Lietotņu domēna izveidošana un asamblejas izpilde tajā tiek realizēta pēc šādas vispārīgas shēmas, kas nodrošina asamblejas izolāciju:

```
AppDomain myDomain = AppDomain.CreateDomain(„MyDomain");
myDomain.ExecuteAssembly(@"C:\SecondAssembly.exe");
```

Īpašnieka apliecinājuma sniegšana asamblejai

Izveidojot lietotņu domēnu un palaižot tajā asamblejas, izstrādātājs iegūst pilnu kontroli pār saimnieka drošības apliecinājumiem. Tā rezultātā izstrādātājs var kontrolēt visas atļaujas, kas tiks piešķirtas asamblejai. Lai asamblejai tiktu sniegts apliecinājums, no sākuma jāizveido objekts `System.Security.Policy.Evidence` un tad jānodod to kā parametru lietotņu domēnu metodei `Execute.Assembly`. Šī objekta izveides laikā tam jānodod divi masīvi. Viens masīvs satur īpašnieka apliecinājumus un otrs - asamblejas apliecinājumus. Katrs no masīviem var pieņemt `null` vērtību, un, ja speciāli netiek veidoti asamblejas apliecinājumi, tad visbiežāk tikai jānorāda īpašnieka apliecinājuma īpašības.

Vienkāršākais veids atļauju uzdošanai asamblejai lietotņu domēnā ir apliecinājuma `Zone` nodošana ar `System.Security.Policy.Zone` un `System.Security.SecurityZone` palīdzību. Koda fragments 6.5. piemērā demonstrē konstruktora `Evidence` izmantošanu, kurš izveido objektu `Zone`, pievieno to objektam `hostEvidence` un izveido `Evidence` objektu ar nosaukumu `internetEvidence`. Šis objekts `Evidence` tiek nodots lietotņu domēna metodei `ExecuteAssembly` kopā ar asamblejas datnes nosaukumu.

6.5. piemērs. Īpašnieka apliecinājuma sniegšana asamblejai

```
using System.Security;
using System.Security.Policy;
...
object [] hostEvidence = {new Zone(SecurityZone.Internet)};
Evidence internetEvidence = new Evidence(hostEvidence, null);

AppDomain myDomain = AppDomain.CreateDomain("MyDomain");
myDomain.ExecuteAssembly(@"C:\SecondAssembly.exe", internetEvidence);
...
```

Rezultātā šī asambleja tiks palaista izolētā lietotņu domēnā ar vienu vienīgu atļauju kopu, ko piešķir kodu grupai *Internet_Zone*. Kad lietotņu domēns palaiž asambleju, izpildes vide analizē uzrādīto apliecinājumu. Tā kā apliecinājums atbilst *Internet* zonai, tad izpildes vide piešķir to kodu grupai *Internet_Zone*, kas savukārt izsauc *Internet* atļauju piešķiršanu.

Īpašnieka apliecinājuma sniegšana lietotņu domēnam

Apliecinājumu var sniegt arī pašam lietotņu domēnam. Šis paņēmiens analoģisks apliecinājuma sniegšanai jaunai asamblejai un izmanto metodes `AppDomain.CreateDomain` pārlādēšanu, ko pieņem objekts `Evidence`. Koda fragments parādīts 6.6. piemērā.

6.6. piemērs. Īpašnieka apliecinājuma sniegšana lietotņu domēnam

```
using System.Security;
using System.Security.Policy;
...
Zone safeZone = new Zone(SecurityZone.Internet);
object [] hostEvidence = {new Zone(SecurityZone.Internet)};
Evidence appDomainEvidence = new Evidence(hostEvidence, null);

AppDomain myDomain = AppDomain.CreateDomain("MyDomain", appDomainEvidence);
myDomain.ExecuteAssembly(@"C:\SecondAssembly.exe");
...
```

Dalības nosacījumu klases

Izveidotajos lietotņu domēnos var veidot savas koda grupas. Taču pirms koda grupas izveides nepieciešams noteikt dalības nosacījumus un atļauju kopas. .NET *Framework* satur astoņas dalības

nosacījumu klases, kuras ļauj norādīt dalības nosacījumu visam kodam vai kādam no septiņiem drošības apliecinājuma tipiem no 6.1. tabulas:

- `AllMembershipCondition;`
- `ApplicationDirectoryMembershipCondition;`
- `HashMembershipCondition;`
- `PublisherMembershipCondition;`
- `SiteMembershipCondition;`
- `StrongNameMembershipCondition;`
- `UrlMembershipCondition;`
- `ZoneMembershipCondition.`

Dalības nosacījuma objekts jānosaka katru reizi, kad tiek veidota grupas koda klase. Piemēram, šāda koda rinda veido objektu `UrlMembershipCondition` ar URL adresi http://www.microsoft.com/assembly.exe:

UrlMembershipCondition myCondition = new UrlMembershipCondition („http://www.microsoft.com/assembly.exe");

Klase `AllMembershipCondition` neprasa dalības nosacījumu, jo attiecas uz visu kodu. Standarta kodu grupa `All_Code` arī izmanto klasi `AllMembershipCondition`.

Kodu grupu klases

.NET *Framework* satur četras kodu grupu klases, kas iegūstamas no `System.Security.Policy.CodeGroup`:

1) `System.Security.Policy.FileCodeGroup` - koda grupa ar dalības nosacījumu (norāda izstrādātājs) un atļauju kopu (ietver tikai atļauju `FileIOPermission`) piekļuvei mapei, no kuras tika palaists kods;
2) `System.Security.Policy.FirstMatchCodeGroup` - koda grupa, kam jāsatur vairākas unikālas pakļautas koda grupas. Asamblejām tiek piešķirtas pirmās kodu grupas atļauju kopa ar attiecīgajiem dalības nosacījumiem, kas apvienoti ar primārajam objektam `FirstMatchCodeGroup` piešķirtajām tiesībām;
3) `System.Security.Policy.NetCodeGroup` - koda grupa ar dalības nosacījumu (norāda izstrādātājs) un atļauju kopu (ietver tikai atļauju `WebPermission`) piekļuvei vietnei, no kuras tika palaists kods;
4) `System.Security.Policy.UnionCodeGroup` - visbiežāk izmantojamā koda grupa. Asamblejām piešķir visu pakļauto kodu grupu atļauju kopas ar attiecīgajiem dalības nosacījumiem, kas apvienoti ar primārajam objektam `UnionCodeGroup` piešķirtajām tiesībām.

Ja nepieciešams izveidot kodu grupu, tiek ieteikts izmantot `UnionCodeGroup`. Konstruktoram jānodod dalības nosacījuma objekts un objekts `PolicyStatement` (sastāv no atļauju kopas un atribūtu kopas). Jaunajiem `CodeGroup` objektiem jānodod objekti `PolicyStatement`, nevis `PermissionSet`.

Lietotņu domēnu izveidošana ar savām drošības politikām

Lai izveidotu lietotņu domēnu ar savu drošības politiku, ir jāuzdod dalības nosacījums, koda grupa, politikas nosacījumi, atļauju kopa un politikas līmenis. 6.7. piemērā tiek demonstrēts, kā izveidot kodu grupu, izmantojot URL adresei *http://www.microsoft.com/assembly.exe* atbilstošu dalības nosacījumu un *Internet* atļauju kopu.

6.7. piemērs. Koda grupas izveidošana

```
UrlMembershipCondition safeMembership = new
                UrlMembershipCondition("http://www.microsoft.com/assembly.exe");
```

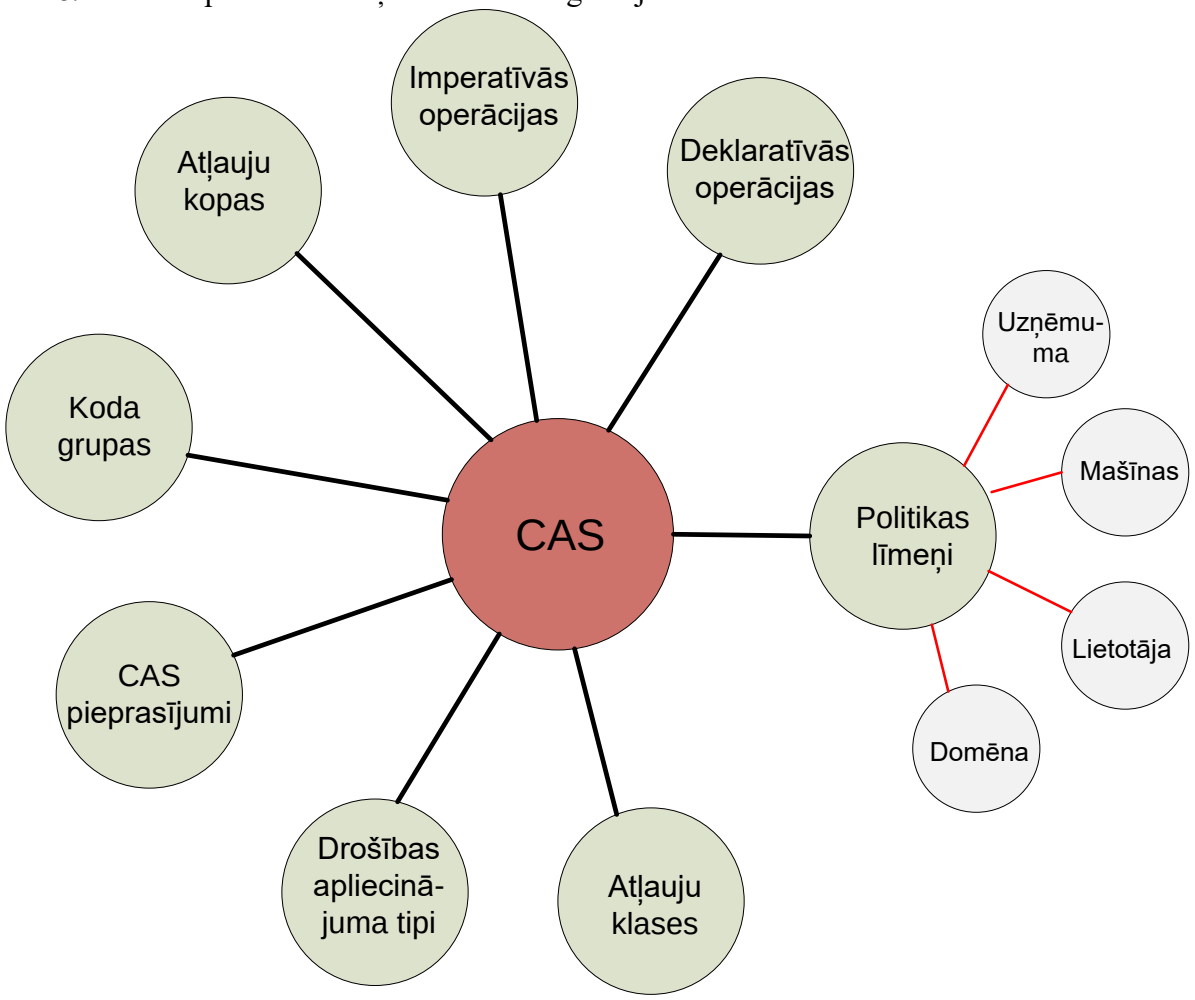
```
PermissionSet safePermissionSet =
                    PolicyLevel.CreateAppDomainLevel().GetNamedPermissionSet("Internet");

PolicyStatement safePolicyStatement = new PolicyStatement(safePermissionSet);

CodeGroup safeCodeGroup = new UnionCodegroup(safeMembership, safePolicyStatement);
```

Pēc savas koda grupas definēšanas nepieciešams izveidot objektu `PolicyLevel` un pievienot šo kodu grupu politikas līmenī. Objekts `PolicyLevel` vienmēr izmanto saknes kodu grupu, ko nosaka īpašība `PolicyLevel.RootCodeGroup`.

Tādējādi iepriekšējo piemēru var papildināt, nosakot politikas līmeni, izveidojot lietotņu domēnu un tam norādot savu politikas līmeni. To var izdarīt ar šāda koda fragmenta palīdzību:

```
PolicyLevel safePolicyLevel = PolicyLevel.CreateAppDomainLevel();
safePolicyLevel.RootCodeGroup = safeCodeGroup;
AppDomain myDomain = AppDomain.CreateDomain("MyDomain");
myDomain.SetAppDomainPolicy(safePolicyLevel);
```

6.11. attēlā parādīts nodaļā minēto svarīgāko jēdzienu koks.



6.11. att. **Sestajā nodaļā minēto svarīgāko jēdzienu koks**

Pārbaudes testa jautājumi

1. .NET asambleja izmanto reģistrus konfigurācijas datu uzglabāšanai. Tāpat šī asambleja nodrošina lietotāju drukas iespējas un notikumu ierakstīšanu notikumu žurnālā. Asambleja tiek palaista *Internet* tīklā. Kādas no funkcijām asambleja nespēs veikt?
    a. Konfigurācijas saglabāšana reģistros
    b. Dokumentu drukāšana
    c. Ierakstīšana notikumu žurnālā
    d. Visas minētās

2. Dalības nosacījums kodu grupai *Microsoft_Learning* ir lietotnes mape C:*Program Files\Microsoft Learning*. Atļauju kopa šai kodu grupai sastāv no šādām tiesībām:
    - nolasīt *Environment* mainīgo *USERNAME*;
    - ierakstīt reģistrā *HKEY_CURRENT_USER\Softwares*;
    - nolasīt datni *MyApp.ini* un ierakstīt datnē *MyApp.ini*.

    Par lietotnes mapes drošības apliecinājumu kalpo asamblejas izcelsme no lokālā datora cietā diska *C:\Program Files\Microsoft Learning*. Kādas atļaujas ir liegtas asamblejai?
    a. Nolasīt *Environment* mainīgo TEMP
    b. Drukāt dokumentu
    c. Nolasīt daļu no *MyApp.ini*
    d. Nekas no minētā

3. Jūs esat *Internet* tīkla pakalpojumu sniedzējs. Divas organizācijas vēlas izvietot savas tīmekļa aplikācijas uz Jūsu tīmekļa servera. Kādu drošības politikas līmeni Jūs veidosiet, lai izolētu šīs divas aplikācijas?
    a. Uzņēmuma līmeni
    b. Datora līmeni
    c. Lietotāja līmeni
    d. Lietotņu domēnu līmeni

4. Asamblejā ir divi definēti *FileIOPermissions* objekti. *FileIOPermissionA* ļauj lasīt datni *C:\Test\MyFile.txt* un *FileIOPermissionB* ļauj lasīt datni *C:\Test\MyIni.txt*. Kas no minētā ir nepieciešams šīs asamblejas izsaukšanai?
    a. Izsaucējam jābūt vienas vai otras datnes lasīšanas atļaujai
    b. Izsaucējam jābūt abu datņu lasīšanas atļaujai
    c. Izsaucējam jābūt vienas vai otras datnes ierakstīšanas atļaujai
    d. Nekas no minētā

5. Asamblejā tiek lietota metode, par kuras atļaujām nav zināms pirms palaišanas. Kāds CAS drošības tips būtu jāizmanto?
    a. Deklaratīvā aizsardzība
    b. Imperatīvā aizsardzība
    c. Abas no minētajām
    d. Neviena no minētajām

6. Nepieciešams identificēt zonu, no kuras palaistā asambleja ir ielādēta. Kāda no identitātes atļaujām būtu jālieto?
    a. *URLIdentityPermission*
    b. *SiteIdentityPermission*
    c. *StrongNameIdentityPermission*
    d. Nekas no minētā

7. Kriptogrāfija ar .NET

Informācijas sistēmās datu aizsardzības pamatā ir datu šifrēšana [18], *t.i., lai tiktu palielināta datu aizsardzības drošība, nepieciešams šifrēt datus. Konfidenciālo datu aizsardzībai jāizmanto kriptogrāfijas iespējas. Lielākā daļa mūsdienu šifrēšanas metodes dalās divās grupās: simetriskās atslēgas kriptogrāfija un publiskās jeb asimetriskās atslēgas kriptogrāfija. .NET Framework ir iekļautas klases šo grupu kriptogrāfisko funkciju izmantošanai.*

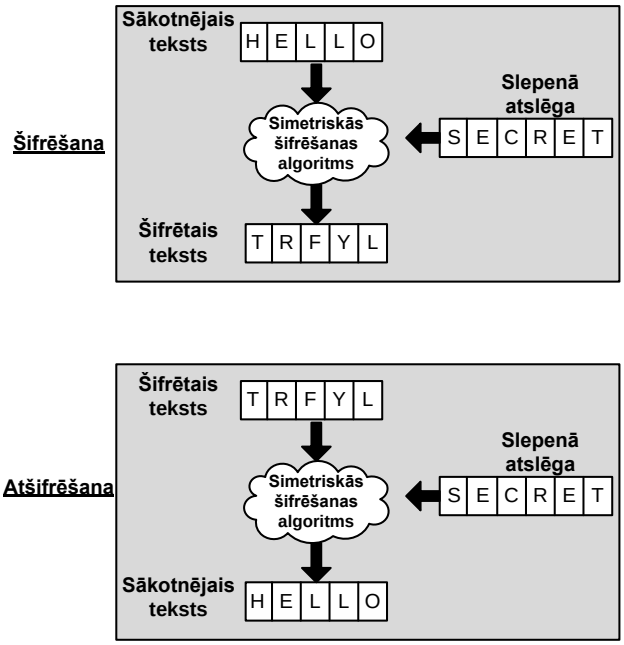
Nodaļā apskatīti datu aizsardzības realizācijas paņēmieni ar .NET kriptogrāfisko funkciju palīdzību. Tiek iztirzāta datu šifrēšana un atšifrēšana ar simetriskās un asimetriskās kriptogrāfijas palīdzību, kā arī ciparparaksta realizācijas iespējas.

7.1. Simetriskās kriptogrāfijas ieviešana

Šifrēšana (*encryption*) – datu un ziņojumu apstrādes process, ko veic datu sagatavotājs vai ziņojuma nosūtītājs, lai datus vai ziņojuma saturu nodrošinātu pret nesankcionētu izmantošanu. Lai šādus datus vai ziņojuma saturu varētu izmantot, jāveic tā atšifrēšana.

Atšifrēšana (*decryption*) – šifrēta ziņojuma apstrāde, ko veic tā saņēmējs, lai atklātu ziņojuma sākotnējo saturu.

Šifrēšana ar simetrisko atslēgu jeb šifrēšana ar slepeno atslēgu ir kriptogrāfisks paņēmiens, kurā izmanto vienu slepenu atslēgu gan šifrēšanai, gan atšifrēšanai. Šifrēšanas algoritms apstrādā datus ar slepenas šifrēšanas atslēgas palīdzību, lai iegūtu šifrētu tekstu. Šifrētu tekstu nav iespējams atšifrēt, ja nav zināma šī slepenā atslēga, un tāpēc šo algoritmu sauc par simetrisku (sk. 7.1. att.).

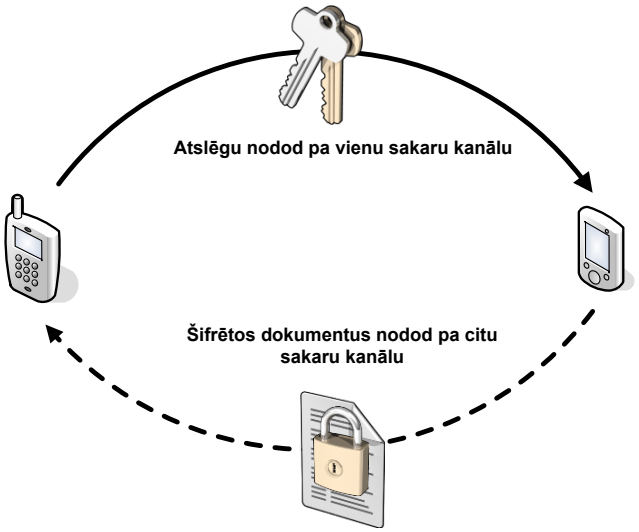


7.1. att. **Šifrēšanai un atšifrēšanai lieto vienu un to pašu atslēgu**

Simetriskā šifrēšana (*symmetric-key cryptography*) – šifrēšanas sistēma, kurā atšķirībā no publiskās atslēgšifrēšanas ziņojuma nosūtītājs un ziņojuma saņēmējs ziņojuma šifrēšanai un atšifrēšanai izmanto vienu un to pašu šifrēšanas atslēgu. Simetriskā atslēgšifrēšana ir vienkāršāka un ātrāka nekā publiskā atslēgšifrēšana, bet, lai izmantotu šo šifrēšanas sistēmu, jāatrod drošs veids, kā abām informācijas apmaiņā iesaistītajām pusēm apmainīties ar šifrēšanas atslēgām. Publiskajā atslēgšifrēšanā privātā šifrēšanas atslēga nekad netiek pārsūtīta.

Simetriskie algoritmi strādā ļoti ātri un labi noder lielu apjomu datu šifrēšanai. Tomēr, neskatoties uz to, ka simetriskā šifrēšana nodrošina noteiktu aizsardzības līmeni, „pietiekami ilgā laika periodā" ļaunprātīgs lietotājs varētu atšifrēt sākotnējo tekstu, secīgi pārlasot visas iespējamās šifrēšanas atslēgu kombinācijas.

Simetriskās šifrēšanas galvenais trūkums ir pieņēmums, ka abas puses (šifrēšanā un atšifrēšanā) atrunājušas izmantojamo slepeno atslēgu. Tāpēc nedrīkst pārāk paļauties uz slepeno atslēgu, jo pati tā netiek šifrēta. Tāpēc lielu nozīmi iegūst veids, kā nogādāt slepeno atslēgu abām pusēm pa drošiem sakaru kanāliem: telefoniski, personiski, ar vēstuli vai kā citādi (sk. 7.2. att.).



7.2. att. **Atslēgu un šifrēto dokumentu nodošanai izmanto dažādus sakaru kanālus**

Vairums .NET *Framework* kriptogrāfisko funkciju iebūvētas vārdu telpā `System.Security.Cryptography`, ieskaitot četrus simetriskās šifrēšanas algoritmu realizācijas paņēmienus (sk. 7.1. tabulu).

7.1. tabula

Klases darbam ar simetriskajiem algoritmiem

Klase	Atslēgas garums	Apraksts
`SymmetricAlgorythm`	Nenoteikts	Bāzes klase, no kuras iegūst visus simetriskos algoritmus
`RijndaelManaged`	128-256 biti ar 32 bitu pieaugumu (128, 192 vai 256)	Simetriskās šifrēšanas algoritma `Rijndael` realizācija (tiek uzskatīts par visdrošāko)
`DES`	56 biti	Simetriskās šifrēšanas algoritma DES (*Data Encryption Standard*) realizācija. Izmanto relatīvi īsas atslēgas, tāpēc nav ieteicams to pielietot. Algoritmu izmanto tā savietojamības ar vecākām platformām nodrošināšanai
`RC2`	Mainīgs (no 40 bitiem līdz 1024 bitiem ar 8 bitu pieaugumu)	Izstrādāts DES nomaiņai
`TripleDES`	128-192 biti ar 64 bitu pieaugumu	Pēc būtības algoritms DES tiek pielietots 3 reizes.

Visām simetrisko algoritmu klasēm piemīt kopīgas īpašības:

- `BlockSize` – kriptogrāfiskās operācijas bloka izmērs bitos, t. i., bitu skaits, ko vienlaicīgi apstrādā algoritms (šo īpašību var ignorēt);

- `FeedbackSize` – kriptogrāfiskās operācijas atgriezeniskās saites izmērs bitos (šo īpašību var ignorēt);
- `IV` – nosaka inicializācijas vektoru (*Initialization Vector*). Šifrēšanai un atšifrēšanai jānorāda vienādas vērtības;
- `Key` – slepenā atslēga. Ja nav norādīta, tā tiks ģenerēta automātiski. Pēc šifrēšanas atslēgas vērtība jāsaglabā un jānodod atšifrēšanas modulim;
- `KeySize` – slepenās atslēgas izmērs bitos. Var ignorēt, jo, izveidojot simetriskā algoritma objektu, izpildes vide izvēlēsies maksimālā garuma atslēgu;
- `LegalBlockSizes` – masīvs no `KeySize`, kur uzglabājas simetriskā algoritma bloku izmēri. Katram masīva elementam piemīt īpašības `MinSize` un `MaxSize`, kas nosaka bloku pieļaujamos diapazonus bitos, un īpašība `SkipSize`, kas norāda intervālu bitos starp pieļaujamajiem bloku izmēriem;
- `LegalKeySizes` – masīvs no `KeySize`, kur uzglabājas simetriskā algoritma atslēgu izmēri. Katram masīva elementam piemīt īpašības `MinSize` un `MaxSize`, kas nosaka atslēgu pieļaujamos diapazonus bitos, un īpašība `SkipSize`, kas norāda intervālu bitos starp pieļaujamajiem atslēgu izmēriem;
- `Mode` – viena no pārskaitāmā parametra `CipherMode` vērtībām, kas nosaka šifrēšanas algoritma režīmu. Pēc noklusējuma iestatīts režīms CBC (*Cipher Block Chaining*);
- `Padding` – pārskaitāmā parametra `PaddingMode` vērtība, kas nosaka to, kādā veidā šifrēšanas algoritms apstrādā atšķirības starp algoritma bloka izmēru un sākotnējā teksta garumu (parasti to nemaina).

 Lai izmantotu šifrēšanu, nav īpašas nepieciešamības zināt par inicializācijas vektoriem. Pietiek saprast to, ka visi .NET *Framework* simetriskās šifrēšanas algoritmi datus šifrē pa blokiem – otrais bloks šifrējas, izmantojot pirmā bloka šifrēšanas rezultātus, trešais – izmantojot otrā bloka šifrēšanas rezultātus utt. Tā kā pirmajam blokam nav iepriekšējā, tad šifrēšanas algoritms izmanto inicializācijas vektoru, t. i., inicializācijas vektors ir dati, ko algoritms pievieno pirmajam šifrējamo datu blokam.

Simetrisko algoritmu klasēm ir šādas kopīgas metodes:

- `CreateEncryptor` – izveido simetriskās šifrēšanas objektu, ko turpmāk izmanto objekti `CryptoStream` datu plūsmas šifrēšanai;
- `GenerateIV` – ģenerē inicializācijas vektoru. Parasti šo metodi izsaukt nevajag, jo inicializācijas vektori tiek ģenerēti automātiski. Lieto, ja izstrādātājs ir noteicis un grib izmantot savu inicializācijas vektoru;
- `GenerateKey` – ģenerē atslēgu. Lieto, ja izstrādātājs ir noteicis un grib izmantot savu atslēgu;
- `ValidKeySize` – nosaka, vai ir pieļaujams norādītais atslēgas izmērs un atgriež `Boolean` vērtību. Lieto, ja izmanto nezināmu simetrisko algoritmu klasi, lai pārbaudītu atslēgas derīgumu dotajam algoritmam.

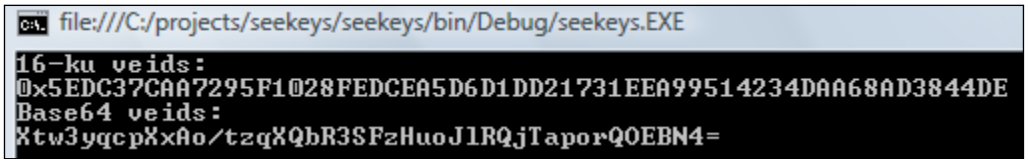
 No visiem šifrēšanas algoritmiem ar simetrisko atslēgu par visdrošāko tiek uzskatīts algoritms `Rijndael`, jo tas ir vienīgais algoritms, kuru tieši uztur .NET *Framework*.

Jau tika minēts, ka šifrēšanai un atšifrēšanai nepieciešama viena un tā pati atslēga. Vizuāli attēlot atslēgas ir diezgan sarežģīti, jo tās ir bitu virknes. Attēlot atslēgas var divējādi: heksadecimālo skaitļu veidā vai izmantojot `Base64` kodēšanu. 7.1. piemērā dots koda fragments, kas ļauj attēlot algoritma `Rijndael` ģenerēto atslēgu abos iepriekš minētajos veidos. `Base64` kodēšanā katra simbola attēlošanai izmanto 6 bitus (heksadecimālajā – 4 bitus). Tas ļauj attēlot atslēgu īsākā veidā.

7.1. piemērs. Atslēgu pārskats

```
using System;
using System.Text;
using System.IO;
using System.Security.Cryptography;

namespace seekeys
{
   class Program
   {
      static void Main(string[] args)
      {
         SymmetricAlgorithm myAlg = new RijndaelManaged();
         Console.WriteLine("16-ku veids:");
         Console.Write("0x");
         foreach (byte thisByte in myAlg.Key)
            Console.Write(thisByte.ToString("X"));
            Console.WriteLine();
         Console.WriteLine("Base64 veids:");
         Console.Write(Convert.ToBase64String(myAlg.Key));
         Console.ReadLine();
      }
   }
}
```

Rezultāts



Lai arī .NET *Framework* automātiski ģenerē gadījuma atslēgas, izstrādātajam var būt nepieciešams izveidot savu atslēgu. Lai ģenerētu patvaļīgu atslēgu, nepieciešams izveidot un izmantot simetriskā algoritma objektu. Ja ir piešķirta vērtība īpašībai `Key` un turpmāk nepieciešams izmantot patvaļīgu atslēgu, ir jāizsauc metode `GenerateKey`.

Patvaļīgu atslēgu ģenerēšanai nav ieteicams izmantot klasi `System.Random`, jo tā nav paredzēta gadījuma skaitļu ģenerēšanai kriptogrāfijas nolūkiem. Tās vietā jāizmanto klase `System.Security.Cryptography.RNGCryptoServiceProvider`.

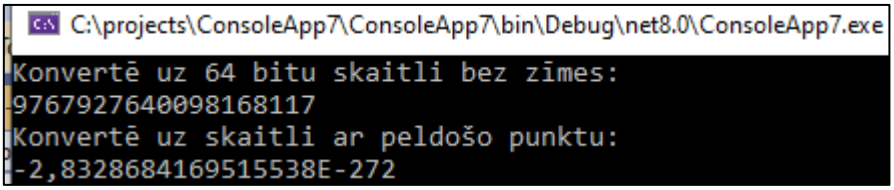
Klase `System.Security.Cryptography.RNGCryptoServiceProvider` ģenerē gadījuma vērtības, izsaucot metodi `GetBytes`. Turklāt tiek izveidoti patvaļīgu baitu masīvi, kurus pēc tam vajag pārveidot vajadzīgajā formātā. Šim nolūkam kalpo klase `System.BitConverter`, kura satur pārveidošanas metodes uz populārākajiem skaitļu formātiem. 7.2. piemērā dots koda fragments, kas ģenerē patvaļīgas baitu vērtības un pārveido tās 64 bitu skaitlī un peldošā punkta tipa skaitlī.

7.2. piemērs. Patvaļīgu baitu vērtību ģenerēšana

```
using System;
using System.Text;
using System.IO;
```

```
using System.Security.Cryptography;

namespace generate
{
   class Program
   {
      static void Main(string[] args)
      {
         RNGCryptoServiceProvider rand = new RNGCryptoServiceProvider();
         byte[] randValue = new byte[256];
         rand.GetBytes(randValue);
         Console.WriteLine("Konvertē uz 64 bitu skaitli bez zīmes:");
         Console.WriteLine(System.BitConverter.ToUInt64(randValue, 0));
         Console.WriteLine("Konvertē uz skaitli ar peldošo punktu:");
         Console.WriteLine(System.BitConverter.ToDouble(randValue, 0));
         Console.ReadLine();
      }
   }
}
```

Rezultāts



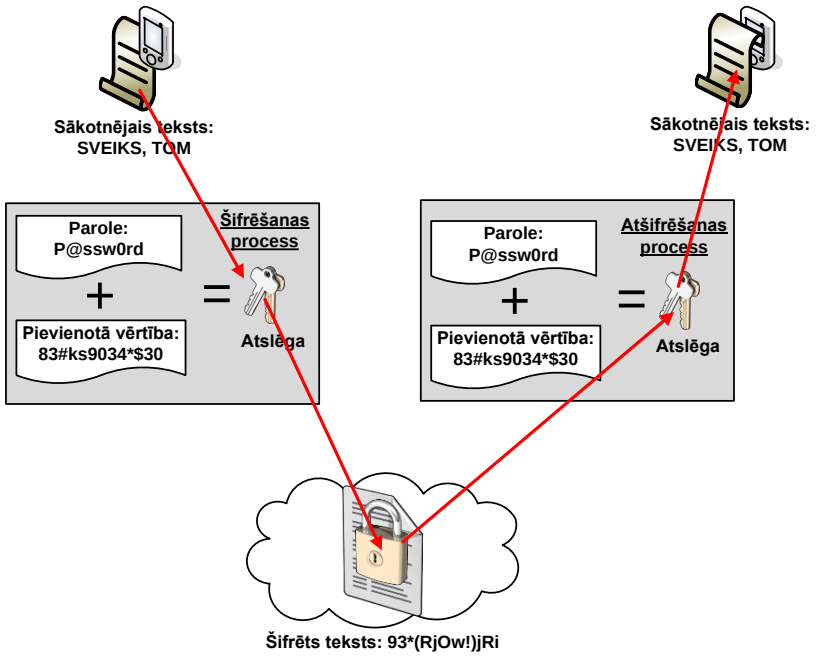
Izplatīts paņēmiens atslēgu ģenerēšanā ir paroles izmantošana (sk. 7.3. att.). Par šifrēšanas atslēgu paroli tiešā veidā izmantot nevar, taču var pārveidot paroli par atslēgu ar klases `System.Security.Cryptography.PasswordDeriveBytes` palīdzību. Šī klase izmanto šādas vērtības:

- paroli;
- parolei patvaļīgu pievienotu vērtību;
- inicializācijas vektoru;
- iterāciju skaitu atslēgas veidošanas operācijai.

Šo vērtību izmaiņa izraisa citas atslēgas ģenerēšanu, tāpēc šifrēšanā un atšifrēšanā jāizmanto vienādas vērtības, kuras jānodod konstruktoram `PasswordDeriveBytes` kopā ar izmantojamo jaucējfunkcijas algoritmu:

```
public PasswordDeriveBytes(
    string strPassword,
    byte[] rgbSalt;
    string strHashName,
    int iterations
);
```

Pēc inicializācijas var iegūt atslēgu, izsaucot metodi `PasswordDeriveBytes.GetBytes`. Metode `GetBytes` pieņem atgriežamo baitu skaitu. Atslēgas iegūšanā jānosaka tās garums, ņemot vērā bitu skaitu īpašībās `KeySize` un `BlockSize`. Tā kā metode `PasswordDeriveBytes.GetBytes` pieņem baitus, tad bitu skaits jādala ar 8.



7.3. att. **Atslēgas izveide uz paroles pamata atvieglo konfidenciālā ziņojuma nodošanu**

7.3. piemērā tiek veidota atslēga un inicializācijas vektors uz paroles pamata, kura tiek uzdota programmas mainīgajā **aaa**.

7.3. piemērs. Atslēgas izveidošana uz paroles pamata

```
using System;
using System.Text;
using System.IO;
using System.Security.Cryptography;

namespace passwordkey
{
   class Program
   {
      static void Main()
      {
         string aaa = "abc";              //mainīgajā aaa var uzdot dažādas paroļu vērtības
         SymmetricAlgorithm myAlg = new RijndaelManaged();
         // nosaka vērtību, ko pievienos parolei
         byte[] saltValueBytes = Encoding.ASCII.GetBytes("saltValue");

         // veidojam objektu PasswordDeriveBytes ar paroli un vērtībām
         PasswordDeriveBytes passwordKey = new PasswordDeriveBytes(aaa, saltValueBytes,
                                                "SHA1", 3);

         //ģenerē atslēgu
         myAlg.Key = passwordKey.GetBytes(myAlg.KeySize / 8);
         myAlg.IV = passwordKey.GetBytes(myAlg.BlockSize / 8);

         //izvada ģenerēto atslēgu heksadecimālajā formā
         Console.WriteLine("Nogenerētā atslēga 16-ku veidā:");
         Console.Write("0x");
         foreach (byte thisByte in myAlg.Key)
            Console.Write(thisByte.ToString("X2")+ " ");
```

```
            Console.WriteLine();
            Console.ReadLine();
        }
    }
}
```

Mainīgajā **aaa** uzdod paroles vērtību *abc*. Tiek ģenerēta atslēga un izvadīta ekrānā.

Rezultāts

Ja šifrēšanā un atšifrēšanā tiek izmantota viena un tā pati atslēga, var sākties apmaiņa ar šifrētiem datiem. .NET *Framework* vidē šifrēšanas izmantošana ļoti līdzīga darbam ar datnēm un plūsmām. Lai lietotne varētu veikt šifrēšanu un atšifrēšanu, nepieciešams izpildīt šādas darbības:

1) jāizveido objekts `Stream`, lai vērstos pie datnes, no kuras nolasa datus vai kurā ieraksta datus;
2) jāizveido objekts `SymmetricAlgorithm`;
3) jānorāda atslēga un inicializācijas vektors;
4) jāizsauc metode `SymmetricAlgorithm.Create.Encryptor()` vai `SymmetricAlgorithm.Create.Decryptor()`, lai varētu izveidot objektu `CryptoTransform`;
5) jāizveido objekts `CryptoStream`, izmantojot objektus `Stream` un `ICryptoTransform`;
6) jānolasa vai jāieraksta dati objektā `CryptoStream`, līdzīgi kā jebkurā objektā `Stream`.

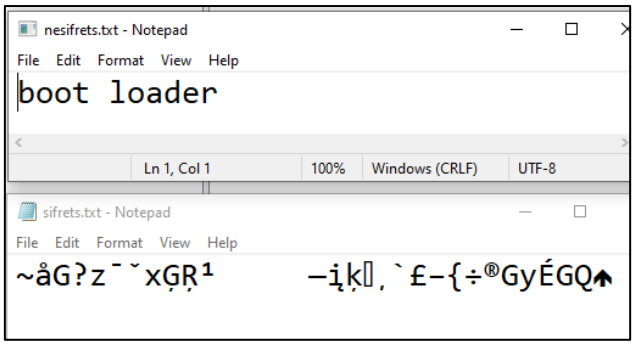
7.4. piemērs demonstrē datu šifrēšanu. Datnē **nesifrets.txt** ieraksta tekstu "**boot loader**", datnē **sifrets.txt** iegūst rezultātu.


7.4. piemērs. Datu šifrēšana ar simetriskā algoritma palīdzību

```
using System;
using System.Text;
using System.IO;
using System.Security.Cryptography;

namespace symmetric
{
   class Program
   {
      static void Main()
      {
         string unencryptedFileName = @"D:\A\nesifrets.txt";
         string encryptedFileName = @"D:\A\sifrets.txt";
         // 1.solis. izveido objektu Stream
         FileStream unencryptedFile = new FileStream(unencryptedFileName, FileMode.Open,
                                              FileAccess.Read);
         FileStream encryptedFile = new FileStream(encryptedFileName,
                                          FileMode.OpenOrCreate, FileAccess.Write);

         // 2.solis. veido objektu SymmetricAlgorithm
         SymmetricAlgorithm myAlg = new RijndaelManaged();

         // 3.solis (nav obligāts).  norāda atslēgu
```

```
            myAlg.GenerateKey();
            // nolasa nešifrēto datni iekš fileData
            byte[] fileData = new byte[unencryptedFile.Length];
            unencryptedFile.Read(fileData, 0, (int)unencryptedFile.Length);

            // 4.solis.  veido objektu ICryptoTransform;
            ICryptoTransform encryptor = myAlg.CreateEncryptor();

            // 5.solis.  veido objektu CryptoStream;
            CryptoStream encryptStream = new CryptoStream(encryptedFile, encryptor,
                                                CryptoStreamMode.Write);

            // 6.solis.  ieraksta saturu objektā CryptoStream;
            encryptStream.Write(fileData, 0, (int)unencryptedFile.Length);

            encryptStream.Close();
            unencryptedFile.Close();
            encryptedFile.Close();
            Console.ReadLine();
        }
    }
}
```

Rezultāts



Lietotnes darbības rezultātā katru reizi tiks ģenerēta cita atslēga. Tā kā atslēga netiek saglabāta, datni nevarēs atšifrēt. Atslēga ir parasts baitu masīvs, kuru var saglabāt ar objekta `BinaryWriter` palīdzību.

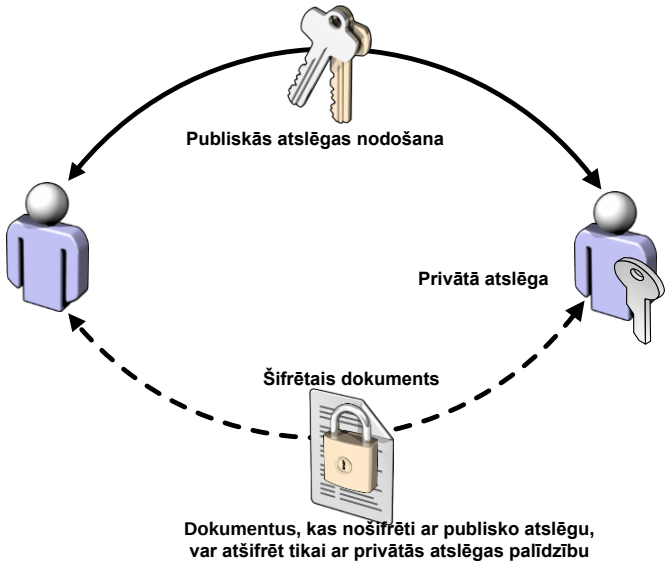
Datnes atšifrēšanas kods gandrīz identisks šifrēšanas kodam ar to izņēmumu, ka atšifrējot ir jānolasa atslēga, nevis jāģenerē. Lietotnes kodā jāizdara šādas izmaiņas:

- jāmaina kods 3.solī, lai nolasītu atslēgu un inicializācijas vektoru;
- jāmaina kods 4.solī, lai tiktu izmantota metode `CreateDecryptor`.
- jāmaina kods 5.solī, lai tiktu izmantota `CryptoStreamMode.Read`.

7.2. Asimetriskās kriptogrāfijas ieviešana

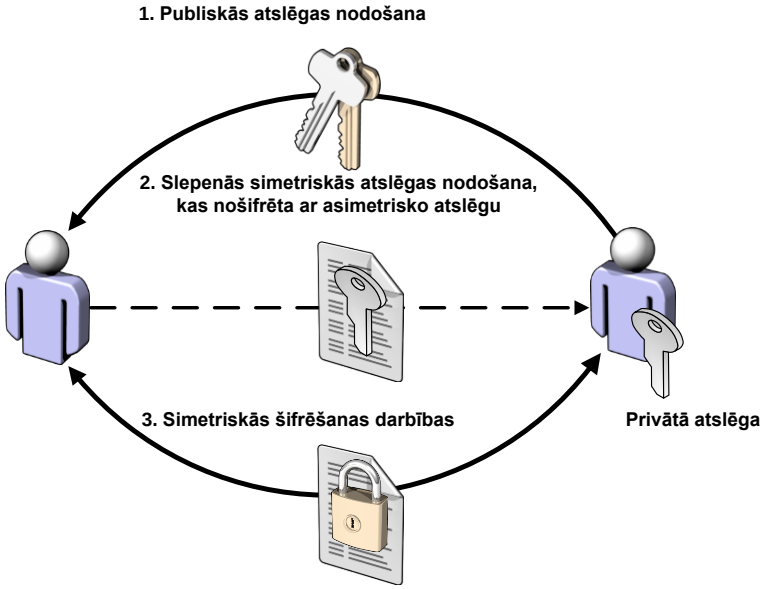
Šifrēšana ar asimetrisko atslēgu jeb šifrēšana ar publisko atslēgu ir kriptogrāfisks paņēmiens, kurā gan šifrēšanai, gan atšifrēšanai izmanto divas dažādas atslēgas – publisko un privāto. Šifrēšanā izmanto privāto atslēgu (jātur slepenībā no nepilnvarotiem lietotājiem) un publisko atslēgu (var nodot jebkuram). Publiskā un privātā atslēga ir matemātiski saistīta. Datus, kas nošifrēti ar publisko atslēgu, var atšifrēt tikai ar privātās atslēgas palīdzību. Savukārt datus, kas parakstīti ar privāto atslēgu, var apstiprināt tikai ar publiskās atslēgas palīdzību. Publisko atslēgu var padarīt pieejamu visiem lietotājiem, to izmanto datu šifrēšanai, kas tiks nogādāti privātās atslēgas īpašniekam.

Asimetriskās šifrēšanas process sākas ar publiskās atslēgas nodošanu. Pēc apmaiņas ar publiskajām atslēgām, dati tiek šifrēti ar saņēmēja publiskās atslēgas palīdzību. Atšifrēt šos datus var tikai atslēgas īpašnieks, kura privātā atslēga atbilst šifrēšanā izmantotajai publiskajai atslēgai. Vienkāršs asimetriskās šifrēšanas piemērs parādīts 7.4. attēlā.



7.4. att. **Asimetriskais algoritms izmanto dažādas atslēgas šifrēšanai un atšifrēšanai**

Asimetriskie algoritmi nedarbojas tik ātri kā simetriskie, toties tie ir daudz drošāki. Viens no izplatītākajiem asimetrisko algoritmu pielietojumu veidiem ir simetriskās atslēgas un inicializācijas vektora šifrēšana pirms pārraides. Tad simetriskās šifrēšanas algoritms tiek izmantots visu pārraidāmo paziņojumu apstrādei. Šādu paņēmienu izmanto protokoli HTTPs un SSL pārraidāmo datu šifrēšanai *Internet* tīklā – asimetrisko šifrēšanu pielieto tikai seansa noteikšanā. Tāda simetriskās un asimetriskās šifrēšanas kombinācija parādīta 7.5. attēlā.



7.5. att. **Simetrisko un asimetrisko algoritmu apvienošana paaugstina aizsardzības drošību**

Vēl viena būtiska asimetrisko algoritmu priekšrocība slēpjas atslēgu pārvaldībā – PKI infrastruktūra (vairāk par to tika stāstīts 2.5. apakšnodaļā). Lietotņu izstrādātāji parasti nenodarbojas ar PKI pielāgošanu.

.NET *Framework* sastāvā ietilpst divas klases darbam ar asimetrisko šifrēšanu, kas iegūtas no klases `System.Security.Cryptography.AsymmetricAlgorithm`. Bāzes klasei ir šādas īpašības (daudzas ir identiskas klases `SymmetricAlgorithm` īpašībām):

- `KeyExchangeAlgorithm` – norāda atslēgu apmaiņas algoritma nosaukumu (parasti nav nepieciešamības vērties pie šīs īpašības tiešā veidā);
- `KeySize` – slepenās atslēgas, ko izmanto simetriskais algoritms, izmērs bitos;
- `LegalKeySizes` – masīvs no `KeySize`, kur uzglabājas simetriskā algoritma atslēgu izmēri. Katram masīva elementam piemīt īpašības `MinSize` un `MaxSize`, kas nosaka atslēgu pieļaujamos diapazonus bitos, un īpašība `SkipSize`, kas norāda intervālu bitos starp pieļaujamajiem atslēgu izmēriem;
- `SignatureAlgorithm` – XML dokumenta URL adrese, kurā apraksta paraksta algoritmu (parasti nav nepieciešamības vērties pie šīs īpašības tiešā veidā).

Atšķirībā no bāzes klases `SymmetricAlgorithm` klasei `AsymmetricAlgorithm` nav lietderīgu metožu. Tā vietā šifrēšanas funkcionalitāte iebūvēta objektos, kas realizē klasi `AsymmetricAlgorithm`. .NET *Framework* nodrošina šīs klases divas realizācijas:

1) `RSACryptoServiceProvider` – tiek izmantota visās asimetriskās šifrēšanas un atšifrēšanas operācijās, tā ir RSA algoritma .NET *Framework* realizācija;
2) `DSACryptoServiceProvider` – tiek izmantota sūtījumu ciparparakstam.

Klasei `RSACryptoServiceProvider` ir šādas papildu īpašības:

- `PersistKeyInCsp` – vērtība, kas norāda, vai atslēgai jāglabājas kriptogrāfisko pakalpojumu sniedzējā (CSP). Parasti uzdod vērtību `true`, ja vēlas atkārtoti izmantot atslēgu, to neeksportējot;
- `UseMachineKeyStore` – vērtība, kas norāda, vai atslēgai jāglabājas datora atslēgu glabātavā, bet ne lietotāja profila glabātavā.

Konstruktori pēc noklusējuma vienmēr piešķir algoritma parametriem tādas vērtības, lai iegūtu pašu drošāko algoritma variantu no pieļaujamajiem. Klasei `RSACryptoServiceProvider` arī ir savas šifrēšanas un atšifrēšanas metodes, kā arī metodes atslēgu importēšanai un eksportēšanai:

- `Decrypt` – atšifrē datus, kas šifrēti ar RSA algoritmu;
- `Encrypt` – šifrē datus ar RSA algoritma palīdzību;
- `ExportParameters` – eksportē struktūru `RSAParameters`, kas nosaka algoritma atslēgu pāri. Ja vērtība ir `true`, tad tiek eksportētas abas atslēgas. Ja vērtība ir `false`, tad tiek eksportēta tikai publiskā atslēga;
- `FromXmlString` – importē atslēgu pāri XML rindā;
- `ImportParameters` – importē publisko atslēgu vai atslēgu pāri norādītajam objektam `RSAParameters`;
- `SignData` – izskaitļo uzrādīto datu jaucējfunkcijas vērtību un saglabā parakstu baitu masīvā;
- `SignHash` – izskaitļo parakstu uzrādītās jaucējfunkcijas vērtībai, šifrējot to ar privātās atslēgas palīdzību, un saglabā parakstu baitu masīvā;
- `VerifyData` – pārbauda uzrādīto datu parakstu, salīdzinot to ar parakstu, kas izskaitļots uzrādītajiem datiem;
- `VerifyHash` – pārbauda uzrādīto datu parakstu, salīdzinot to ar parakstu, kas izskaitļots uzrādītajai jaucējfunkcijas vērtībai.

RSA atslēgas ir daudz sarežģītākas par simetriskās šifrēšanas atslēgām. RSA atslēgas saucas par parametriem, kas iekļauti struktūrā `RSAParameters`. Lai izprastu parametru būtību, īsumā jāapskata RSA algoritma darbības princips. RSA algoritmā atslēgu pāru ģenerēšana sākas ar divu lielu pirmskaitļu izvēli – **p** un **q**. Šos skaitļus sareizina un iegūst **n**. Tā kā abi skaitļi **p** un **q** ir

pirmskaitļi, tad kopīgie reizinātāji **n** ir skaitļi **1**,**p**,**q** un **n**. Ja apskata skaitļus, kas mazāki par **n**, tad savstarpējo pirmskaitļu (kam nav kopīgu reizinātāju ar **n**) skaits ir **(p-1)(q-1)**. Izvēlas tādu skaitli **e**, kas savstarpēji ir pirmskaitlis kādai izskaitļotai vērtībai. Publisko atslēgu tagad var uzdot veidā **{e, n}**. Lai iegūtu privāto atslēgu, ir jāizskaitļo tāds **d**, lai **(d)(e) mod n = 1**. Saskaņā ar Eiklīda algoritmu privātā atslēga tagad ir veidā **{d, n}**. Vienkārša teksta **m** šifrēšana ir noteikta kā **c = (m ^ e) mod n**. Attiecīgi dešifrēšana – **m = (c ^ d) mod n**. RSA algoritma drošību nosaka tas, ka dotajai publiskajai atslēgai **{e, n}** praktiski nav iespējams izskaitļot **d** (detalizētāku RSA algoritma aprakstu skatiet speciālajā literatūrā).

7.2. tabulā parādīti svarīgākie `RSAParameters` struktūras locekļi un to apzīmējumi.

7.2. tabula

`RSAParameters` struktūras locekļi

Parametrs	Saturs	Apraksts
D	*d*	Privātā atslēga
DP	*d mod (p - 1)*	
DQ	*d mod (q - 1)*	
Exponent	*e*	Publiskā atslēga
InverseQ	*(InverseQ)(q) = 1 mod p*	
Modulus	*n*	
P	*p*	
Q	*q*	

Gandrīz vienmēr ir nepieciešamība eksportēt publisko atslēgu, jo bez tās neviens nevarēs aizsūtīt privātās atslēgas īpašniekam šifrētus datus. Lai varētu eksportēt publisko atslēgu struktūras `RSAParameters` eksemplārā, nepieciešams izmantot metodi `RSACryptoServiceProvider.ExportParameters`, nododot tam vērtību `false`.

 Privāto atslēgu ieteicams eksportēt tikai tajos gadījumos, ja to nepieciešamas izmantot atkārtoti.

Nākamās divas koda rindiņas parāda, kā notiek RSA algoritma eksemplāra izveide un automātiski ģenerētās publiskās atslēgas eksportēšana uz objektu `RSAParameters` ar nosaukumu `publicKey`.

```
RSACryptoServiceProvider myRSA = new RSACryptoServiceProvider();
RSAParameters publicKey = myRSA.ExportParameters(false);
```

Lai varētu šifrēt un atšifrēt ziņojumus ar asimetrisko algoritmu, ir jāizsauc metodes `RSACryptoServiceProvider.Encrypt` un `RSACryptoServiceProvider.Decrypt`. Abām metodēm ir divi parametri:

- `byte[] rbg` – baitu masīvs, satur paziņojumu, kuru vajag nošifrēt vai atšifrēt;
- `bool fOAEP` – ja vērtība `true`, tad datu aizpildīšana notiks pēc metodes OAEP. Ja vērtība `false`, tad datu aizpildīšana notiks pēc metodes PKCS#1 v1.5.

Datu aizpildīšana ir asimetriskajos algoritmos izmantotais kriptogrāfiskais paņēmiens, lai aizsargātos pret uzbrukumiem, kas vērsti pret paziņojumu šifrēšanu, t.i., šifrējamais ziņojums tiek pārveidots baitu masīva formā un tikai tad šifrēts. .NET *Framework* RSA algoritmam uztur divas aizpildīšanas shēmas:

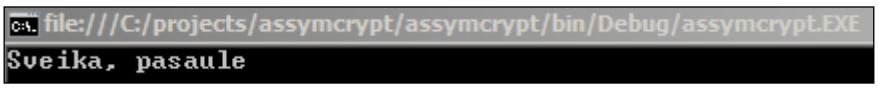
- OAEP (*Optimal Asymmetric Encryption Padding*) – uztur operētājsistēmas, sākot ar *Windows XP* (ieteicams lietot šo shēmu, jo tā ir jaunāka un drošāka);
- PKCS#1 v1.5 – uztur vecākās operētājsistēmas.

Lai pārveidotu tekstu baitu masīvos, jāizmanto metodes `Encoding.Unicode.GetBytes` un `Encoding.Unicode.GetString`. 7.5. piemērā demonstrētais kods šifrē paziņojuma rindu, izmantojot datu aizpildīšanas shēmu OAEP, un pēc tam uzreiz atšifrē to.

7.5. piemērs. Datu šifrēšana ar asimetriskā algoritma palīdzību

```
using System;
using System.Text;

using System.Security.Cryptography;

namespace assymcrypt
{
   class Program
   {
      static void Main(string[] args)
      {
         string messageString = "Sveika, pasaule";
         RSACryptoServiceProvider myRSA = new RSACryptoServiceProvider();
         byte[] messageBytes = Encoding.Unicode.GetBytes(messageString);
         byte[] encryptedMessage = myRSA.Encrypt(messageBytes, true);

         byte[] decryptedBytes = myRSA.Decrypt(encryptedMessage, true);
         Console.WriteLine(Encoding.Unicode.GetString(decryptedBytes));
         Console.ReadLine();
      }
   }
}
```

Rezultāts



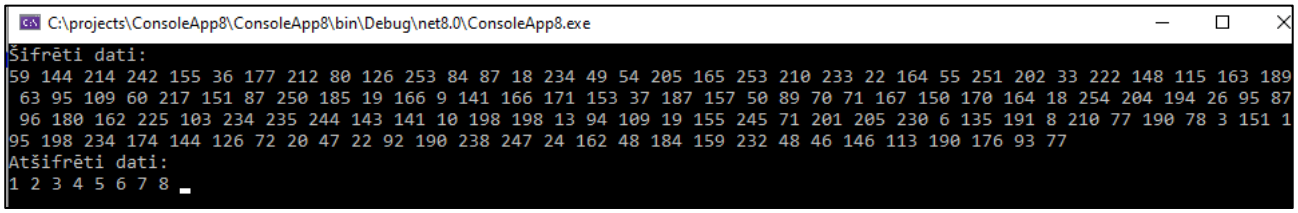
7.6. piemērs ir līdzīgs iepriekšējam – tiek šifrēts skaitļu masīvs.

7.6. piemērs. Datu šifrēšana un atšifrēšana ar asimetriskā algoritma palīdzību

```
using System;
using System.IO;
using System.Security.Cryptography;

namespace assymcrypt1
{
   class Program
   {
      static void Main(string[] args)
      {
         RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();
         Byte[] testData = { 1, 2, 3, 4, 5, 6, 7, 8 };                    // šifrējamie dati
         Byte[] encryptedData = rsa.Encrypt(testData, true);
         Console.WriteLine("Šifrēti dati:");
         for (int i = 0; i < encryptedData.GetLength(0); i++)
         {
            Console.Write("{0} ", encryptedData[i]);
         }
         Byte[] decryptedData = rsa.Decrypt(encryptedData, true);
                  Console.WriteLine();
         Console.WriteLine("Atšifrēti dati:");
         for (int i = 0; i < decryptedData.GetLength(0); i++)
```

```
        {
            Console.Write("{0} ", decryptedData[i]);
        }
        Console.ReadLine();
    }
  }
}
```

Rezultāts



Ja objekta `RSACryptoServiceProvider` izveidošanas brīdī nepieciešams atslēgu pāris, šis objekts tās automātiski izveido. Ja šifrējamie dati kaut kādu iemeslu dēļ nav jāsaglabā, tad, protams, nav jāglabā arī atslēgas. Taču, ja lietotnei jāatšifrē dati, kas šifrēti citviet, tad nepieciešams uzglabāt izmantojamo atslēgu pāris. Klase `RSACryptoServiceProvider` ļauj saglabāt atslēgas kriptogrāfisko pakalpojumu sniedzējā CSP.

Lai privātā atslēga tiktu ievietota glabātavā, nepieciešams veikt šādas darbības:

1) jāizveido objekts `CspParameter`;
2) jāuzrāda īpašība `Csp.Parameters.KeyContainerName`;
3) jāizveido objekts `RSACryptoServiceProvider`, izmantojot objektu `CspParameter`;
4) jāpiešķir īpašībai `RSACryptoServiceProvider.PersistKeyInCsp` vērtība `true`.

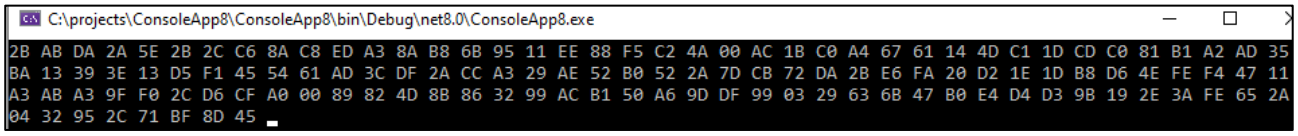
.NET *Framework* veiks atslēgu izveidošanu un atslēgu saglabāšanu automātiski. Uzreiz pēc objekta `CspParameter` izveides un `true` vērtības piešķiršanas īpašībai `PersistKeyInCsp` .NET *Framework* izveidos atslēgu konteineru un saglabās tur atslēgu. Pēc atkārtotas lietotnes palaišanas .NET *Framework* konstatēs, ka konteiners ar tādu atslēgu jau eksistē un izņems no tā privāto atslēgu. Ja 7.7. piemērā dotā lietotne tiks palaista vairākas reizes, rezultātā ikreiz tiks parādīta viena un tā pati atslēga.

 7.7. piemērs. Privātās atslēgas saglabāšana

```
using System;
using System.Text;
using System.Security.Cryptography;

namespace savekeys
{
  class Program
  {
    static void Main(string[] args)
    {
      CspParameters persistantCsp = new CspParameters();
      persistantCsp.KeyContainerName = "AsymmetricPiemērs";
      RSACryptoServiceProvider myRSA = new RSACryptoServiceProvider(persistantCsp);
      myRSA.PersistKeyInCsp = true;
      RSAParameters privateKey = myRSA.ExportParameters(true);

      foreach (byte thisByte in privateKey.D)
```

```
        {
            Console.Write(thisByte.ToString("X2") + " ");
        }
        Console.ReadLine();
    }
  }
}
```

Rezultāts



7.3. Jaucējfunkciju un ciparparaksta pielietošana

Vēl viens svarīgs kriptogrāfijas pielietojums ir datu integritātes pārbaude ar jaucējfunkciju palīdzību. Jaucējfunkcijas vērtība (*hash*) ir kontrolsumma, kas ir unikāla katrai datnei vai datu fragmentam. To izmanto pārbaudei – vai datne netika izmainīta pēc jaucējfunkcijas vērtības ģenerēšanas.

Integritāte (*integrity*) – sistēmu, programmu un datu aizsardzība pret netīšu vai ļaunprātīgu bojāšanu vai pārveidošanu.

Jaucējfunkcijas vērtība tiek iegūta, summējot datu fragmenta elementus, un to var izmantot apstiprinājumam, ka dati netika izmainīti kopš jaucējfunkcijas vērtības iegūšanas brīža.

.NET *Framework* sastāvā ietilpst pieci jaucējfunkciju algoritmi bez atslēgām un divi ar atslēgām (sk. 7.3. un 7.4. tabulu). Visi algoritmi bez atslēgām realizēti ar klases `System.Security.Cryptography.HashAlgorithm` palīdzību.

7.3. tabula

Jaucējfunkciju algoritmi bez atslēgām

Abstraktā klase	Realizācijas klase	Apraksts
MD5	*MD5CryptoServiceProvider*	Algoritms *Message Digest*. Jaucējfunkcijas vērtības izmērs – 128 biti.
SHA1	*SHA1CryptoServiceProvider*	Algoritms *Secure Hash Algorithm 1*. Jaucējfunkcijas vērtības izmērs – 160 biti.
SHA256	*SHA256Managed*	Algoritms *Secure Hash Algorithm 256*. Jaucējfunkcijas vērtības izmērs – 256 biti.
SHA384	*SHA384Managed*	Algoritms *Secure Hash Algorithm 384*. Jaucējfunkcijas vērtības izmērs – 384 biti.
Sha512	*SHA512Managed*	Algoritms *Secure Hash Algorithm 512*. Jaucējfunkcijas vērtības izmērs – 512 biti.

Jaucējfunkciju algoritmi ar atslēgām nodrošina jaucējfunkciju vērtību aizsardzību, šifrējot to ar slepenās atslēgas palīdzību. Visi algoritmi ar atslēgām realizēti ar klases `System.Security.Cryptography.KeyedHashAlgorithm` palīdzību.

7.4. tabula

Jaucējfunkciju algoritmi ar atslēgām

Klase	Apraksts
HMACSHA1	Sūtījumu autentificēšanas kods (*Message Authentication Code*) uz jaucējfunkcijas vērtības pamata izmanto SHA1 algoritmu. Ļauj noteikt, vai sūtījums tika izmainīts, piešķirot sūtītājam un saņēmējam slepeno atslēgu. HMACSHA1 uztur jebkāda garuma atslēgas un veido jaucējfunkcijas vērtību 20 baitu garumā.
MACTripleDES	Sūtījumu autentificēšanas kods (*Message Authentication Code*) izmanto TripleDES algoritmu. Līdzīgi iepriekšējam algotitmam ļauj noteikt, vai sūtījums tika izmainīts, piešķirot sūtītājam un saņēmējam kopīgu slepeno atslēgu. MACTripleDES uztur 8, 16 vai 24 baitu garas atslēgas un veido jaucējfunkcijas vērtību 8 baitu garumā.

Lai varētu izskaitļot jaucējfunkcijas vērtību bez atslēgas, jāveic šādas darbības (sk. 7.8. piemērs):

1) jāizveido algoritma objekts;
2) jāsaglabā baitu masīvā datus, kuriem grib izskaitļot jaucējfunkcijas vērtību;
3) jāizsauc metode `HashAlgorithm.ComputeHash`;
4) tiks iegūts baitu masīvs `HashAlgorithm.Hash`, kas satur jaucējfunkcijas vērtību.

7.8. piemērs. Jaucējfunkcijas vērtības bez atslēgas izskaitļošana (mainīgajā **aaa** ir datnes nosaukums, kas satur tekstu "Sveika, pasaule")

```
using System;
using System.Text;
using System.IO;
using System.Security.Cryptography;

namespace md5hash
{
  class Program
  {
    static void Main()
    {
      string aaa = @"D:\A\dati.txt";
      MD5 myHash = new MD5CryptoServiceProvider(); // 1.solis
      FileStream file = new FileStream(aaa, FileMode.Open, FileAccess.Read); //2.solis
      BinaryReader reader = new BinaryReader(file);
      myHash.ComputeHash(reader.ReadBytes((int)file.Length)); // 3.solis
      Console.WriteLine(Convert.ToBase64String(myHash.Hash)); // 4.solis
      Console.ReadLine();
    }
  }
}
```

Rezultāts

```
C:\projects\ConsoleApp8\ConsoleApp8\bin\Debug\net8.0\ConsoleApp8.exe
vsTl8RJXiTquUQ3NVcqE3Q==
```

Ja šī lietotne tika palaista vairākas reizes pēc kārtas, rezultāts būs viens un tas pats, ja tikai netiks pamainīts datnes saturs.

Lai varētu izskaitļot jaucējfunkcijas vērtību ar atslēgām, jāveic šādas darbības (sk. 7.9. piemērs):

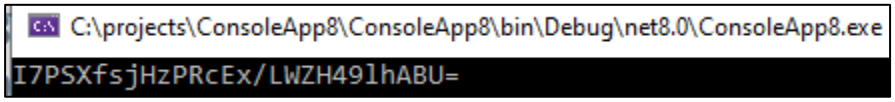
1) jāizveido slepenā atslēga un jānodod abām pusēm;
2) jāizveido algoritma objekts ar slepenās atslēgas palīdzību (ja slepenā atslēga nav uzrādīta, tā tiks ģenerēta automātiski);
3) jāsaglabā baitu masīvā datus, kuriem grib izskaitļot jaucējfunkcijas vērtību;
4) jāizsauc metode `KeyedHashAlgorithm.ComputeHash`;
5) tiks iegūts baitu masīvs `KeyedHashAlgorithm.Hash`, kas satur jaucējfunkcijas vērtību.


7.9. piemērs. Jaucējfunkcijas vērtības ar atslēgām izskaitļošana

```
using System;
using System.Text;
using System.IO;
using System.Security.Cryptography;

namespace shahash
{
   class Program
   {
      static void Main(string[] args)
      {
         string parole = "abc";              //mainīgajā parole var uzdot dažādas paroļu vērtības
         string fails = @"D:\A\dati.txt";  // datne satur tekstu "Sveika, pasaule"

         byte[] saltValueBytes = Encoding.ASCII.GetBytes("Šitā ir mana slepenā atslēga");  //1
         PasswordDeriveBytes passwordKey = new PasswordDeriveBytes(parole, saltValueBytes,
                                                   "SHA1", 3);
         byte[] secretKey = passwordKey.GetBytes(16);

         HMACSHA1 myHash = new HMACSHA1(secretKey);  // 2.solis
         FileStream file = new FileStream(fails, FileMode.Open, FileAccess.Read); //3.solis
         BinaryReader reader = new BinaryReader(file);
         myHash.ComputeHash(reader.ReadBytes((int)file.Length)); // 4.solis
         Console.WriteLine(Convert.ToBase64String(myHash.Hash)); // 5.solis
         Console.ReadLine();
      }
   }
}
```

Mainīgajā **parole** uzdod paroli, mainīgajā **fails** datnes nosaukumu, kurai tiks izskaitļota jaucējfunkcijas vērtība.

Rezultāts



Ciparparakstu tehnoloģijā faktiski ir apvienoti asimetriskie algoritmi ar jaucējfunkciju vērtībām. Ciparparaksts ir vērtība, kas tiek pievienota elektroniskajiem datiem, lai apstiprinātu, ka šos datus ir izveidojis noteiktas privātās atslēgas īpašnieks. Citiem vārdiem sakot, ciparparaksta jaucējfunkcijas vērtība aizsargā datni pret izmaiņām, bet publiskā atslēga garantē to, ka jaucējfunkcijas vērtību ir ģenerējis privātās atslēgas īpašnieks.


Ciparparaksts neaizsargā parakstīto datu konfidencialitāti. Tāpēc datus vajag šifrēt.

.NET *Framework* sastāvā iekļautas divas klases, kas ļauj ģenerēt un pārbaudīt ciparparakstus: `DSACryptoServiceProvider` un `RSACryptoServiceProvider`. Šīs klases izmanto dažādus algoritmus, bet piedāvā līdzīgas funkcijas. Katra klase realizē četras metodes darbam ar ciparparakstiem:

1) `SignHash` – veido ciparparakstu uz datnes jaucējfunkcijas vērtības pamata;
2) `SignData` – veido ciparparakstu. No sākuma datnei tiek ģenerēta jaucējfunkcijas vērtība, uz kuras pamata tad arī tiek ģenerēts ciparparaksts;
3) `VerifyHash` – pārbauda ciparparakstu uz datnes jaucējfunkcijas vērtības pamata;
4) `VerifyData` – pārbauda ciparparakstu uz visa datnes satura pamata.

Ciparparaksti izmanto asimetrisko šifrēšanu. Tas nozīmē, ka saņēmējs nevar ģenerēt parakstu, nezinot sūtītāja privāto atslēgu (tomēr pārbaudīt parakstu ar publiskās atslēgas palīdzību var). Metodes `VerifyData` un `VerifyHash` izmanto sūtītāja publisko atslēgu, bet metodes `SignData` un `SignHash` izmanto sūtītāja privāto atslēgu.

Lai datnei tiktu izveidots ciparparaksts, jāveic šādas darbības:

1) jāizveido ciparparaksta algoritma objekts;
2) jāsaglabā parakstāmie dati baitu masīvā;
3) jāizsauc metode `SignData` un jāsaglabā paraksts;
4) jāeksportē publiskā atslēga.

Lai ciparparaksts tiktu pārbaudīts, jāveic šādas darbības:

1) jāizveido ciparparaksta algoritma objekts;
2) jāimportē paraksts un publiskā atslēga;
3) jāsaglabā pārbaudāmie dati baitu masīvā;
4) jāizsauc metode `VerifyData`.

7.10. piemērā parādīta ciparparaksta izveidošana datnei, kuras nosaukums lietotnei tiek nodots kā arguments. Publiskā atslēga un ciparparaksts tiek saglabāti mainīgajos un pēc tam tiek pārbaudīts ciparparaksts.

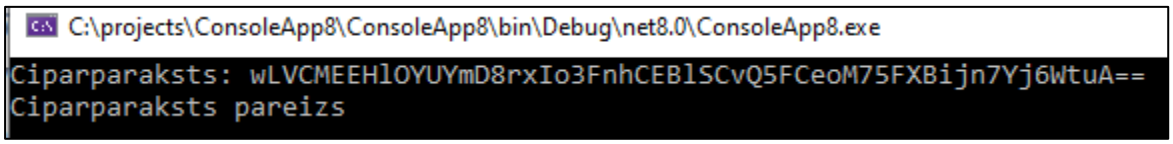
 7.10. piemērs. Datnes ciparparaksta izveidošana un pārbaude

```
using System;
using System.Text;
using System.IO;
using System.Security.Cryptography;

namespace ciparparaksts
{
   class Program
   {
      static void Main()
      {
         string fails = @"D:\A\dati.txt";  // datne satur tekstu "Sveika, pasaule"

        // Ciparparaksta izveide
         DSACryptoServiceProvider signer = new DSACryptoServiceProvider(); // 1.solis
         FileStream file1 = new FileStream(fails, FileMode.Open, FileAccess.Read); //2.solis
         BinaryReader reader1 = new BinaryReader(file1);
         byte[] data1 = reader1.ReadBytes((int)file1.Length);
         byte[] signature = signer.SignData(data1); // 3.solis
         string publicKey = signer.ToXmlString(false);  // 4.solis
         Console.WriteLine("Ciparparaksts: " + Convert.ToBase64String(signature));
```

```
            reader1.Close();
            file1.Close();
            // Ciparparaksta pārbaude
            DSACryptoServiceProvider verifier = new DSACryptoServiceProvider(); // 1.solis
            verifier.FromXmlString(publicKey);  //2.solis
            FileStream file2 = new FileStream(fails, FileMode.Open, FileAccess.Read); //3.solis
            BinaryReader reader2 = new BinaryReader(file2);
            byte[] data2 = reader2.ReadBytes((int)file2.Length);
            if (verifier.VerifyData(data2, signature))  // 4.solis
            {
                Console.WriteLine("Ciparparaksts pareizs");
            }
            else
            {
                Console.WriteLine("Ciparparaksts nepareizs");
            }
            reader2.Close();
            file2.Close();
            Console.ReadLine();
        }
    }
}
```

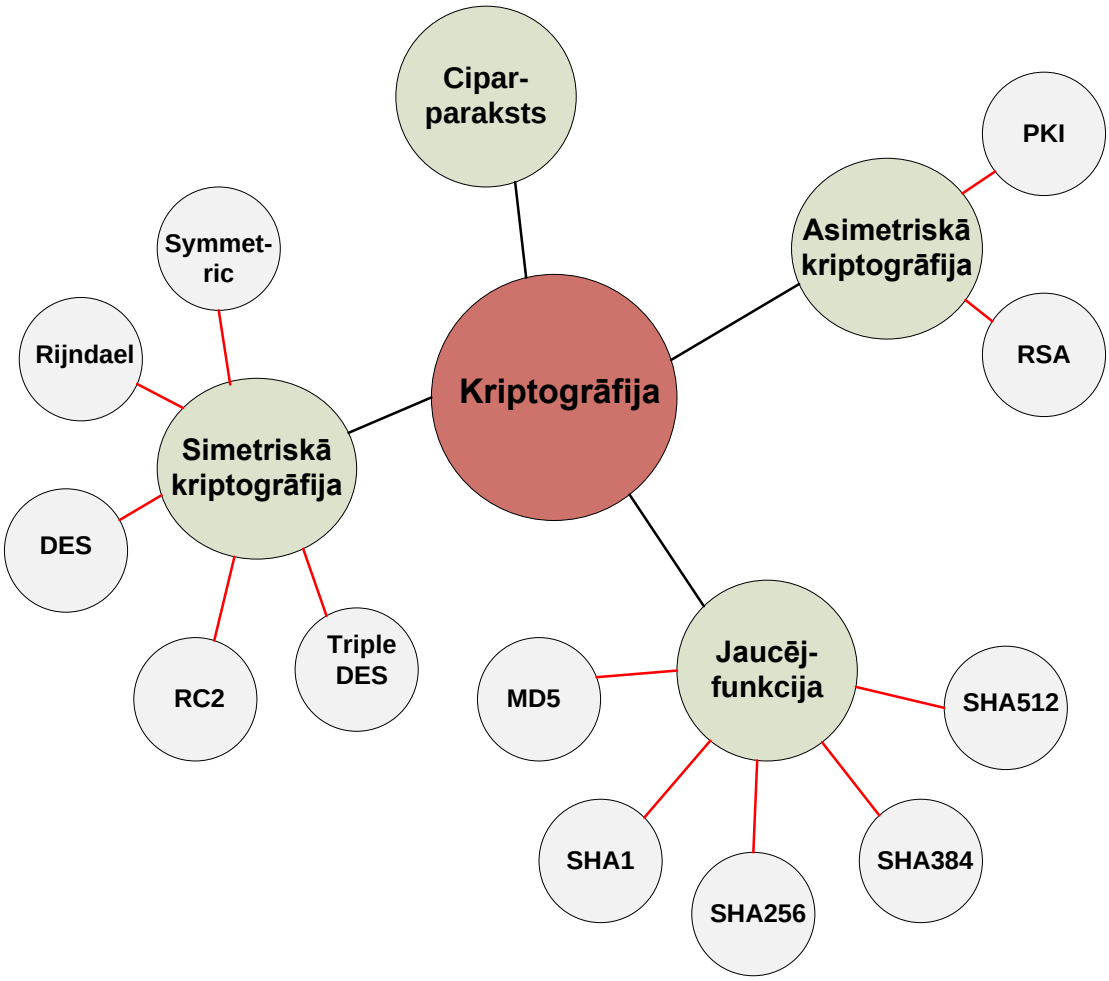
Mainīgajā **fails** ir datnes nosaukums, kurai tiks izskaitļots ciparparaksts.

Rezultāts



Šajā piemērā ciparparaksts tiek izveidots un pārbaudīts ar vienas un tās pašas lietotnes palīdzību. Parasti publiskā atslēga un ciparparaksts tiek pārsūtīti tīklā, tāpēc labākais ciparparakstu pārraides risinājums būtu binārās datnes izveidošana, kura saturētu publisko atslēgu, ciparparakstu un pašu datni. Protams, ka visu to var sūtīt arī atsevišķu datņu veidā.

7.6. attēlā parādīts nodaļā minēto svarīgāko jēdzienu koks.



7.6. att. **Septītajā nodaļā minēto svarīgāko jēdzienu koks**

Pārbaudes testa jautājumi

1. Tiek ieviests programmnodrošinājums bankomātiem. Kādu simetrisko algoritmu vajadzētu izvēlēties drošības nodrošināšanai?
    a. *DES*
    b. *3DES*
    c. *RC2*
    d. *Rijndael*

2. Kādu no klasēm jāizvēlas kriptogrāfiski drošu gadījuma skaitļu ģenerēšanai?
    d. *RandomNumberGenerator*
    e. *RNGCryptoServiceProvider*
    f. *RSACryptoServiceProvider*
    g. *SymmetricAlgorithm*

3. Kura no šīm metodēm ir piederīga SymmetricAlgorithm objektam?
    d. *SymmetricAlgorithm tripledes = new SymmetricAlgorithm()*
    e. *SymmetricAlgorithm tripledes = new SymmetricAlgorithm.Create("TripleDES")*
    f. *SymmetricAlgorithm tripledes = new TripleDES.Create()*
    a. Visas minētās

4. Kurus no šiem parametriem izmanto RSA algoritms?
    a. DP
    b. DQ
    c. InverseQ
    d. PQ

5. Kuri no apgalvojumiem nav patiesi par jaucējfunkciju algoritmiem?
    a. Jaucējfunkcijas pārveido patvaļīga garuma binārās rindas par nelielām fiksēta garuma binārajām rindām
    b. Nav iespējams atrast divus atšķirīgus ievaddatu datu fragmentus, kam tiktu izskaitļotas vienādas jaucējfunkciju vērtības
    c. Nelielu datu izmaiņu rezultātā tiek veiktas lielas un neparedzamas jaucējfunkciju vērtību izmaiņas
    d. Visu īstenoto jaucējfunkciju algoritmu klases ir tieši iegūtas no *HashAlgorithm* bāzes klases

6. Kādas informācijas nodošanai no sūtītāja saņēmējam izmanto klašu *DSACryptoServiceProvider* un *RSACryptoServiceProvider* metodes *ExportParameters* un *ImportParameters*?
    a. *Public* key
    b. *Hash*
    c. *Message digest*
    d. Nekas no minētā

8. ASP.NET programmu aizsardzība

Tā kā Internet tīkla lietotnes vairāk tiek pakļautas dažādiem uzbrukumiem, tad šāda tipa lietotnēm nepieciešama paaugstināta drošība. ASP.NET lietotņu drošības nodrošināšanā izšķir trīs fundamentālus drošības mehānismus: autentificēšana, pilnvarošana un personifikācija. Tīmekļa lietotņu lietotāju autentificēšana un pilnvarošana tiek veikta savādāk nekā Windows formu autentificēšana un pilnvarošana. Turklāt klienta un servera mijiedarbība tiek veikta bez datu šifrēšanas (ja nav uzstādīts SSL protokols). Tāpēc ir svarīgi izprast ASP.NET lietotņu drošības nodrošināšanas iespējas.

Nodaļā apskatīta tīmekļa lietotņu autentificēšanas mehānismu realizācija noteiktu drošības iestatījumu prasību veikšanai. Tiek izskaidrotas pilnvarošanas iespējas piekļuves tīmekļa lietotnēm nodrošināšanai, kā arī parādīti tīmekļa mapju un datņu aizsardzības paņēmieni [27].

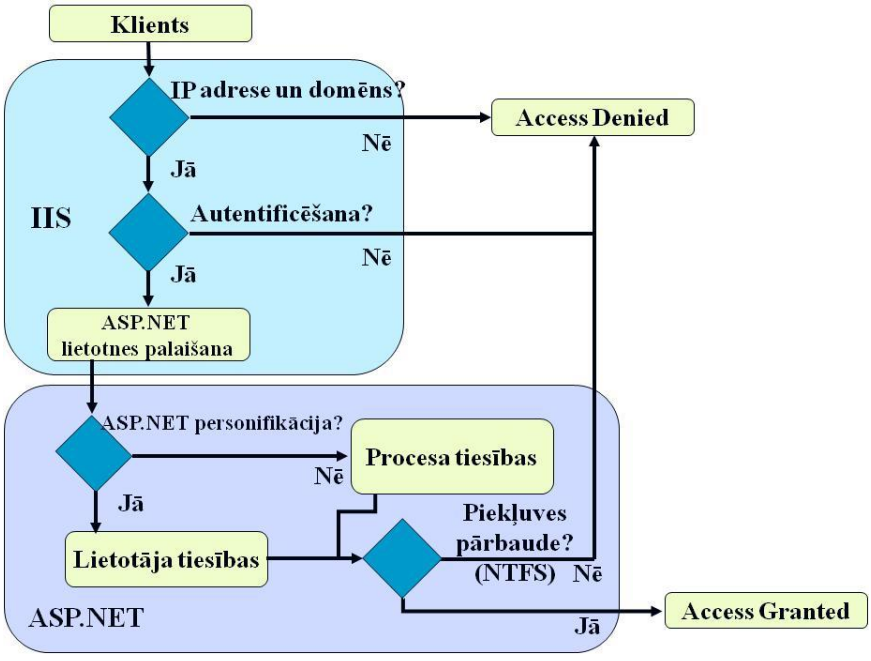
8.1. Autentificēšanas ieviešana ASP.NET

Tīmekļa lietotnes uzstāda specifiskas prasības autentificēšanai, jo lietotāju uzskaites dati jāpārsūta ne pa parastajiem autentificēšanas protokoliem, piemēram, *Kerberos*, bet starp tīmekļa pārlūkprogrammu (*Web browser*) un tīmekļa serveri (*Web server*). Šo prasību nodrošināšanai ASP.NET uztur četrus standarta autentificēšanas tipus:

1) *Windows* autentificēšana;
2) anonīmā piekļuve;
3) autentificēšana uz formu pamata;
4) autentificēšana *Passport* sistēmā.

Nedrīkst jaukt ASP.NET autentificēšanas tipus ar IIS autentificēšanas tipiem (sk. 8.1. tabulu), kas tiek izmantoti lietotāju autentificēšanai tikai tīmekļa serverī, bet ne lietotnē.

IIS serveris nodod lietotāju uzskaites datus ASP.NET lietotnei, ļaujot tai veikt *Windows* autentificēšanu, pamatojoties uz tiem pašiem datiem, kas bija uzrādīti IIS serverim (IIS komponenti tika apskatīti 2.1. apakšnodaļā). IIS un ASP.NET mijiedarbības shēma parādīta 8.1. attēlā.



8.1. att. **IIS un ASP.NET mijiedarbības shēma**

8.1.tabula

IIS un ASP.NET aizsardzības nodrošinājums

IIS nodrošinājums	ASP.NET nodrošinājums
• Autentificēšana: - *Anonymous* - *Basic* - *Windows Integrated* - Sertifikāts • Pilnvarošana - NTFS • Ierobežošana pēc domēna un IP adreses • Lietotņu izolācija (*App pool*) • SSL	• Autentificēšana - *Forms* - *Passport* - *Windows* (IIS) - *Anonymous* - Savs variants • Pilnvarošana - URL - Lomas - Piekļuve kodam • Lietotņu izolācija (*AppDomain*)

Visām šīm autentificēšanas metodēm nepieciešama IIS dienestu saskaņošana un konfigurācijas datņu iestatīšana.

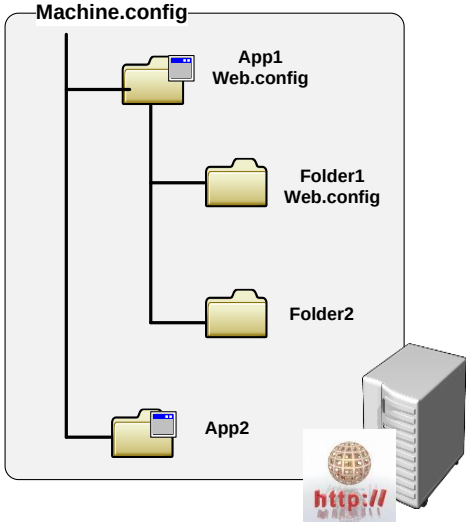
ASP.NET satur divas konfigurācijas datnes: `Machine.config` un `Web.config` (sk. 8.2. tabulu).

8.2.tabula

ASP.NET konfigurācijas datnes

`Machine.config`	`Web.config`
• Izmanto servera konfigurācijas uzdošanai • Uzstādīts kopā ar .NET *Framework* • Atrodas mapē *%WinDir%\Microsoft.Net\Framework\version\Config* • Datnes iestatījumi attiecas uz visām ASP.NET mapēm un pakļautajām mapēm	• Lieto konkrētās lietotnes konfigurācijas uzdošanai • Atrodas ASP.NET lietotnes saknes mapē • Katrai ASP.NET tīmekļa lietotnes pakļautajai mapei var būt sava `Web.config` datne

Datnes `Web.config` satur vienā mapē izvietotas ASP.NET lietotnes datņu konfigurācijas parametrus – tā saucamās sekcijas. Tajās var pārdefinēt augstāka līmeņa konfigurācijas datnes `Machine.config` (šāda datne ir tikai viena) parametrus atbilstoši konkrētās lietotnes specifikai. `Machine.config` parametri attiecas uz visām .NET lietotnēm, ieskaitot arī .ASP.NET lietotnes. Tādas parametru hierarhijas piemērs dots 8.2. attēlā.



8.2. att. **`Web.config` datnes nodrošina drošības parametru hierarhiju**

Šajā piemērā mape *\App1* satur datni `Web.config`. Tādējādi *\App1* manto no `Machine.config` visus parametrus, izņemot tos, kas pārdefinēti datnē `Web.config`. Mape *\Folder1* arī satur datni `Web.config`, kurā pārdefinēti mapes *\App1* datnes `Web.config` un datora datnes `Machine.config` parametri. Mape *\Folder2* nesatur konfigurācijas datni, tāpēc manto visus mapes *\App1* parametrus. Mape *\App2* arī nesatur konfigurācijas datni, tāpēc manto datnē `Machine.config` noteiktos parametrus.

 `Machine.config` atrodas mapē *C:\Windows\Microsoft.NET\Framework\versija\CONFIG*, bet `Web.config` – konkrētās lietotnes saknes mapē.

Datne `Machine.config` nosaka arī parametrus, kas nevar tikt pārdefinēti. Tādi ir parametri ar īpašību `allowDefinition`, kuru vērtība ir `MachineOnly`. Ja īpašības vērtība ir `MachineToApplication`, tad to var pārdefinēt katrā lietotnē, bet tikai tajā `Web.config` datnē, kas atrodas lietotnes saknes mapē. Piemēram, parametru `authentication` var uzdot katrai lietotnei atsevišķi, jo tās noteiktā īpašības `allowDefinition` vērtība ir `MachineToApplication`, t. i., sekciju `authentication` var uzrādīt tikai lietotnes saknes konfigurācijas datnē. Savukārt sekcija `authorization` var tikt pārdefinēta lietotnes pakļautajās mapēs, jo tai tiešā veidā nav noteikta īpašība `allowDefinition`.

Ja izstrādājamā ASP.NET lietotne tiks izmantota iestādes ietvaros un lietotāju uzskaites dati atrodas servera lokālajā datu bāzē, tad pietiek ar *Windows* autentificēšanas izmantošanu. Šādu autentificēšanu var iestatīt divējādi – ar IIS un pašā ASP.NET lietotnē (IIS konfigurēšana šajā mācību līdzeklī netiek apskatīta). Vispusīgas aizsardzības nodrošināšanai tiek ieteikts izmantot abus paņēmienus.

Ja lietotne izmanto *Windows* autentificēšanu, tad tā noraida visus pieprasījumus, kas nesatur pieļaujamo lietotāja vārdu un paroli. Pārlūkprogramma piedāvā lietotājam ievadīt vārdu un paroli, tāpēc nav nepieciešamības izstrādāt savu tīmekļa lappusi lietotāja uzskaites datu pieprasījumam. Turklāt arī nav nepieciešamības izstrādāt savu datu bāzi lietotāju uzskaites datu glabāšanai. Tā rezultātā *Windows* autentificēšana ir pats vienkāršākais autentificēšanas mehānisms.

Lai lietotnē tiktu veikta *Windows* autentificēšana, datnē `Web.config` ir jānodefinē sekcija `<authentication>`, kas atrodas sekcijā `<system.web>`. Nākamais koda fragments demonstrē datnes `Web.config` sekciju `<authentication>`, kas iestatīta *Windows* autentificēšanas izmantošanai (izveidojot ASP.NET lietotnes projektu *Visual Studio* vidē, šie parametri tiek iestatīti pēc noklusējuma):

```
<?xml version="1.0"?>
    <configuration>
        <system.web>
            <authentication mode="Windows"/>
             <authorization>
                  <deny users=”?”/>
             </authentication>
        </system.web>
    </configuration>
```

Šajā gadījumā sekcija `<authentication>` pieprasa, lai visi lietotāji veiksmīgi izietu autentificēšanu. Pilnvarošanas parametra `<deny users=”?”/>` iekļaušana (tā tiks apskatīta 8.2. apakšnodaļā) pieprasa lietotājiem iziet autentificēšanu. Attiecīgi parametrs `<allow users=”*”/>` ļauj izlaist autentificēšanu.

Var tiešā veidā atslēgt autentificēšanu, ja zināms, ka to izmantos tikai anonīmi lietotāji. Gadījumos, kad lietotne neprasa autentificēšanu, datnē `Web.config` nav nepieciešams iekļaut sekciju `<authentication>`, bet uzticēt sistēmas administratoriem konfigurēt autentificēšanu pašā IIS.

Koda fragments, kas ļauj anonīmi piekļūt ASP.NET lietotnei, izskatās šādi:

```
<?xml version="1.0"?>
     <configuration>
          <system.web>
               <authentication mode="None"/>
          </system.web>
     </configuration>
```

Windows autentificēšanas gadījumā lietotājs ievada savus datus pārlūkprogrammas dialoga logā. Šāda pieeja ļauj automatizēt autentificēšanu tīmeklī, taču atstāj ļoti maz iespēju izstrādātājam. Tāpēc parasti tiek izmantota autentificēšana uz formu pamata. Lietotāja uzskaites dati tiek ievadīti atsevišķā HTML lappusē. Pēc autentificēšanas veikšanas pārlūkprogramma saglabā *cookie* datni ar informāciju par lietotāju un turpmāk izmanto šo datni, vēršoties pie vietnes, ļaujot ASP.NET lietotnei pārbaudīt pieprasījumus.

Koda fragments, kas pieprasa izmantot autentificēšanu uz formu pamata, izskatās šādi:

```
<configuration>
     <system.web>
          <authentication mode="Forms" />
               <forms loginUrl=”LoginForms.aspx” />
          </authentication>
          <authorization>
               <deny users=”?” />
          </authentication>
     </system.web>
</configuration>
```

Šajā piemērā visi lietotāji, kas vēršas pie jebkuras ASP.NET datnes, tiks pāradresēti uz lappusi **`LoginForms.aspx`**. Parasti lietotājam formā ir jāievada vārds un parole, tad lietotne veiks autentificēšanu ar saviem līdzekļiem. Neatkarīgi no tā, kādā veidā lietotne apstrādā ievadītos datus, pēc noklusējuma tie tiek nosūtīti uz serveri HTTP pieprasījumu veidā bez jebkādas šifrēšanas. HTTP protokols tikai nodrošina pārlūkprogrammas mijiedarbību ar serveri. Labākais konfidencialitātes nodrošināšanas līdzeklis lietotnes formā ievadītajiem datiem būtu SSL sertifikāta konfigurēšana IIS serverī un prasība izmantot HTTPS (*Hypertext Transfer Protocol Secure*) protokolu.

Lietotāja vārdu un paroli var salīdzināt ar datiem, kas glabājas, piemēram, datu bāzē, datnē `Web.config`, XML datnē vai citā vietā. Lai gan autentificēšana uz formu pamata ir ļoti elastīga, tomēr visa atbildība par drošas autentificēšanas ieviešanu gulstas uz izstrādātāju. Piemēram, tā kā ziņas par sekmīgas autentificēšanas iziešanu glabājas *cookie* datnē, kas parasti satur tikai lietotāja vārdu, tad ļaunprātīgs lietotājs var izveidot viltus *cookie* datni, lai liktu serverim uzskatīt, ka autentificēšana ir veiksmīgi izdarīta. Tāpēc ASP.NET pastāv iespēja šifrēt un pārbaudīt *cookie* datnes, taču tāda aizsardzība prasa no servera papildus resursus.

Šifrēšanas tipu un pārbaudes nosaka sekcijas `<forms>` atribūts `protection`. Ja tas nav uzrādīts, pēc noklusējuma atribūta vērtība ir `All`. Ja šī atribūta vērtība ir `Encryption`, tad *cookie* datne tiek šifrēta ar algoritma 3-DES palīdzību. Šifrēšana nodrošina *cookie* datnē esošo datu konfidencialitāti, taču nenodrošina to pārbaudi. Ja atribūta `protection` vērtība ir `Validation`, tad serveris pārbauda *cookie* datnes datus katras transakcijas laikā, lai pārliecinātos, ka tie netika izmainīti pēc tam, kad pārlūkprogramma tos aizsūtīja uz serveri:

```
<authentication mode="Forms" protection=”Validation” >
     <forms loginUrl=”LoginForms.aspx” />
</authentication>
```

Ja atribūta `protection` vērtība ir `None`, tad šifrēšana un pārbaude netiek veikta.

 Optimālā drošības līmeņa nodrošināšanai ieteicams atstāt noklusējuma vērtību `All`.

Vēl viens svarīgs sekcijas `<forms>` atribūts ir `timeout`, kas nosaka laika intervālu starp pieprasījumiem, pēc kura lietotājam atkārtoti nāksies iziet autentificēšanu. Ja, piemēram, sekcija `<forms>` ir uzdota ar šādiem atribūtiem:

```
<forms loginUrl="LoginForms.aspx" timeout="10">,
```

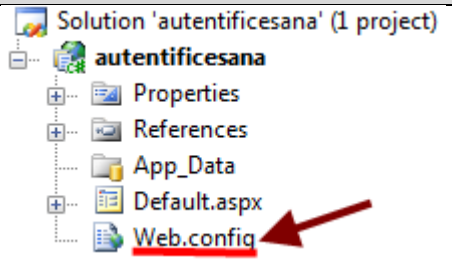
tad lietotājam nāksies atkārtoti veikt autentificēšanos, ja 10 minūšu laikā ASP.NET lietotnei netiks nosūtīts neviens pieprasījums.

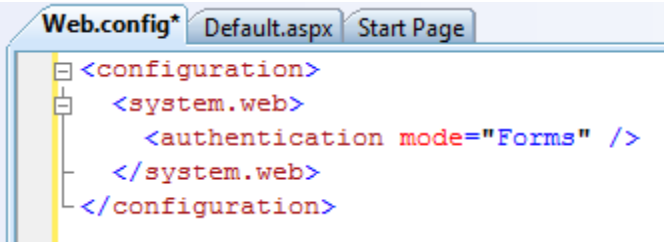
Sekcijai `<forms>` ir arī citi atribūti, taču `loginUrl`, `protection` un `timeout` ir svarīgākie no tiem.

Vispārīgā gadījumā formu autentificēšanā izmanto divas `Web.config` datnes: viena atrodas lietotnes saknes mapē un uzrāda informāciju par autentificēšanas tipu, otra atrodas mapē, kur atrodas ar paroli aizsargājamās lapas, un nosaka pilnvarošanas iespējas.

8.1. piemērā demonstrēta lietotnes lapas aizsardzība ar formu autentificēšanas palīdzību. Veicamās darbības aprakstītas soli pa solim.

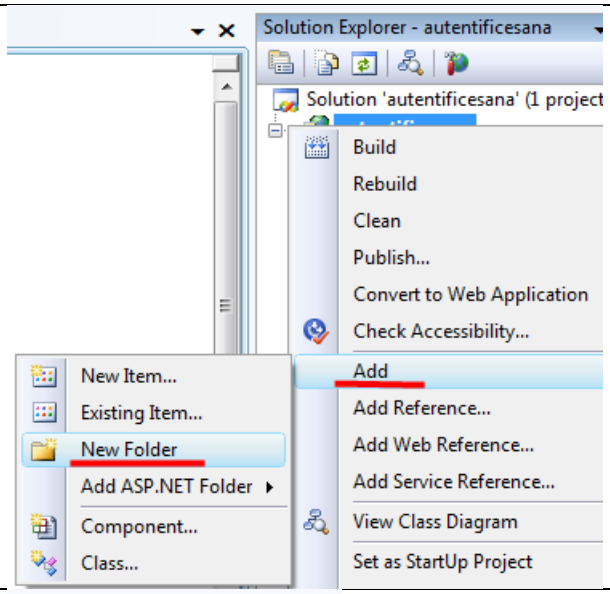
8.1. piemērs. Autentificēšana uz formu pamata

1.Visual Studio vidē izveido jaunu **ASP.NET Web Application (.NET Framework)** ar nosaukumu **autentificesana – Web Forms**	
2.**Solution Explorer** logā veic dubultklikšķi uz saknes mapes **`Web.config`** datnes.	

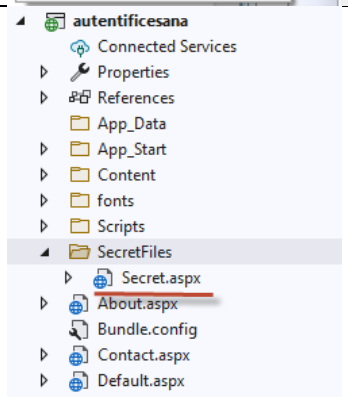
3.Izdzēš visu **`Web.config`** datnes saturu un uzraksta šādus konfigurācijas parametrus:
`<configuration>`
`<system.web>`
`<authentication mode="Forms" />`
`</system.web>`
`</configuration>`
	

4. Saglabā izmaiņas **`Web.config`** datnē.	
Turpmākajos soļos tiek veidota lapa, kas būtu jāaizsargā.	

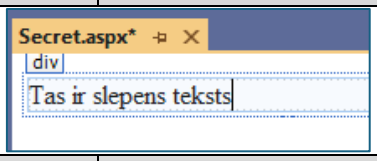
5. Izveido jaunu mapi ar nosaukumu **SecretFiles**, kurā atradīsies aizsargājamās lapas.



6. Mapē **SecretFiles** pievieno jaunu komponentu- **Web Form** ar nosaukumu **Secret** un apstiprina ar pogu **OK**.



7. Lapas **Secret.aspx** panelī **Design** ieraksta tekstu: „Tas ir slepens teksts".



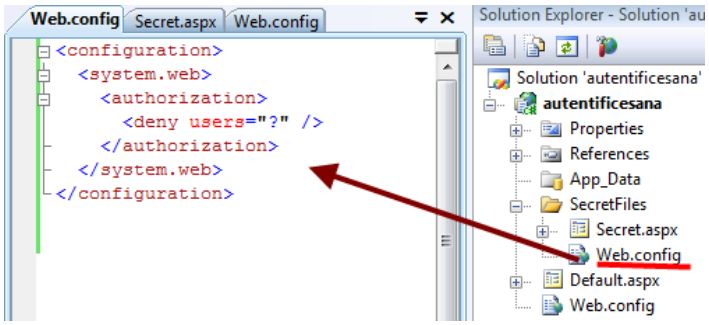
8. Datņu aizsardzībai mapē **SecretFiles** nepieciešams otra **`Web.config`** datne.

Izvēlas mapi **SecretFiles**, labais peles taustiņš, **Add**, **Add New Item** un izvēlas **Web Configuration File**.

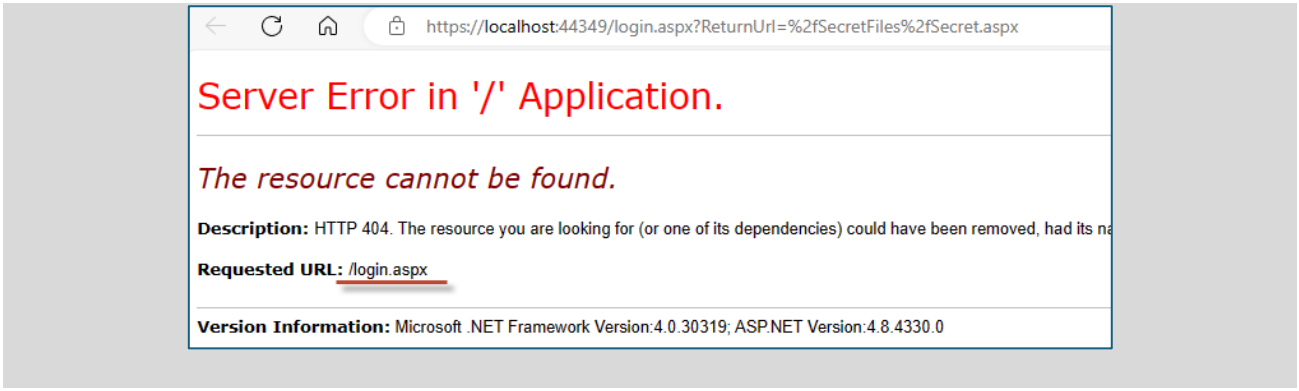
9. Izdzēš visu **`Web.config`** datnes saturu un uzraksta šādus konfigurācijas parametrus:

```
<configuration>
<system.web>
  <authorization>
    <deny users="?" />
  </authorization>
  </system.web>
</configuration>
```

un saglabā šo datni.

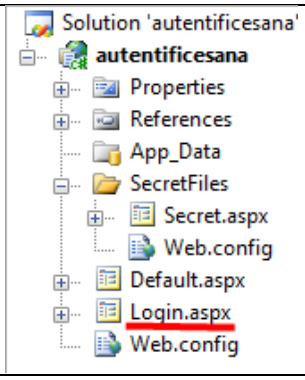
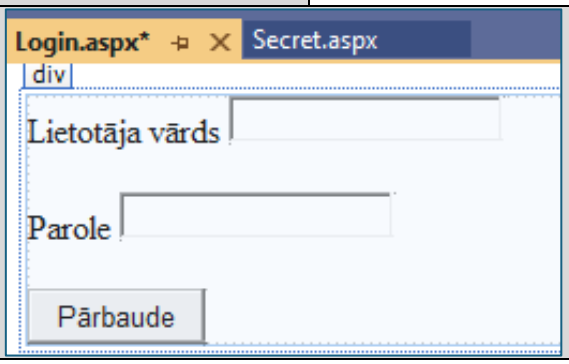


Šo darbību rezultātā visas mapes **SecretFiles** datnes ir aizsargātas ar paroli, un, mēģinot palaist datni **Secret.aspx**, tiks saņemts kļūdas paziņojums par to, ka lietotāju autentificēšana pēc noklusējuma tiek veikta lapā **Login.aspx**, kuras pagaidām nav.



Ja lietotājs grib vērsties pie lapas, kas aizsargāta ar paroli, tas automātiski tiek pāradresēts uz lapu, kurā tiek veikta lietotāja autentificēšana. Piemēra turpinājumā tiks izveidota vienkārša **Login** lapa, kurā jāievada lietotāja vārds un parole. Sekmīgas autentificēšanās rezultātā (piemērā lietotāja vārds ir **Administrator** un parole ir **Secret**) tiks attēlots aizsargātās lapas saturs: „Tas ir slepens teksts".

8.2. piemērs. Vienkārša autentificēšanas lapa

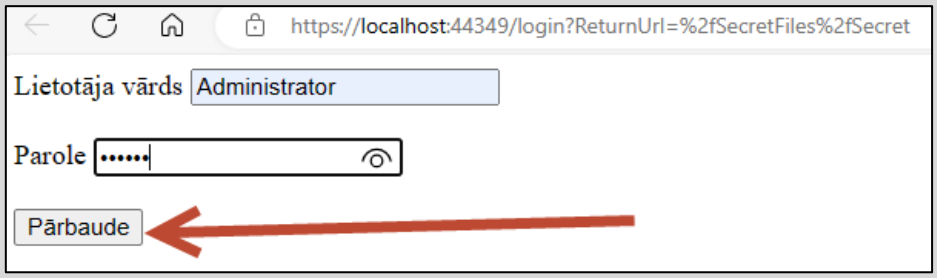
1. Līdzīgi kā iepriekšējā piemēra 6. punktā, pievieno jaunu **Web Form** ar nosaukumu **Login** projekta saknes mapē.	
2. No **Toolbox** standarta rīku joslas uz formu pārvelk vadības objektu **Label** un **TextBox** lietotāja vārda ievadīšanai.	Properties — Vērtība **Label1 (Text)** — Lietotāja vārds **TextBox (ID)** — txtUsername
3. No **Toolbox** standarta rīku joslas uz formu pārvelk vadības objektu **Label** un **TextBox** paroles ievadīšanai.	Properties — Vērtība **Label2 (Text)** — Parole **TextBox (ID)** — txtPassword **TextBox (TextMode)** — Password
4. No **Toolbox** standarta rīku joslas uz formu pārvelk vadības objektu **Button** lietotāja datu pārbaudes veikšanai.	Properties — Vērtība **Button (Text)** — Pārbaude
	

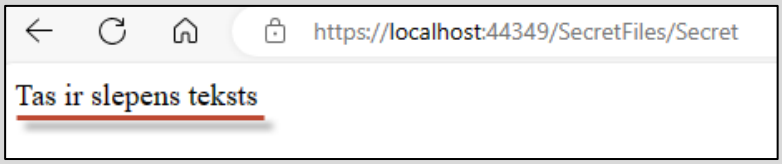
5. Pēc peles dubultklikšķa uz pogas **Pārbaude** pievieno programmas kodu lietotāja vārda un paroles pārbaudei:

```
if (txtUsername.Text == "Administrator" && txtPassword.Text == "Secret")

System.Web.Security.FormsAuthentication.RedirectFromLoginPage(txtUsername.Text,
false);
     else
 Label1.Text = "Nepareizs vārds vai parole";
```

```
protected void Button1_Click(object sender, EventArgs e)
{

if (txtUsername.Text == "Administrator" && txtPassword.Text == "Secret")
   System.Web.Security.FormsAuthentication.RedirectFromLoginPage(txtUsername.Text,
       false);
else
   Label1.Text = "Nepareizs vards vai parole";
```

Lietotnes darbības pārbaudes nolūkā palaiž datni **Secret.aspx**. Notiks pāradresācija uz lapu **Login.aspx** un veikta lietotāja vārda un paroles pārbaude. Veiksmīgas autentificēšanas rezultātā tiks parādīts aizsargātās lapas saturs:





Metode `RedirectFromLoginPage(userName,createPersistentCookie)` pāradresē autentificēšanu veikušo lietotāju uz sākotnēji pieprasīto URL lapu (vai uz lapu pēc noklusējuma). Parametrs `userName` satur lietotāja vārdu, bet parametrs `createPersistentCookie` var pieņemt vērtību *true* (lai tiktu izveidota ilglaicīga *cookie* datne) vai *false* (pretējā gadījumā).

Iepriekšējos piemēros lietotāja vārds un parole tika uzdoti programmas kodā. Var glabāt lietotāja uzskaites datus arī `Web.config` datnē. Paroles var uzdot trijos veidos:

1) atklātā veidā;
2) šifrētā veidā ar MD5 palīdzību;
3) šifrētā veidā ar SHA1 palīdzību.

Jaucējfunkciju vērtību izmantošana samazina risku, ka ļaunprātīgs lietotājs iegūs citu lietotāju uzskaites datus no datnes `Web.config`. Izmantojamais jaucējfunkciju algoritms tiek noteikts sekcijā `<credentials>`.

Koda fragments, kad lietotāja vārds un parole uzdoti atklātā veidā, izskatās šādi:

```
<configuration>
   <system.web>
     <authentication mode="Forms">
```

```
          <forms>
            <credentials passwordFormat="Clear">
            <user name="Bob" password="Secret" />
            <user name="Jane" password="Secret" />
            <user name="Fred" password="Secret" />
            </credentials>
          </forms>
        </authentication>
      </system.web>
      </configuration>
```

Koda fragments, kad parole nošifrēta, izskatās šādi:

```
<authentication mode="Forms">
     <forms loginUrl="login.aspx" protection="Encryption" timeout="30" >
         <credentials passwordFormat="SHA1" >
            <user name="Peter"
                  password="07B7F3EE06F278DB966BE960E7CBBD103DF30CA6"/>
            <user name="Anna"
                  password="5753A498F025464D72E088A9D5D6E872592D5F91"/>
         </credentials>
     </forms>
</authentication>
```

Jaucējfunkciju vērtības no konsoles ievadītai parolei var aprēķināt pēc 8.3. piemērā dotā parauga un pēc tam ievietot tās `Web.config` datnē.

8.3. piemērs. Jaucējfunkciju vērtību iegūšana izmantošanai `Web.config` datnē

```
using System;
using System.Security.Cryptography;
using System.Text;

namespace HashExample
{
   class Program
   {
      static void Main()
      {
         string aaa = "abc";              //mainīgajā aaa var uzdot dažādas paroļu vērtības
         SHA1CryptoServiceProvider myHash = new SHA1CryptoServiceProvider();
         byte[] password = Encoding.ASCII.GetBytes(aaa);
         myHash.ComputeHash(password);
         foreach (byte thisByte in myHash.Hash)
            Console.Write(thisByte.ToString("X2"));
         Console.WriteLine();
         Console.ReadLine();
      }
   }
}
```

Rezultāts

Microsoft oficiāli rekomendē izstrādātājiem neuzglabāt lietotāja uzskaites datus `Web.config` datnē, pat ja tiek izmantotas jaucējfunkciju vērtības. Rekomendācijai var piekrist, tomēr reālā dzīvē daudzi izstrādātāji uzskata, ka jaucējfunkciju vērtību izmantošana ir viens no drošākajiem autentificēšanas mehānismiem, jo pēc noklusējuma ASP.NET nepārsūta `Web.config` datni pārlūkprogrammai, kas apgrūtina ļaunprātīga lietotāja piekļūšanu šai datnei.

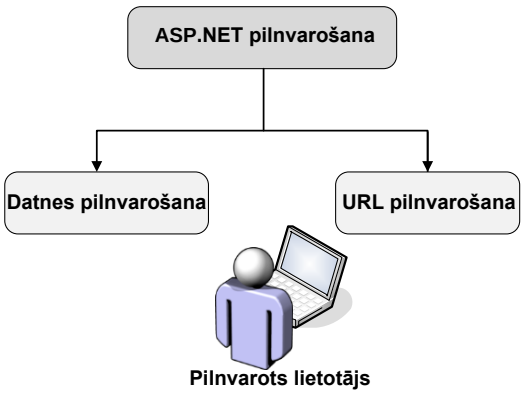
8.2. Pilnvarošanas ieviešana ASP.NET programmās

Tīmekļa lietotnes piedāvā daudzveidīgas iespējas piekļuves tiesību ierobežošanā. Ar `Web.config` datņu palīdzību var norādīt, kādiem lietotājiem ir piekļuves tiesības lietotnēm, mapēm vai pat atsevišķām datnēm (sk. 8.3. att.).

Ja autentificēšana ļauj noskaidrot lietotāja personību, tad pilnvarošana nosaka to, pie kādiem resursiem lietotājam ir tiesības piekļūt. Pirms .NET *Framework* parādīšanās lietotāju pilnvaru administrēšana tika veikta tikai ar NTFS atļauju palīdzību. Kaut arī tagad NTFS atļaujas ir tīmekļa lietotņu aizsardzības pamatā, tās ir papildinātas ar ASP.NET sastāvā iekļautajām pilnvarošanas iespējām. Tagad pilnvarošanu, tāpat kā autentificēšanu, var veikt ar `Web.config` datņu palīdzību, kas dod iespēju izpildīt pilnvarošanu jebkuram autentificēšanās tipam.

Pēc noklusējuma datne `Machine.config` satur šādu informāciju attiecībā uz pilnvarošanu:

```
<authorization>
      <allow users="*" />
</authentication>
```



8.3. att. **Tīmekļa lietotņu pilnvarošanas iespējas**

Ja šo sekciju nemaina un nepārdefinē, tad, pievienojot tieši tādu pašu sekciju lietotnes datnei `Web.config`, visi autentifikāciju izgājušie lietotāji varēs piekļūt visām ASP.NET lietotnes sastāvdaļām (uz to norāda `<allow users="*">`). Tāpat var uzskaitīt lietotāju vārdus vai izmantot simbolu „?", kas atbilst visiem neautentificētajiem lietotājiem. Piemēram, tikai lietotājiem *Peter* un *Anna* ir piekļuves tiesības lietotnei, kā parādīts dotajā koda fragmentā:

```
<authorization>
      <allow users="Peter, Anna" />
      <deny users="?" />
</authentication>
```

Apakšsekcijas `<allow>` un `<deny>` var saturēt atribūtus `users`, `roles` un `verbs`. Atribūta `users` vērtības var būt lietotāju saraksts, simbols „*" (norāda visus autentificētos un neautentificētos lietotājus) vai simbols „?" (norāda anonīmus lietotājus). Ja tiek izmantota *Windows*

autentificēšana, tad lietotāju vārdiem jāatbilst vārdiem no lokālās datu bāzes vai *Active Directory* kataloga (tie var ietvert domēna vārdu, t.i., „*Domēns\lietotājs*”).

Atribūts `roles` satur lomu sarakstu. Ja tiek izmantota *Windows* autentificēšana, tad lomas atbilst *Windows* lietotāju grupu lomām. Ja tiek izmantota autentificēšana uz formu pamata, tad lomas tiek noteiktas pašā lietotnē un tās neietekmē sistēmas administrators.

Atribūts `verbs` tiek pielietots, lai ierobežotu lietotājus un lomas HTTP pieprasījumu `HEAD`, `GET`, `POST`, `DEBUG` u.c. izmantošanā. Visiem lietotājiem būtu jādod iespēja izmantot operācijas `HEAD`, `GET` un `POST`. Piekļuvi operācijai `DEBUG` var atļaut tikai izstrādātājiem un administratoriem. Biežāk izmantojamās pārlūkprogrammu HTTP operācijas ir šādas:

- `GET` – datnes saņemšanai;
- `POST` – datu nosūtīšanai;
- `HEAD` – informācijas par datni iegūšanai.

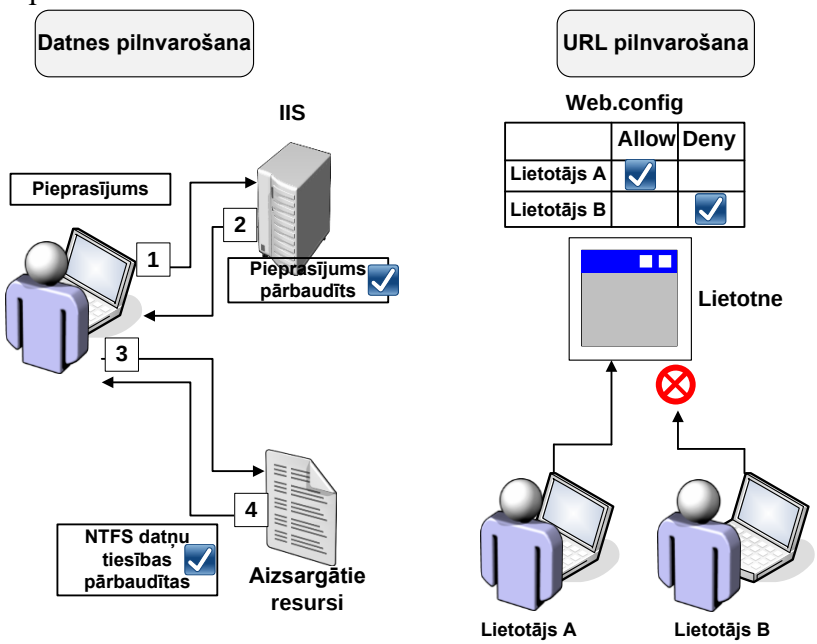
Ierobežojumus uz dažādām HTTP operācijām demonstrē šāds koda fragments:

```
<authorization>
   <allow verbs="HEAD, GET, POST, DEBUG" roles="Developers" />
   <allow verbs="HEAD, GET, POST" roles="Users" />
   <deny users="?" />
</authentication>
```

Vispārīgā gadījumā, lai novērstu nesankcionētu piekļuvi aizsargātiem resursiem, ir nepieciešams īstenot IIS pilnvarošanu ASP.NET lietotnēs (sk. 8.4. att.).

Tiek realizēti šādi pilnvarošanas mehānismi:

- datnes pilnvarošana;
- URL pilnvarošana.



8.4. att. **IIS pilnvarošana ASP.NET lietotnēs**

Datņu pilnvarošana paļaujas uz NTFS atļaujām. Lai nolasītu datni, tiek pārbaudītas lietotāja piekļuves tiesības, t. i., vai lietotāja atļaujas ASP.NET (IIS) vidē nepārsniedz NTFS atļaujas. Pēc noklusējuma ASP.NET izmanto NT AUTHORITY\NETWORK SERVICE lietotāja kontus. To var izmainīt ar personifikācijas (*impersonation*) palīdzību.

URL pilnvarošana tiek realizēta ar `Web.config` datnes `<authorization>` sekcijas palīdzību, izmantojot elementus `<allow>` un `<deny>` pēc iepriekš aprakstītajiem paņēmieniem.

Kā jau tika minēts, ASP.NET sistēmas resursu pieprasījumiem izmanto IIS uzskaites datus. Tos var pamainīt datnes `Machine.config` sekcijā `<system.web>`, izmantojot elementu `<processModel>`. Pēc noklusējuma sekcijā ir šādi parametri:

```
<processModel
   enable="true"
   userName="machine"
   password="AutoGenerate"
/>
```

Sekcijas `userName` un `password` nosaka uzskaites datus, kurus ASP.NET personificē, kad lietotāju vārdā pieprasa sistēmas resursus. Parametrs `"machine"` norāda uz to, ka ASP.NET izmantos IIS uzskaites datus. Parametrs `"AutoGenerate"` norāda, ka ASP.NET izmantos gadījuma paroli, kas ir daudz drošāk nekā rakstīt paroli datnē `Machine.config`.

Šo parametru ir pietiekami lielākajai daļai ASP.NET lietotņu. Taču atsevišķos gadījumos administratoriem nākas pielāgot ASP.NET, lai personificētu konkrēta lietotāja vai IIS anonīma lietotāja uzskaites datus. Tam ir nepieciešams izmainīt sekcijas `<identity>` parametru `impersonate` datnē `Machine.config` (lai izmaiņas attiektos uz visu serveri) vai datnē `Web.config` (lai izmaiņas attiektos uz konkrētu lietotni vai mapi).

Lai iekļautu konkrēta *Windows* lietotāja personifikāciju vai IIS uzskaites datus *IUSR_DatoraVārds* anonīmai piekļuvei, datnes `Web.config` sekcijā `<system.web>` jāpievieno šāda rinda:

```
<identity impersonate="true" userName="" password="" />
```

Ja IIS dienesti pielāgoti anonīmai izmantošanai, tad ASP.NET pieprasīs sistēmas resursus, izmantojot uzskaites datus *IUSR_DatoraVārds*. Kad lietotājs veic autentificēšanos IIS, izmantojot *Windows* uzskaites datus, ASP.NET personificē šos uzskaites datus. Personificēšanas nolūkā sekcijā `<system.web>` jāpievieno šāda rinda (atribūtos `DOMAIN`, `UserName` un `Password` jāievieto vajadzīgā lietotāja uzskaites datus):

```
<identity impersonate="true" userName="DOMAIN\UserName"
                    password="Password" />
```


Personifikācijas izmantošana `config` datnē iespaido visu ASP.NET kodu, uz kuru attiecas šīs datnes darbība. Ja lietotāji izmanto uzskaites datus ar plašu privilēģiju klāstu, piemēram, administratori, tad personifikācija var būt nevajadzīga. Var pielietot mazāko privilēģiju principu, ļaujot lietotājam veikt uzdevumus, kas prasa paaugstinātas privilēģijas, t.i., var personificēt autentificētu lietotāju tikai gadījumos, kad paaugstinātas privilēģijas patiešām ir nepieciešamas.

8.3. Tīmekļa datņu un mapju aizsardzība

Ar datnes `Web.config` sekciju palīdzību iespējams ierobežot piekļuvi atsevišķām datnēm vai veselām mapēm – sekcijā `<configuration>` jāpievieno sekcija `<location>`.


Sekcijā `<location>` jābūt savai apakšsekcijai `<system.web>`, tāpēc to nedrīkst iekļaut jau esošajā sekcijā `<system.web>`.

Lai ierobežotu piekļuvi noteiktai datnei vai mapei, sekcijā `<location>` jāpievieno parametrs `path`, kurā uzrādīts relatīvais ceļš līdz datnei vai mapei (absolūto ceļu uzrādīt nedrīkst). Pēc tam šajā sekcijā iekļauj apakšsekciju `<system.web>`, kurā uzrādītas nepieciešamās konfigurācijas ziņas par doto datni vai mapi. Piemēram, lai tiktu veikta autentificēšana uz formu pamata datnei **ListUsers.aspx** un tikai administratoram būtu piekļuves tiesības šai datnei, jāpievieno šāds koda fragments:

```
<location path="ListUsers.aspx">
     <system.web>
         <authentication mode="forms">
              <forms loginUrl="AdminLogin.aspx" protection="All"/>
         </authentication>
         <authorization>
              <allow users="admin"/>
              <deny users=”*”/>
         </authorization>
     </system.web>
</location>
```

Kad tiek izmantotas sekcijas `<location>`, tad datne un pakļautās mapes automātiski manto visus vecāku mapes parametrus, t. i., nav nepieciešams atkārtoti pārdefinēt parametrus. Taču atsevišķos gadījumos šāda mantošana var izraisīt ievainojamību, kā parādīts šajā koda piemērā:

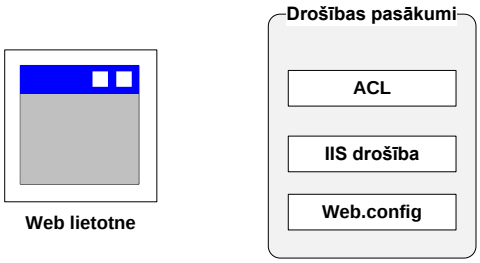
```
<configuration>
   <system.web>
      <authentication mode="Windows" />
      <authorization>
          <deny users="?" />
      </authorization>
   </system.web>

  <location path="Protected">
     <system.web>
        <authorization>
            <allow roles="CONTOSO\IT" />
        </authorization>
     </system.web>
  </location>
</configuration>
```

Šajā piemērā pastāv trīs mantošanas līmeņi. Pirmais līmenis – datne `Machine.config`, kura pēc noklusējuma satur sekciju `<allow users=”*”/>`. Otrais līmenis – pirmā sekcija `<system.web>`, kas attiecas uz visu lietotni. Tajā norādīts `<deny users="?" />`, kas aizliedz piekļuvi visiem neautentificētajiem lietotājiem. Pats par sevi otrais līmenis aizliedz piekļuvi jebkuram lietotājam, taču kombinācijā ar `Machine.config` šajā līmenī piekļuve atļauta visiem autentificētajiem lietotājiem un liegta visiem pārējiem.

Trešajā līmenī sekcija `<location>` nodrošina piekļuvi grupas CONTOSO\IT locekļiem. Taču vienlaikus šī sekcija manto parametrus `<deny users="?" />` un `<allow users=”*”/>`. Tātad pakļautajai mapei *Protected* būs spēkā tie paši parametri, kas vecāku mapei, t.i., piekļuve būs visiem autentificētajiem lietotājiem. Lai nodrošinātu piekļuvi tikai CONTOSO\IT locekļiem, nepieciešams aizliegt piekļuvi visiem lietotājiem, kam nav tiešas atļaujas, kā tas parādīts koda piemērā:

```
<location path="Protected">
  <system.web>
    <authorization>
       <allow roles="CONTOSO\IT" />
       <deny users="*" />
    </authorization>
  </system.web>
</location>
```

Tīmekļa lietotņu aizsardzības drošības pasākumi ietver IIS drošību, `Web.config` sekciju konfigurēšanu, kā arī ACL sarakstu izmantošanu (sk. 8.5. att.). Tas nozīmē, ka datņu un mapju aizsardzību var realizēt arī ar ACL atļauju palīdzību.



8.5. att. **Tīmekļa lietotņu aizsardzības drošības pasākumi**

Tiek rekomendēts tīmekļa serverī padarīt stingrākas NTFS atļaujas, izdzēšot no ACL sarakstiem grupu *Everyone*, kam pēc noklusējuma ir piekļuves atļaujas *Read & Execute*. Ja šīs grupas atļaujas tiek izdzēstas, tad ASP.NET lietotnes nestrādās, ja IIS uzskaites datiem un *IUSR_DatoraVārds* nebūs atļaujas uz *.aspx* datņu nolasīšanu. *IUSR_DatoraVārds* uzskaites datiem atļauja *Read & Execute* ir nepieciešama tikai datnēm, kuras pieprasa pārlūkprogramma, parasti tās ir datnes *.aspx* un *.asmx*.

8.3. tabulā uzskaitītas minimālās atļaujas, kam jābūt *Network Service* (IIS 6.0) uzskaites datiem. Šīs atļaujas spēkā tikai lietotnēm, kuras pirms palaišanas tiek kompilētas *.DLL* datnē (*Visual Studio* lietotnēm).

8.3.tabula

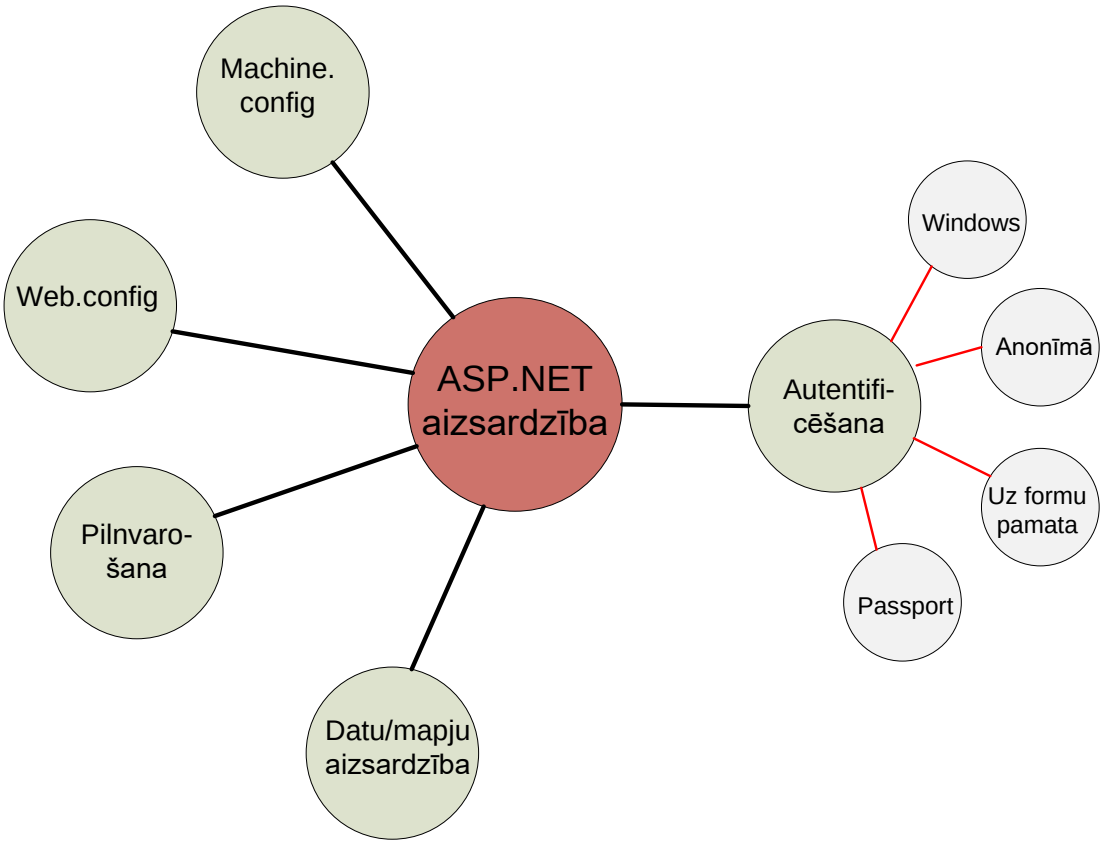
ASP.NET lietotnēm nepieciešamās minimālās atļaujas

Datnes tips	Minimālās atļaujas
asax, ascx, ashx, asmx, aspx, config, dll, rem, soap,	*Read*
cs, csproj, licx, pdb, resources, resx, vb, vbproj, vbdisco, webinfo, xsd, xsx	*No Access*

Ja uzstādīta IIS autentificēšana, tad tiek pārbaudīti visi pieprasījumi uz mapi neatkarīgi no datņu tipa. ASP.NET autentificēšana tiek veikta savādāk un attiecas tikai uz tiem pieprasījumiem, kurus IIS dienesti nodod .NET *Framework* videi. Pēc noklusējuma IIS nodod pieprasījumus par datnēm, kuru tipi ir *asax, ascx, ashx, asmx, aspx, axd, config, cs, csproj, java, jsl, licx, rem, soap, vb, vbproj, webinfo, resx, resources, vjsproj* un *vsdisco*. Tas nozīmē, ka pieprasījumi par pārējām datnēm neizies caur ASP.NET un netiks kontrolēti ar ASP.NET autentificēšanas palīdzību.

Piemēram, datne ar tipu *.pdf* nav saistīta ar ASP.NET, tāpēc IIS vēršas pie šīs datnes anonīmā lietotāja kontekstā un nosūta datni lietotājam. Lai risinātu šo problēmu, ir jāsasaista datnes tips ar IIS (ar bibliotēku *aspnet_isapi.dll*). Pēc tam lietotāja pieprasījums, kurš pieprasījis šādu datni, tiks pāradresēts uz autentificēšanās formu.

8.6. attēlā parādīts nodaļā minēto svarīgāko jēdzienu koks.



8.6. att. **Astotajā nodaļā minēto svarīgāko jēdzienu koks**

Pārbaudes testa jautājumi

1. Jūsu ASP.NET lietotne saglabā datus binārajā datnē, kas izvietota atsevišķā mapē tīmekļa serverī. Jūs gribat piešķirt tikai savai lietotnei piekļuves atļauju šai mapei. Jāveic *Windows* autentificēšana. Lietotne korekti strādā uz Jūsu datora, taču, kad tā tika pārnesta uz serveri, tika konstatēta kļūda (*permission error*), mēģinot piekļūt binārajai datnei. Jums jāgarantē, ka lietotne varēs ielādēt datus no binārās datnes, un jānovērš iespēja, ka citi lietotāji nolasa Jūsu datni. Kādas darbības nepieciešams veikt?
    a. Pievienot šādu elementu pilnvarošanas sekcijā *Web.config* datnē:
       <identity impersonate="true"/>
    b. Pievienot šādu elementu *<system.web>* sekcijā *Web.config* datnē:
       <allow users="system"/>
    c. Piešķirt lietotājam „Read” atļauju mapei, kurā atrodas šī datne
    d. *Machine.config* datnē piešķirt atribūtam *userName* no sekcijas *processModel* vērtību "system"

2. ASP.NET lietotnē tiek lietota *Windows* autentificēšana. Lietotne izvērsta Intranetā. Jūs vēlaties, lai lietotne tiktu izpildīta lietotāja drošības kontekstā. Kādas darbības nepieciešams veikt?
    a. Pievienot šādu elementu autentificēšanas sekcijā *Web*.config datnē:
       <allow users="?"/>
    b. Pievienot šādu elementu *<system.web>* sekcijā *Web.config* datnē:
       <identity impersonate="true"/>
    c. Pievienot šādu elementu autentificēšanas sekcijā *Web*.config datnē:
       <deny users="?"/>
    d. Uzrakstīt kodu *Application_AuthenticateRequest event handler*, lai konfigurētu lietotni darbam lietotāja drošības kontekstā

3. Izstrādājot ASP.NET lietotni, Jūs kādā brīdī aptverat, ka Jūs un Jūsu kolēģis esat veikuši vairākas konfigurācijas izmaiņas datnēs *Machine.config* un *Web.config*. Jūs abi lietojat atšķirīgus iestatījumus. Kurš no apgalvojumiem labāk raksturo to, kas notiks, palaižot lietotni?
    a. Lietotne nestrādās korekti tik ilgi, kamēr netiks veiktas izmaiņas vai nu datnē *Web.config*, vai datnē *Machine.config* un lietotnes iestatījumi kļūs identiski
    b. Lietotne strādās korekti, ja būs identiski iestatījumi datnēs *Web.config* un *Machine.config*
    c. Lietotne strādās korekti un izmantos datnes *Web.config* iestatījumus, ignorējot datnes *Machine.config* iestatījumus
    d. Lietotne nestrādās korekti tik ilgi, kamēr netiks nodzēsta datne *Web.config* vai arī netiks izdzēsti iestatījumi datnē *Machine.config*

4. Jūs izstrādājat ASP.NET lietotni un uzdodat *SalesOrders.aspx* kā sākuma lapu. Jūs gribat, lai lietotāji nokļūst lapā *SalesOrders.aspx*, ievadot lietotāja vārdu un paroli. Jūs izveidojat lapu *Login.aspx*, kur tiek veikta lietotāja vārda un paroles pārbaude. Jums jābūt pārliecībai, ka lietotāji veiks reģistrāciju lapā *Login.aspx*, pirms viņi nokļūs lapā *SalesOrders.aspx*. Kādas no šīm darbībām Jūs veiksiet?
    a. Autentificēšanas sekcijas datnē *Web.config* „mode" atribūtam jāiestāda vērtība „Forms". Formas elementam atribūtam „name" jāpiešķir vērtība *Login.aspx*
    b. Autentificēšanas sekcijas datnē *Web.config* „mode" atribūtam jāiestāda vērtība „Forms". Formas elementam atribūtam „LoginUrl" jāpiešķir vērtība *Login.aspx*
    c. Pilnvarošanas sekcijas datnē *Web.config* lietotāja atribūtam jāpiešķir vērtība *deny users="?"*
    d. „Credentials" sekcijas datnē *Web.config* lietotāja atribūtam jāpiešķir vērtība *deny users="?"*

5. Jūs izstrādājat lietotni elektroniskās komercijas nolūkiem, kas maksājumiem izmanto kredītkartes. Elektroniskā veikala vietne izvietota *Windows* serverī kopā ar IIS. Jums jābūt pārliecībai, ka pircēju kredītkartes dati ir drošībā transakciju laikā. Kādas no šīm darbībām Jūs veiksiet?
    a. Izmantosiet *Secure Server* drošību visai tīmekļa vietnes komunikācijai
    b. Konfigurēsiet tīmekļa vietni, lai varētu lietot *Internet Protocol Security* (IPSec) visām transakcijām
    c. Lietosiet SSL tīmekļa vietnē
    d. Neko. IIS aizsargā *Internet* tīkla komunikācijas pēc noklusējuma

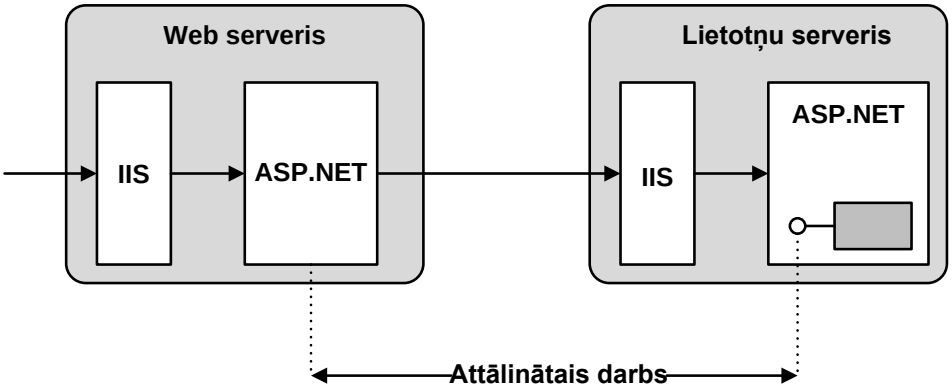
9. Attālināto .NET programmu aizsardzība

ASP.NET Web servisi ir ļoti jaudīgas tehnoloģijas, kas nodrošina nepieciešamo infrastruktūru attālas piekļuves lietotņu izstrādē – lietotne var izsaukt citas lietotnes objektus, kas atrodas dažādos domēnos, dažādos procesos vai dažādos datoros tīklā. Objektu mijiedarbības nodrošināšanai pastāv divi standarta datu pārraides kanāli: HTTP un TCP, pa kuriem informācija var tikt pārraidīta binārajā formā vai, izmantojot protokolu SOAP. Datu pārraides drošības nodrošināšanas aspektā svarīga nozīme ir lietotāja autentificēšanai.

Nodaļā īsumā apskatīti .NET attālinātās pieejas arhitektūras svarīgākie funkcionēšanas principi, demonstrēta klienta lietotnes autentificēšanas nodrošināšana ar SOAP galvenes palīdzību.

.NET *Framework* komunikācijas nolūkā uztur TCP un HTTP protokolus [28], [29]. Operētājsistēma *Windows* tiek raksturota ar starpprocesu komunikāciju IPC (*Interprocess communication*). Šādas komunikācijas metodes izmanto lietotnes uz attāliem datoriem, un tās ir ievērojami ātrākas par tradicionālajiem TCP un HTTP kanāliem, jo neizmanto portus. Arī .NET *Framework* atbalsta IPC kanālu izmantošanu.

Plašu popularitāti ieguvušie tīmekļa servisi realizācijai izmanto IIS un .ASP NET iespējas (sk. 9.1. att.).



9.1. att. **Serveru mijiedarbības shēma ar .ASP.NET izmantošanu**

Sākotnēji tīmekļa servisi tika plānoti kā līdzeklis programmu mijiedarbības organizēšanai vai attālu procedūru izsaukumam RPC (*Remote Procedure Call*). RPC būtība ir tāda, ka palaista programma var izsaukt citas programmas funkciju/procedūru, kas aktivizēta tajā pašā datorā vai citā datorā tīklā.

Tīmekļa pakalpojumi jeb servisi (*Web services*) – pakalpojumi, kas tiek sniegti tīmeklī. Tie ir starptīkla pakalpojumu speciālgadījumi, lai arī bieži vien tiek ar tiem jaukti. Tā, piemēram, e-pasts ir starptīkla, bet nav tīmekļa pakalpojums, savukārt tīmekļa saskarne e-pasta aplūkošanai ir gan tīmekļa, gan arī starptīkla pakalpojums. Vairumā gadījumu tīmekļa pakalpojumi ir pieejami, izmantojot tīmekļa pārlūku, tomēr tas nevar būt par formālu kritēriju to identificēšanai; piemēram, FTP vietnes iespējams aplūkot arī vairumā tīmekļa pārlūku, tomēr FTP nav tīmekļa pakalpojums.

Tīmekļa servisu pamatā ir protokols SOAP [30], ko formāli var dēvēt par HTTP paplašinājumu. HTTP pieprasījums parasti sastāv no divām daļām: galvenes (*header*) un ķermeņa (*body*). Galvenē tiek dota dažāda informācija par pašu pieprasījumu, bet ķermenī – informāciju saturoša ziņojuma daļa, kuru lietotājs vēlas pārsūtīt vai saņemt. SOAP paplašinājuma gadījumā nedaudz izmainīta galvene – pieprasījums tiek identificēts kā SOAP pieprasījums, bet ķermenī tiek nodots XML teksts, kas apraksta to, kas tiek nodots vai saņemts.

SOAP (*Simple Object Access Protocol*) – vienkāršais objektu piekļuves protokols. Tas ir protokols, ar kura palīdzību *Internet* tīklā un citās izkliedētās skaitļošanas vidēs var sazināties un kopīgi darboties dažādu platformu un operētājsistēmu programmatūra, sūtot cita citai XML formātā strukturētus datus. Ir specificēts, kā šāda apmaiņa var notikt ar HTTP palīdzību.

Galvene (*header*) – datu pārraidē ziņojuma sākuma daļa, kurā ir vadības informācija, ziņas par datus pārraidošo un uztverošo objektu, ziņojuma tipu un tā prioritātes līmeni.

9.1. piemērā realizēta vienkārša tīmekļa servisa autentificēšana ar SOAP galvenes palīdzību. Pirmajā solī tiek izveidots tīmekļa serviss, otrajā – klienta lietotne.

Lai izvairītos no problēmām, kas var rasties dažādu *Windows* un *Visual Studio* versiju izmantošanā, turpmākajos piemēros ieteicams palaist divus *Visual Studio* eksemplārus. Vienā tiks realizēts tīmekļa serviss, otrajā – klienta daļa.

9.1. piemērs. Vienkārša tīmekļa servisa autentificēšana ar SOAP galvenes palīdzību

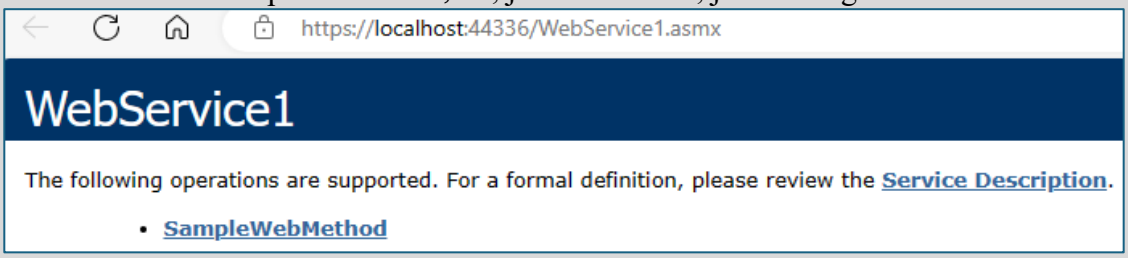
1.Pirmajā *Visual Studio* eksemplārā izveido jaunu **ASP.NET Web Application (.NET Framework)** ar nosaukumu **serviss**. Tad secīgi **Empty**, **Add / New item** un izvēlas **Web Service (ASMX)**

2.**WebService1.asmx.cs** logā ievieto šādu kodu:

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

namespace serviss
{
   [WebService(Namespace = "http://tempuri.org/")]
   [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]

   public class WebService1 : System.Web.Services.WebService
   {
      public WebService1 ()
      {
      }

     public AuthHeader SoapAuthentication;
     [SoapHeader("SoapAuthentication",Required=true)]
     [WebMethod]
public string SampleWebMethod()
{
  if (SoapAuthentication.Username == "demo" && SoapAuthentication.Password == "123")
   {
return SoapAuthentication.Username + " ir autentificēts lietotājs";
   }
  else
   {
return "Pieeja liegta lietotājam " + SoapAuthentication.Username;
   }

}
```

```
}
  public class AuthHeader : SoapHeader
  {
    public string Username;
    public string Password;
  }
}
```

3. Pārbaudes nolūkā palaiž servisu, un, ja viss normāli, jābūt līdzīgam rezultātam:

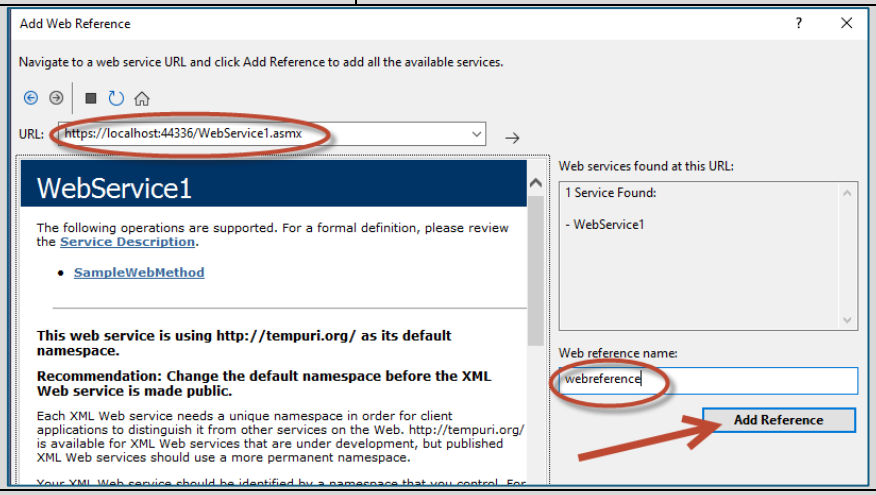


Iegaumē adresi, no kuras tiek palaists serviss – šajā konkrētajā gadījumā: **http://localhost:44336/WebService1.asmx.**

Tika izveidota metode **SampleWebMethod()**, kura izmanto SOAP galveni klienta pieprasījuma pārbaudei. Autentificēšanas rezultāts tiks atgriezts klientam.

4. Otrajā *Visual Studio* eksemplārā izveido jaunu **ASP.NET Web Application (.NET Framework)** ar nosaukumu **klients,** tad izvēlas **Empty.**

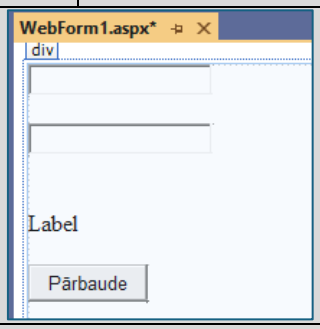
5. Izveidotajam projektam jāpievieno atsauce uz iepriekš izveidoto tīmekļa servisu: **Add Service reference**, tad **Advanced, Add Web Reference.**	- Laukā **URL** uzraksta servisa adresi (šajā gadījumā **http://localhost:44336/WebService1.asmx** un nospiež **bultiņu**. - Ja serviss veiksmīgi atrasts, laukā **Web reference name** uzraksta **webreference** un apstiprina ar pogu **Add Reference**.



6. Projektam pievieno tīmekļa formu: **Add/ New Item/ Web Form**.
Lapas **WebForm1.aspx** panelī **Design** konstruē formu ar diviem **TextBox** objektiem (lietotāja vārdam un parolei), vienu **Label** objektu (rezultātam) un vienu pogu **Button** (autentificēšanas iesākšanai servisā).

7. No **Toolbox** standarta rīku joslas uz formu pārvelk divus vadības objektus **TextBox**:	Properties	Vērtība
	TextBox1 (ID)	usertxt
	TextBox2 (ID)	passtxt

8. No **Toolbox** standarta rīku joslas uz formu pārvelk vadības objektu **Label**:	Properties Vērtība **Label1 (ID)** result
9. No **Toolbox** standarta rīku joslas uz formu pārvelk vadības objektu **Button** lietotāja datu pārbaudes veikšanai:	Properties Vērtība **Button (Text)** Pārbaude



10. Uzklikšķinot uz pogas **Pārbaude**, notikumā `Button1_Click` ievieto šādu kodu:

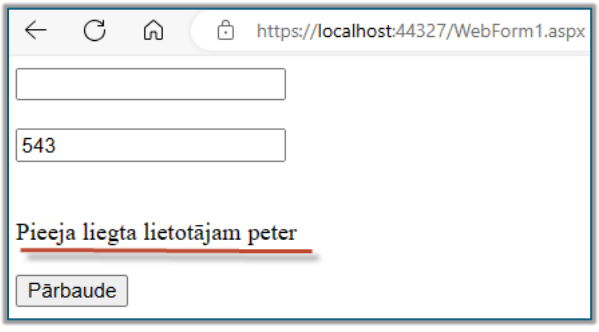
```
webreference.AuthHeader objAuth = new webreference.AuthHeader();
objAuth.Username = usertxt.Text;
objAuth.Password = passtxt.Text;

webreference.WebService1 objService = new webreference.WebService1();
objService.AuthHeaderValue = objAuth;
string str = objService.SampleWebMethod();
result.Text = str;
usertxt.Text = "";
```

Ar šo koda fragmentu tiek izveidots objekts **objAuth** klasei **AuthHeader** no tīmekļa servisa, kas aprakstīts ar vārdu telpu **webreference**. Objektam **objAuth** tiek piešķirtas vērtības **Username** un **Password**. Tad tiek veidots objekts **objService**, kurš satur tīmekļa metodi. Objektam **objAuth** tiek piešķirtas **objService** īpašības (**AuthHeaderValue**). Un, visbeidzot, tiek izsaukts serviss **SampleWebMethod**, kas arī veic lietotāja ievadīto datu autentificēšanu. Piemērā autentificētā lietotāja vārds ir **demo** un parole **123**.

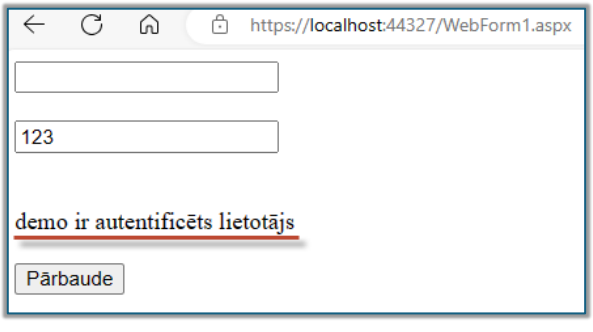
11. Palaiž klienta lietotni pārbaudei.

Rezultāts

Ievadot lietotāja vārdu **peter** un paroli **543**, iegūts šāds rezultāts:



Ievadot lietotāja vārdu **demo** un paroli **123**, iegūts šāds rezultāts:



Protams, ka ar SOAP protokolu palīdzību var veikt ne tikai lietotāju autentificēšanu. To var izmantot dažādiem mērķiem, piemēram, ar servisu palīdzību realizēt kaut kādu aprēķinu veikšanu, t. i., klienta lietotnē tiek ievadīti skaitļi, bet serviss atgriež rezultātu. 9.2. piemērā ar servisu palīdzību tiek realizēta divu skaitļu saskaitīšana (servisa un klienta lietotnes izstrāde līdzīgi kā iepriekšējā piemērā).

9.2. piemērs. Aprēķinu veikšana ar SOAP servisu palīdzību.

1. Pirmajā *Visual Studio* eksemplārā izveido jaunu **ASP.NET Web Application (.NET Framework)**.

WebService1.asmx.cs logā klasē *Service1* izdzēš rindas:

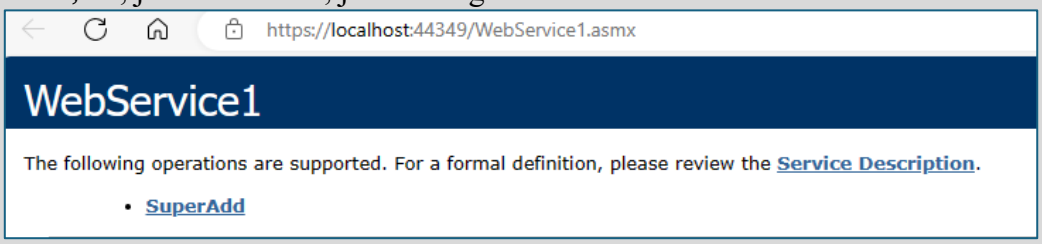
```
public string HelloWorld()
{
    return "Hello World";
}
```

To vietā ievieto šādu metodi:

```
public int SuperAdd(int value1, int value2)
{
 return value1+value2;
}
```

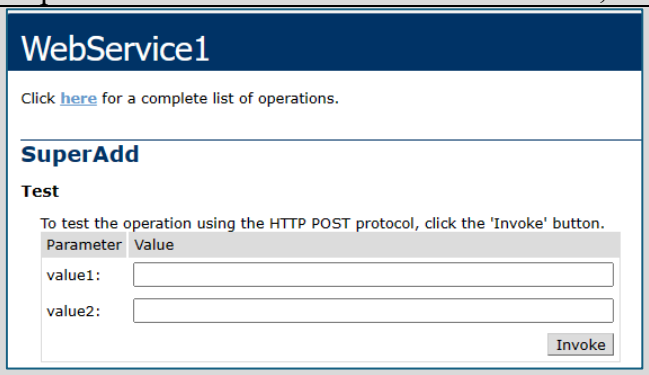
Koda rinda **[Web Method]** norāda uz to, ka tā ir servisa metode.

Palaiž servisu, un, ja viss normāli, jābūt līdzīgam rezultātam:

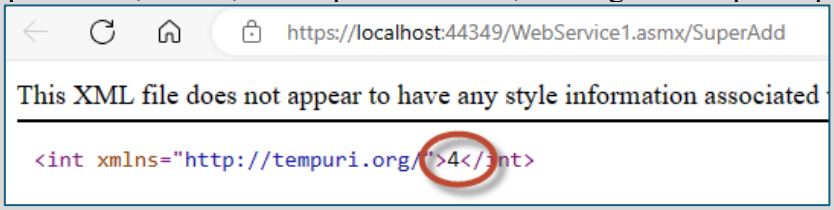


Parādījās servisa apraksts WSDL formātā, ko var apskatīt norādē **Service Description**. Tur aprakstītas servisā izmantojamās metodes, tostarp arī **SuperAdd**.
(Atkal iegaumē adresi, kas būs vajadzīga klienta lietotnē!)

Noklikšķinot uz **SuperAdd**, parādās metodes apraksts, kā vērsties pie tās caur SOAP un HTTP-POST, kā arī piedāvājums pārbaudīt metodi darbībā – divi teksta lauki, kuros var ievadīt skaitļus.



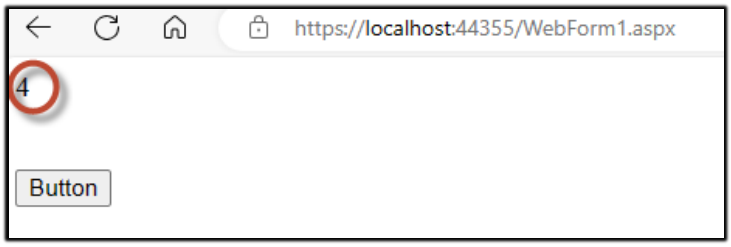
Ievadot skaitļus, piemēram, 2 un 2, un nospiežot **Invoke**, tiek iegūts šāds pārlūkprogrammas logs:



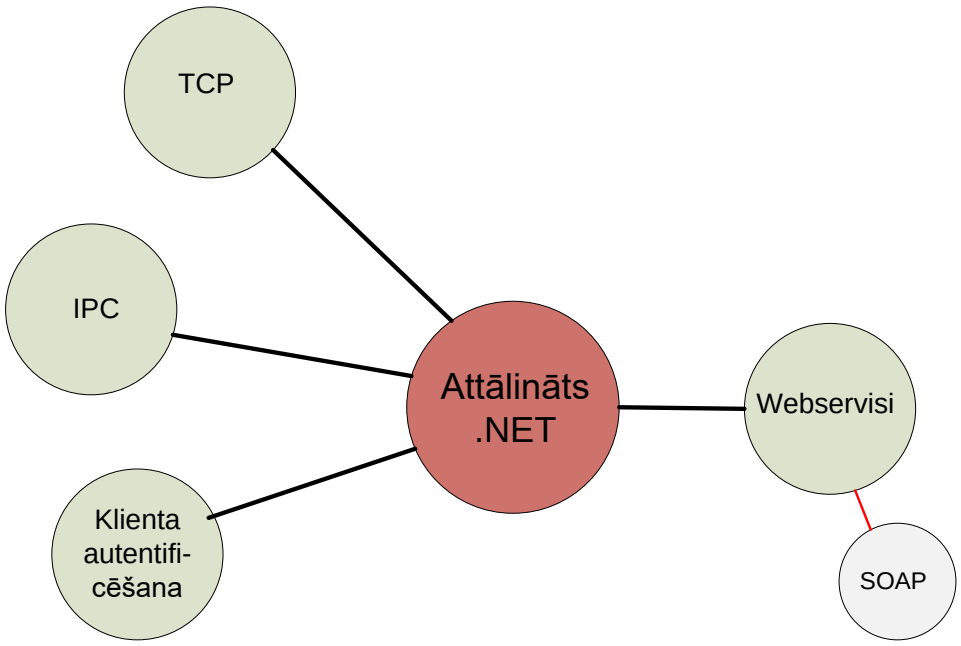
Tā arī būs servisa atbilde.

2. Otrajā *Visual Studio* eksemplārā izveido jaunu **ASP.NET Web Application (.NET Framework)**.
Līdzīgi kā iepriekšējā piemēra 5. punktā izveidotajam projektam jāpievieno atsauce uz iepriekš izveidoto tīmekļa servisu: **Add Web reference** (nosaukums lai paliek **localhost**).
Tīmekļa formā konstruē formu ar vienu **Label** objektu (rezultātam) un vienu pogu **Button** (skaitļošanas uzsākšanai servisā).
Uzklikšķinot uz pogas **Button**, notikumā `Button1_Click` ievieto šādu kodu: **localhost.WebService1 myservice = new localhost.WebService1();** **Label1.Text=Convert.ToString(myservice.SuperAdd(2, 2));**
Palaiž lietotni.

Rezultāts



9.2. attēlā parādīts nodaļā minēto svarīgāko jēdzienu koks.



9.2. att. **Devītajā nodaļā minēto svarīgāko jēdzienu koks**

Pārbaudes testa jautājumi

1. Tiek izstrādāta lietotne attālam darbam caur TCP kanālu. Tā kā šis kanāls nav drošs, ir svarīga klienta autentificēšanas procedūra. Kas no minētā ir piemērotāks attāla klienta autentificēšanai?
    a. SSL
    b. SOAP *Header* autentificēšana
    c. WS-Routing
    d. *Windows* autentificēšana

2. Tiek izstrādāta e-apmācība ar .NET attālinātajiem līdzekļiem. Apmācības moduļi izvietoti serverī .MPEG formātā. Studentiem, kas vēlas veikt apmācību, nepieciešams pievienoties serverim un tad izvēlēties nepieciešamo moduli. Video datni nepieciešams straumēt uz studenta datoru. Students pievienojas serverim, autentificējoties katras sesijas sākumā, un starp sesijām notiek viņa pilnvaru pārbaude. Kādu iespēju no minētajām jāizvēlas šādai lietotnei?
    a. IIS
    b. Console
    c. Window service
    d. Apache
    e. Windows service

3. Ja komunikācijas nodibināšanai starp serveri un klientu tiek izmantots TCP kanāls, tad kādu formatēšanas veidu vajadzētu izvēlēties, ja komunikācija paredzēta ar „nepazīstamu" tīklu?
    a. *Binary* formatēšana
    b. SOAP formatēšana
    c. *Custom* formatēšana
    d. *Zip* formatēšana

4. Kāda ir visatbilstošākā metode klientu autentificēšanai?
    a. Ciparsertifikāts
    b. *Windows* autentificēšana
    c. Drošības marķieris
    d. *Custom* autentificēšana

10. Drošības testēšana

Drošības testēšanas veids – moduļu testēšana – kļūst par izplatītu paņēmienu koda kvalitātes paaugstināšanai, novēršot potenciālās kļūdas projekta izstrādes sākuma stadijā. Moduļu testēšana ir automatizētās testēšanas paņēmiens, kuru izstrādātāji var pielietot problēmu risināšanā līdz koda nodošanai testētājiem, kas ļauj samazināt testēšanas ciklu skaitu.

Nodaļā apskatīta testēšanas stratēģija, moduļu automatizētās testēšanas metodoloģija, kā arī tiek doti ieteikumi lietotņu drošības pārbaudei uz ievainojamību un rekomendācijas ārējo komponentu izsaukšanas riska minimizēšanai.

10.1. Testēšanas stratēģija un automatizēta moduļu testēšana

Testēšana (*testing*) – programmatūras un aparatūras darbības pārbaude, izmantojot testdatus. Testēšanas mērķis var būt defekta atklāšana, tā atrašanās vietas lokalizēšana vai arī testējamā objekta dinamisko parametru (piemēram, ātrdarbības) noskaidrošana.

Tests (*test*) – speciāli izveidota datu kopa, ko izmanto, lai pārbaudītu datora vai sistēmas darbības atbilstību paredzētajām prasībām. Kā testu var izmantot datus, kas iegūti sistēmas iepriekšējās darbības rezultātā vai arī mākslīgi radīti šīs pārbaudes veikšanai.

Testpiemērs (*test case*) – konkrētam mērķim izstrādāta testa ievaddatu, izpildes noteikumu un sagaidāmo rezultātu kopa, lai dotu iespēju pārbaudīt testējamā objekta atbilstību iepriekš definētajām prasībām.

Programmnodrošinājuma testēšana ir process, kas palīdz noteikt izstrādājamā programmu produkta kvalitāti, pilnīgumu un korektumu. Tiek apgalvots, ka testēšanas pasākumi aizņem ap 40 % no visa programmprodukta izstrādes procesa. Tomēr nekāda testēšana nevar dot absolūtu garantiju par programmprodukta darbspējas korektumu reālā darbībā.

Pastāv vairākas pazīmes, pēc kurām tiek pieņemts veikt testēšanas veidu klasifikāciju. Vispārpieņemtā klasifikācija ir šāda:

- pēc testēšanas objekta:
  - funkcionālā testēšana (*functional testing*);
  - slodzes testēšana:
    - veiktspējas testēšana (*perfomance/stress testing*);
    - stabilitātes testēšana (*stability/load testing*);
  - lietojamības testēšana (*usability testing*);
  - lietotāja saskarnes testēšana (*UI testing*);
  - drošības testēšana (*security testing*);
  - lokalizācijas testēšana (*localization testing*);
  - savietojamības testēšana (*compatibility testing*).
- pēc zināšanām par sistēmu:
  - melnās kastes testēšana (*black box testing*);
  - baltās kastes testēšana (*white box testing*);
  - pelēkās kastes testēšana (*gray box testing*);
- pēc automatizācijas pakāpes:
  - manuāla testēšana (*manual testing*);
  - automatizēta testēšana (*automated testing*);
  - pusautomatizēta testēšana (*semiautomated testing*);

- pēc komponentu izolētības pakāpes:
  - moduļu testēšana (*component/unit testing*);
  - integrācijtestēšana (*integration testing*);
  - sistēmtestēšana (*system/end-to-end testing*);
- pēc testēšanas veikšanas laika:
  - *Alfa* testēšana – koda funkcionālā aizpildīšana (*alpha testing*);
  - "dūmu testēšana" (*smoke testing*);
  - regresīvā testēšana (*regression testing*);
  - akcepttestēšana (*acceptance testing*);
  - *Beta* testēšana – kļūdu novēršanas stadija (*beta testing*);
- pēc scenāriju kvalitātes:
  - pozitīva testēšana (*positive testing*);
  - negatīva testēšana (*negative testing*);
- pēc gatavības pakāpes testēšanai:
  - testēšana pēc dokumentācijas (*formal testing*);
  - intuitīvā testēšana (*ad hoc testing*).

Nodaļā tiks apskatīti tikai daži testēšanas paņēmieni, kas minēti *Microsoft* oficiālajā kursā.

Vispārīgā gadījumā drošības testēšanas stratēģija tiek pielietota sistēmas drošības pārbaudei un balstās uz trim pamatprincipiem.

- Konfidencialitāte (*confidentiality*) – noteiktu resursu vai informācijas slēpšana. Ar konfidencialitāti var saprast noteiktas kategorijas lietotāju ierobežošanu piekļuvei resursiem jeb, citiem vārdiem sakot, pie kādiem nosacījumiem lietotājs ir pilnvarots iegūt tiesības piekļuvei dotajiem resursiem.
- Integritāte (*integrity*). Pastāv divi kritēriji integritātes jēdziena noteikšanai:
  - uzticība. Pieņem, ka resursu varēs izmainīt tikai noteikta lietotāju grupa noteiktā veidā;
  - bojājumi un atjaunošana. Gadījumā, ja dati tiek bojāti vai nepareizi izmainīti pilnvarotu vai nepilnvarotu lietotāju darbības rezultātā, ir jādefinē, cik svarīga ir datu atjaunošanas procedūra.
- Izmantojamība (*availability*) – nosaka prasības par to, ka resursiem jābūt pieejamiem pilnvarotiem lietotājiem, iekšējiem objektiem vai iekārtām. Jo kritiskāks resurss, jo augstākam jābūt pieejamības līmenim.

Drošības testēšana var iekļaut šādus aspektus:

- piekļuves kontroles testēšana – ļauj atklāt defektus, kuru rezultātā lietotājs var iegūt nesankcionētu piekļuvi lietotnes objektiem un funkcijām;
- lietotāju pilnvarošanas testēšana – ļauj atklāt defektus, kas saistīti ar atsevišķu lietotāju vai lietotāju grupu pilnvarošanu un to identificēšanu;
- ievades testēšana – izmanto datu validēšanai, piemēram, servera pusē, pirms tie tiek izmantoti lietotnē;
- datu šifrēšanas drošības testēšana – ļauj atklāt defektus, kas saistīti ar datu šifrēšanu un atšifrēšanu, ciparparakstu izmantošanu;
- kļūdu apstrādes procedūru testēšana – satur tādu aspektu pārbaudi kā koda fragmenta izvade ekrānā kļūdu gadījumos, kļūdu ietekme uz visas lietotnes darbību kopumā u. c.

Testēšanas profesionāļu terminoloģijā frāzes „baltās un melnās kastes testēšana" attiecas uz to, vai testu izstrādātājam ir pieejams programmas kods vai arī testēšana tiek veikta, balstoties uz lietotāja saskarni. Baltās kastes testēšanas gadījumā testa izstrādātājam ir pieejams programmas kods, un viņš var rakstīt testēšanas kodu, kas saistīts ar testējamās programmatūras bibliotēkām. Šāda testēšana ļauj veikt kļūdu lokalizāciju, drošības analīzi un stabilitāti, tādējādi būtiski paaugstinot

sistēmas kvalitāti kopumā. Tas ir tipiski moduļu testēšanas gadījumā, kurā tiek testētas atsevišķas sistēmas sastāvdaļas.

Melnās kastes testēšanas gadījumā programmas kods nav pieejams, un testēšana tiek veikta, pamatojoties uz lietotāja saskarni un programmatūras prasību specifikāciju.

Pirms lietotnes testēšanas ir jāizstrādā drošības testēšanas plāns. Testēšanas plāns ir dokuments, kas identificē testējamās lietotnes aspektus un nosaka, kā to testēt. Testēšanas plāns iever katra testējamās lietotnes komponenta aprakstu un drošības pieņēmumus (sk. 10.1. tabulu).

10.1. tabula

Drošības testēšanas plāna sastāvdaļas

Elements	Apraksts
Projekta plāns (*Project plan*)	Laika sadalījuma un budžeta, personāla un citu resursu vajadzību apraksts
Risku novērtējums (*Risk assessment*)	Esošo risku novērtējums (DREAD, STRIDE)
Testēšanas stratēģija (*Testing strategy*)	Dažādu testēšanas tehnoloģiju un testpiemēru izvēle
Vide (*Environmemt*)	Izpildes vides specificēšana (izvēlētā platforma un klienta vide)

Drošības testēšanas plāns satur šādus komponentus:

- lietotnes izmantošanas scenāriji – šis komponents specificē visus scenārijus, kādos lietotne tiks testēta. Tie iever labāko gadījumu (*best-case*) un sliktāko gadījumu (*worst-case*) scenārijus;
- testi, kas tiks izpildīti, – dažādu izpildāmo testu saraksts prioritārā secībā;
- testēšanas grafiks – plānotais testu laiks un secība;
- izpildes vide – specificē vidi, kurā tiks izpildīti testi;
- testpiemēri – vissvarīgākais testēšanas plāna komponents. Testpiemēri apraksta testējamās funkcijas, izmantojamos datus un sagaidāmo rezultātu.

Drošības testēšanas plāna izstrādes procesā nepieciešams paredzēt šādus uzdevumus:

- identificēt lietotnes struktūru un sadalīt lietotni atsevišķos komponentos;
- identificēt testējamo komponentu saskarnes;
- sakārtot saskarnes apdraudējumu prioritāšu secībā;
- izstrādāt testpiemērus.

Testpiemēri tiek rakstīti saskarņu līmenī, tāpēc ir svarīgi saprast, kādā veidā tiek traktēta saskarne. Lietotne var saturēt dažādus komponentus: izpildāmās datnes *(.dll* un *.exe*), ASP.NET lapas, skriptu datnes (*VBScript, Microsoft Jscript, Perl*), HTML lapas, COM+, XML tīmekļa servisus, datu bāzes u. c. Katrs no tiem var tikt apdraudēts, tāpēc katram no tiem jāpiemēro savi aizsardzības pasākumi.

Saka, ka katrs no šiem lietotnes komponentiem izmanto savu saskarni. Par saskarni var kalpot COM (*Component Object Model*) metodes un īpašības, SOAP pieprasījumi, aparatūras ierīces, datnes utt. Piemēram, ASP.NET lapai var būt šādas saskarnes: *Default.aspx, Login.aspx, Global.aspx, Default.aspx.vb, Login.aspx.vb, Global.aspx.vb, LoginDenied.htm*. Konfigurācijas datnei var būt šādas saskarnes: *Web.config, AssemblyInfo.vb*.

Pēc lietotņu saskarņu identificēšanas tās jāsakārto drošības apdraudējumu bīstamības ziņā, t. i., jāievieš prioritātes – kuras no saskarnēm ir vairāk apdraudētas, kuras potenciāli mazāk. Apdraudējumu identificēšanai var izmantot STRIDE modeli.

Praksē saskarnes raksturīgo pazīmju drošības risku subjektīvai novērtēšanai var izmantot skaitļus diapazonā no 0 līdz 5 (5 – vislielākais apdraudējums, 0 – vismazākais). Pēc tam katrai

saskarnei punktus sasummē, un saskarne ar lielāko punktu skaitu atradīsies prioritāšu rindas sākumā, t. i., šo saskarni vajadzētu testēt pirmo. 10.2. tabulā piemēra nolūkā parādītas tīmekļa lietotnes saskarnes raksturīgās pazīmes un tām piešķirtie punkti.

10.2. tabula

Saskarnes raksturīgāko pazīmju saraksts

Saskarnes raksturīgā pazīme	Piešķirtie punkti	Punktu piešķiršanas iemesls
Saskarne akceptē patvaļīga garuma rindas	1	Pazīme nav kritiska lietotnes funkcionalitātei
Saskarnei nav ACL	4	Pazīme ir kritiska. Augsts drošības risks
Saskarne ir uzstādīta ar iestatījumiem pēc noklusējuma	1	Pazīme nav kritiska lietotnes funkcionalitātei
Saskarnei nav nepieciešams autentificēt lietotājus	2	Pazīmei nav augsts drošības risks
Saskarne darbībā izmanto serveri	4	Pazīme ir kritiska. Augsts drošības risks

Pēc saskarņu sakārtošanas prioritāšu režīmā var sākt izstrādāt testpiemērus. Testpiemēra struktūra parādīta 10.3. tabulā.

10.3. tabula

Testpiemēra struktūras paraugs

Elements	Piemērs
Testa nosacījums	SQL injekcija pieteikšanās (*logon*) laikā
Prioritāte	Augsta
Izpildes detaļas	Injicēt SQL lietotāja vārda laukā pieteikšanās formā.
Sagaidāmais rezultāts	Pieteikšanās atteikums

Lietotņu drošības testēšanas nolūkā izmanto divas testēšanas metodes:

1) manuālo testēšanu, kad testa soļus un novērtējumu veic testētājs;
2) automatizēto testēšanu.

Manuālo testēšanu lieto:

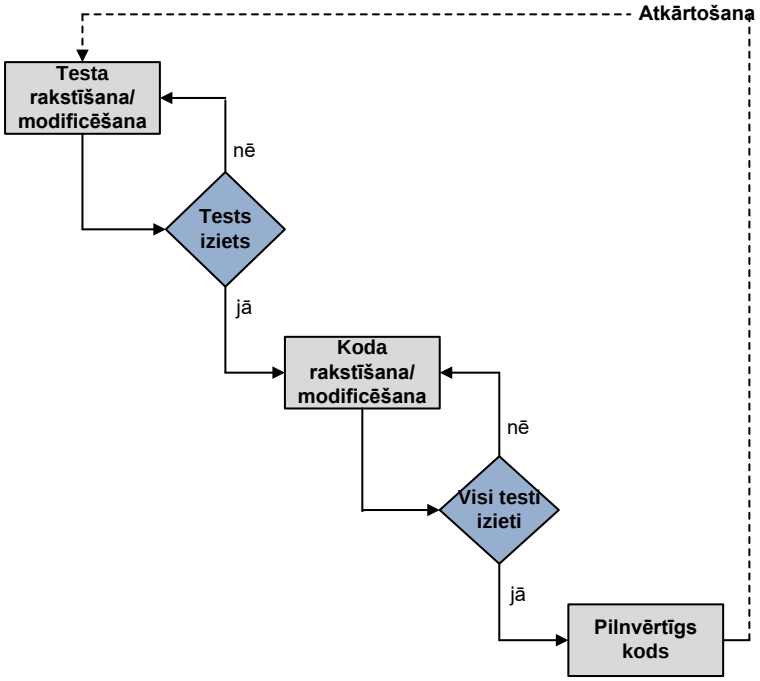
- scenārijos, kurus grūti automatizēt;
- jaunu kļūdu atklāšanai;
- lietotāja saskarnes pārbaudei;
- lietotnes nepārtrauktā (*end-to-end*) pārbaudē.

Automatizēto testēšanu parasti pielieto moduļu testēšanā. Moduļu testēšana (*Unit Tests*) ir metode, ko izmanto izstrādātāji lietotņu komponentu automatizētai testēšanai pēc dažādu izmaiņu veikšanas kodā. Būtiskākais ir tas, ka testēšanu veic izstrādātājs, t. i., testēšana tiek izdarīta līdz tam, kad lietotne tiek nodota testētāju rīcībā. Moduļu testēšana padara izstrādes procesu mērķtiecīgāku, kas ļauj izstrādātājiem atrast kļūdas savā kodā.

Moduļu testi ir moduļi, kas pārbauda citus moduļus. Moduļa tests izsauc moduli, kam jāatgriež prognozējamais rezultāts un jāanalizē iegūtais rezultāts. Ja rezultātā iegūtas nepieļaujamas vērtības, tad tests nav iziets. Piemēram, moduļa tests var izsaukt metodi `AddIntegers` ar vērtībām 2 un 3. Ja šī metode atgriež vērtību, kas atšķirīga no vērtības 5, tā neiziet testu. Ar moduļu testu palīdzību var tāpat pārbaudīt, vai metode korekti apstrādā kļūdas, vai tiek apstrādātas izņēmuma situācijas utt.

Ar testiem vadāmā izstrāde (*Test-Driven Development*) ir metodoloģija, pēc kuras izstrādātāji veido testu moduļus pirms pašiem moduļiem (sk. 10.1. att.). Var likties dīvaini, ka no sākuma jāizstrādā kods, piemēram, vēl neeksistējošas klases pārbaudei, tomēr šai metodoloģijai piemīt divas priekšrocības:

1) ja izstrādā moduļu testus katrai būtiskai lietotnes funkcijai, tad neiespējami aizmirst realizēt kādu funkciju, jo citādi kods neizies moduļu testu pārbaudi;
2) moduļu testi norāda uz aizsardzības funkcijām un pretlīdzekļiem, kas vēl nav realizēti laika trūkuma dēļ vai prioritāšu izmaiņas rezultātā.



10.1. att. **Ar testiem vadāmās izstrādes darbības princips**

Šāda metodoloģija var dārgi izmaksāt: metožu pārbaudes scenāriju izstrāde aizņem 25–50 % laika attiecībā pret pašu metožu izstrādi. Taču ar laiku šie pūliņi atmaksājas. Moduļu testi var tikt izmantoti visā lietotnes dzīves cikla laikā un fiksēt kļūdas, ko ienes labojumi vai jaunas funkcijas.

Drošības kontekstā moduļu testus var izmantot, lai atklātu nejauši atslēgtas drošības nodrošināšanas funkcijas izmaiņu ieviešanas rezultātā. Piemēram, ja tiek uzlikti komentāri koda rindai, kas attīra ievades datus, tad moduļa tests var atklāt šīs izmaiņas un atgādināt par svarīgas funkcijas neesamību.

Apsteidzošās moduļu testēšanas koncepcija satur trīs soļus:

1) jāizveido metodes katras būtiskas lietotnes funkcijas drošības pārbaudei, iekļaujot šādas metodes:
   - kas uzsāk darbību pieļaujamu datu nodošanas laikā;
   - kas uzsāk darbību nepieļaujamu datu nodošanas laikā;
   - kas paziņo par kļūdām;
   - kuru izpilde nav iespējama, ja lietotājam nav pietiekamu privilēģiju;
2) jāizveido vai jāpapildina lietotnes metodes;
3) jāveic testēšana, lai pārliecinātos, ka metodes veiksmīgi iziet drošības testus. Nepieciešamības gadījumā jāveic izmaiņas.

Moduļu testēšanai var izmantot dažādus neatkarīgu izstrādātāju rīkus, piemēram, bezmaksas utilītu *NUnit* [31], kas paredzēta tieši .NET *Framework* un piedāvā klases moduļu testu izstrādei.

Turpmākajā izklāstā tiks demonstrētas *Visual Studio* iespējas moduļu testu veidošanā un moduļu testēšanā. Moduļu testēšanā izmanto 3 testa metodes atribūtus:

- `[TestInitialize]` – dažādu testa atribūtu iestatījumi pirms testa metodes izsaukšanas;
- `[TestCleanup]` – veic parametru attīrīšanu pēc testa izpildes;
- `[TestMethod]` – satur testpiemēra darbības loģiku.

Testējamo metožu pārbaudei tiek izmantota klase `Assert`, kura izmanto vairākas metodes: `AreEqual`, `AreNotEqual`, `AreSame`, `Equals`, `Fail` u. c. Tieši šī klase arī veic moduļa testēšanu pēc shēmas, kāds rezultāts ir sagaidāms (*expected*) un kāds ir iegūts (*actual*).

Moduļu testēšanas demonstrācijas nolūkā tiek formulēts šāds vienkāršs uzdevums – jāizstrādā un jānotestē klase, kura veic divu skaitļu saskaitīšanu un reizināšanu.

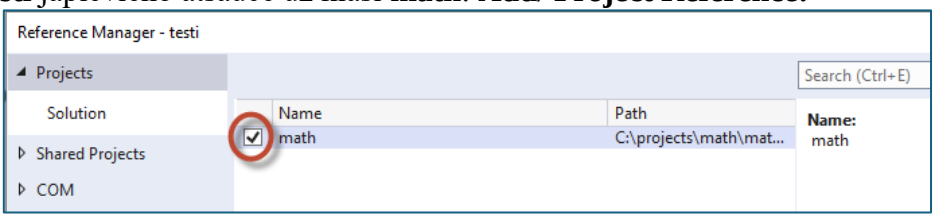
10.1. piemērs. Klases testēšana

1. *Visual Studio* vidē sāk jaunu projektu **Class Library** un piešķir tam nosaukumu **math**.
2. **Class1.cs** logā ievieto šādu kodu:

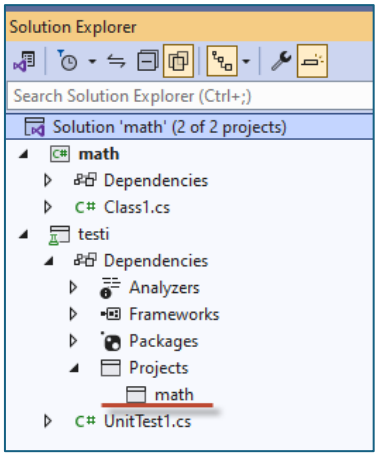
```
using System;
using System.Text;
namespace math
{
        public class Class1
        {
                public int AddNumbers(int a1, int b1)
                {
                        return a1 + b1;
                }
                public int MultiplyNumbers(int a1, int b1)
                {
                        return a1 * b1;
                }
        }
}
```

3. Komilē to: **Build / Build Solution**.

4. Pievieno jaunu projektu: **Add/ New Project/ NUnit Test Project** ar nosaukumu **testi**.
5. Testiem **testi** jāpievieno atsauce uz klasi **math**: **Add/ Project Reference:**



Tagad **Solution Explorer** logam jāizskatās šādi:

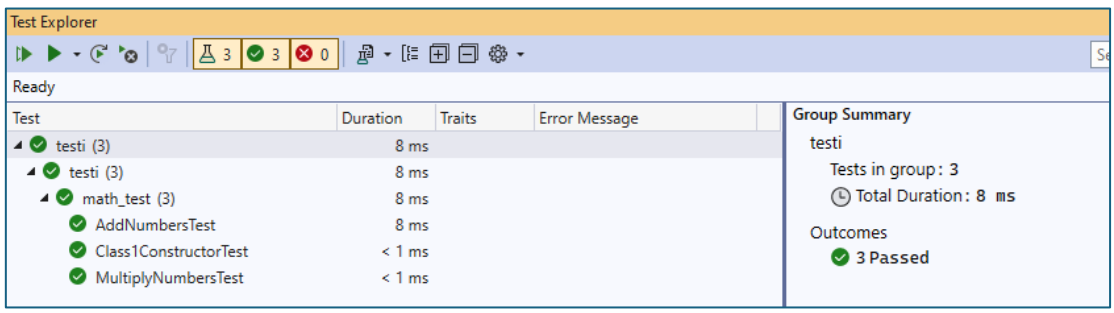


6. Pievieno testpiemērus – **UnitTest1.cs** ievieto šādu kodu:

```
using math;
using NUnit.Framework;
```

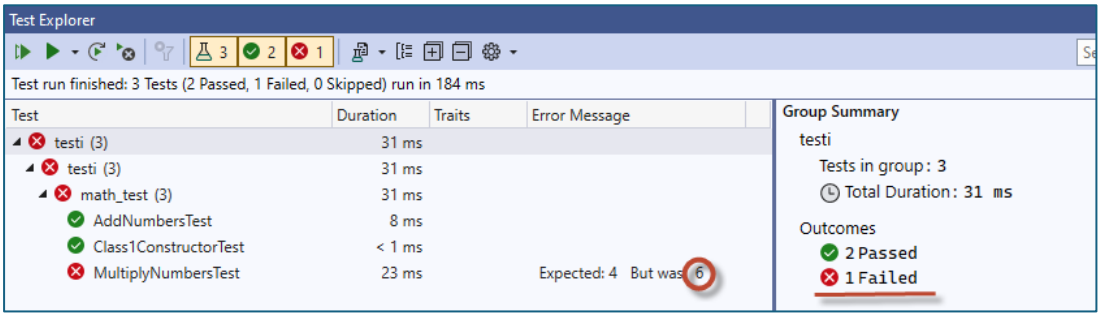
```
namespace testi
{
  public class math_test
  {
     public math_test()
     {

     }

     // Konstruktora tests
     [Test]
     public void Class1ConstructorTest()
     {
        Class1 target = new Class1();
     }
     // Saskaitīšanas tests
     [Test]
     public void AddNumbersTest()
     {
        Class1 target = new Class1();
        int a1 = 10;
        int b1 = 15;
        int expected = 25;      //sagaidāmā vērtība 10+15=25
        int actual;
        actual = target.AddNumbers(a1, b1);
        Assert.AreEqual(expected, actual);
     }
     // Reizināšanas tests
     [Test]
     public void MultiplyNumbersTest()
     {
        Class1 target = new Class1();
        int a1 = 2;
        int b1 = 2;
        int expected = 4;      // sagaidāmā vērtība 2*2=4
        int actual;
        actual = target.MultiplyNumbers(a1, b1);
        Assert.AreEqual(expected, actual);
     }
  }
}
```

7. Palaiž testus: **Test / Run All Tests**. Ja viss kārtībā, vajadzētu parādīties šādam rezultātu logam:



Kā redzams, šajā vienkāršotajā piemērā visi testi ir veiksmīgi izpildīti, kas dod pārliecību, ka projektējamās klases funkcijas korekti strādās.

Pārbaudes nolūkā reizināšanas testā pamaina mainīgā **a1** vērtību uz 3 (2*3=6):



Tests nav izpildīts.

Datorsistēmu drošības stāvokļa novērtēšanai *Microsoft* iesaka izmantot tā saucamās drošības testēšanas platformas, t. i., specializētus rīkus, kas ļauj identificēt datorsistēmas vai lietotņu problēmas un ievainojamību. Daži no šādiem agrāk izmantojamiem bezmaksas rīkiem aprakstīti 10.4. tabulā.

10.4. tabula

Drošības testēšanas rīki un to iespējas

Rīka nosaukums	Sniegtās iespējas
Microsoft Baseline Security Analyzer (MBSA) [32]	*Microsoft* pamatlīnijas drošības analizētājs (MBSA) ir rīks, kas paredzēts IT profesionāļiem, kuri palīdz mazajiem un vidējiem uzņēmumiem noteikt to drošības stāvokli saskaņā ar *Microsoft* drošības ieteikumiem un piedāvā noteiktus koriģēšanas norādījumus. MBSA datorsistēmās nosaka vispārējas drošības konfigurācijas kļūdas un trūkstošos drošības atjauninājumus. Tas pārbauda *Windows* vājās vietas un piedāvā atrast trūkstošos drošības ielāpus. Programmu var vadīt no vizuālās saskarnes un dot komandas no komandrindas, kas ļaus pārbaudīt ne tikai savu datoru, bet arī visus iekšējā tīkla datorus, ja būs attiecīgās tiesības. Uzlabojumi, kuri ir inkorporēti jaunajākajās MBSA versijās, ietver alternatīvu datņu versiju atbalstu, ugunsmūra, automātisko jauninājumu un IE zonu konfigurācijas pārbaudi
IIS Lockdown Tool	Rīks var tikt izmantots IIS drošības nodrošināšanai un palīdz konfigurēt IIS, atslēdzot neizmantojamos IIS servisus un dienestus, piemēram, FTP un SNTP. Šādā režīmā strādā tikai tie servisi, kurus piedāvā administrators, tādējādi tiek samazinātas draudu iespējas no ļaunprātīgu lietotāju puses
URLscan Security Tool	Rīks bloķē noteiktus HTTP pieprasījumus, lai ierobežotu IIS izsaukumus

Moduļu testēšanas iespējas ir diezgan ierobežotas, Piemēram, moduļu testēšana neļauj atklāt tādus trūkumus kā nepieciešamo atļauju nedeklarēšana vai konfidenciālo datu pārraide bez šifrēšanas. Pastāv virkne rekomendāciju, ko izstrādātājam vajadzētu ņemt vērā, pirms lietotne tiek nodota testētāju komandai [33].

1. Jānovērtē uzbrukuma apjoms. Ir jānosaka visi sākuma punkti, kuros dati tiešā veidā tiek ievadīti lietotnē. Katru šādu sākuma punktu jāpārbauda uz ievainojamības iespējām.
2. Jānosaka varbūtējie uzbrukuma mērķi, kā arī potenciālie ļaunprātīgie lietotāji, kurus šie mērķi varētu ieinteresēt. Jāpatur prātā, ka lietotne var tikt pakļauta visu iespējamo tipu uzbrukumiem, ja vien nav speciālas aizsardzības. Piemēram, *Windows Forms* lietotnes parasti nav pakļautas CSS (*Cross-Site Scripting*) uzbrukumiem, bet lietotnēm uz .NET *Framework* bāzes ir laba aizsardzība pret uzbrukumiem, kas izmanto bufera pārpildīšanos.

3. Jāizvērtē komunikācijas aizsardzība – vai ļaunprātīgam lietotājam ir iespēja pārtvert tīkla trafiku un iegūt konfidenciālu informāciju.
4. Jāpārliecinās, ka lietotne pēc noklusējuma izmanto visdrošākos iestatījumus. Ja pēc lietotnes ieviešanas aizsardzības uzlabošanai nepieciešams veikt papildu konfigurēšanu, tad aizsardzības pēc noklusējuma princips netiek izpildīts.
5. Jāpārliecinās, ka izpildās mazāko privilēģiju izmantošanas princips. Piemēram, ir jāpārbauda, vai tiek ierobežotas tiesības piekļuvei datu bāzēm un vai klienta lietotne strādās standarta lietotāja režīmā.
6. Jāpārbauda, vai servera komponenti ir noturīgi pret DoS uzbrukumiem.

Izstrādātājam, kā likums, nepietiek laika savas lietotnes testēšanai. Atsevišķos gadījumos var izmantot bezmaksas līdzekļus, kas ļauj izstrādātājam veikt sākotnējo testēšanu maksimāli efektīvi, vienkāršojot ievainojamības noteikšanu savā lietotnē.

10.2. Ārējo komponentu izsaukšanas riska minimizēšana

.NET *Framework* nodrošina visdrošāko lietotņu izstrādes vidi, ko jebkad piedāvājis *Microsoft*. Izstrādājot projektus, var un vajag izmantot .NET *Framework* drošības nodrošināšanas funkcijas. Tajā pašā laikā ir daudz izstrādņu, kas tika veidotas tagad jau novecojušām platformām tādām kā Win32, COM vai ActiveX. .NET *Framework* ļauj mijiedarboties ar šo platformu komponentiem, taču tā rezultātā var tikt zaudētas aizsardzības priekšrocības.

Nepārvaldītu komponentu izsaukšanas risks ir ļoti liels, jo tie strādā ārpus izpildes vides un tādējādi uz tiem neattiecas koda piekļuves (CAS) pasākumi. Turklāt šādi komponenti neuztur stingros vārdus, tāpēc ļaunprātīgam lietotājam ir iespējas nomainīt šo komponentu ar savu kodu. Visu šo iemeslu dēļ nepārvaldītus komponentus ieteicams izsaukt tikai galējas nepieciešamības gadījumā. Riska mazināšanai šādām lietotnēm jāpiemēro lomās bāzētas aizsardzības (RBS) ieteikumi.

Pārvaldītu komponentu izsaukšanā arī pastāv zināms risks:

- ja komponents nav parakstīts ar ciparparakstu, pastāv iespēja, ka ļaunprātīgs lietotājs to var modificēt vai nomainīt;

- ja izstrādātājs neseko rekomendācijām par aizsargāta koda izstrādi, tas var izraisīt ievainojamības rašanos;

- pats izstrādātājs var pievienot kaitniecisku kodu konfidenciālas informācijas iegūšanai, kontroles pār sistēmu pārņemšanai vai datu dzēšanai.

Visefektīvākais līdzeklis tāda riska ierobežošanai ir mazāko privilēģiju principa izmantošana un minimālu CAS atļauju piešķiršana komponentam. 10.5. tabulā dota vispārīga informācija par ārējo komponentu izsaukšanas riskiem un to ierobežošanas veidiem.

Ja nevar izvairīties no nepārvaldīta koda izsaukšanas, ir jāņem vērā rekomendācijas riska minimizēšanai.

- Jāpārbauda visi ievades un izvades dati. Nepārvaldītas bibliotēkas atšķirībā no .NET *Framework* bibliotēkām ir ievainojamas pret bufera pārpildi. Tāpēc ir svarīgi pārbaudīt ievades datu tipu atbilstību bibliotēkas prasībām. Tāpat jāpārbauda visi izvades dati, kas tiek iegūti no nepārvaldītas bibliotēkas, lai izslēgtu iespēju, ka bibliotēka ir tikusi izmainīta.
- Jāizveido čaula (*shell*) nepārvaldītu DLL funkciju izpildīšanai. Čaulas izveidošana bieži izmantojamām funkcijām ir efektīvs līdzeklis platformas iespēju inkapsulācijai. Drošības nodrošināšanai čaulas klasē jāpievieno ievades un izvades datu pārbaude. Čaulas klases izmantošana kopā ar datu pārbaudi garantē, ka izstrādātājs neaizmirsīs attiecīgā veidā pārbaudīt datus, kas padarīs lietotni drošāku.
- Jāizvairās no koda izpildes ar administratora tiesībām. Uz nepārvaldītu kodu neattiecas CAS ierobežojumi, bet tā darbību nosaka operētājsistēmas ierobežojumi. Ja lietotājs

izpildīs lietotni ar parasta lietotāja tiesībām, ļaunums, ko varētu sagādāt nepārvaldīts kods, būs mazāks, nekā lietotnes izpildes ar administratora tiesībām gadījumā.

10.5.tabula

Ārējo komponentu izsaukšanas riski

Komponents	Risks	Riska ierobežošana
.NET komponenti	ja komponents nav parakstīts, to var izmainīt vai nomainīt var saturēt ievainojamību var saturēt kaitniecisku kodu	izsaukt tikai parakstītus komponentus ierobežot privilēģijas ar CAS palīdzību izmantot uzticamu izstrādātāju komponentus
COM un Win32 DLL bibliotēkas	nedarbojas CAS ierobežojumi var būt bufera pārpildes ievainojamība; var tikt izmainīti vai aizvietoti	neizpildīt lietotnes ar administratora privilēģijām pārbaudīt ievades un izvades datus izvairīties izmantot COM komponentus
ActiveX vadības elementi	nedarbojas CAS ierobežojumi pieprasa piekļuvi reģistram	neizpildīt lietotnes ar administratora privilēģijām pārbaudīt ievades un izvades datus izvairīties izmantot ActiveX komponentus
Tīmekļa dienesti	ievainojamība pret uzbrukumiem ar starpnieku tīkla infrastruktūra var būt nedroša ārējais serveris var nenodrošināt pārraidāmo datu aizsardzību	strādāt ar tīmekļa dienestiem, kas izmanto SSL sertifikātus izmantot šifrēšanu, ja tīmekļa dienests to uztur strādāt tikai ar uzticamiem tīmekļa dienestiem

Nepārvaldīta koda izsaukšana vienmēr rada papildu risku, jo šāds kods var tikt izmainīts vai nomainīts ar kaitniecisku kodu. Ja lietotne izsauc nepārvaldītu kodu, tad kaitnieciskais kods tiek izpildīts kā uzticams. Viena no iespējām tāda riska minimizēšanai ir nepārvaldīta koda datnes integritātes pārbaude pirms izpildīšanas. To var izdarīt, izskaitļojot datnes jaucējfunkcijas vērtību un salīdzinot ar iepriekš zināmo jaucējfunkcijas vērtību (sk. 7.8. piemēru).

Ārējās datnes integritātes pārbaudes veikšana nav tik vienkārša, kā varētu likties. Piemēram, vairums izstrādātāju ceļa noteikšanā līdz ārējai DLL datnei paļaujas uz .NET *Framework* izpildes vidi, uzdodot tikai datnes vārdu. Tas ļauj lietotnei strādāt sistēmās ar nestandarta mapju struktūru. Taču tādā gadījumā nevar garantēt, ka tiks ielādēta tieši tā datne, kas tiek pārbaudīta lietotnes kodā.

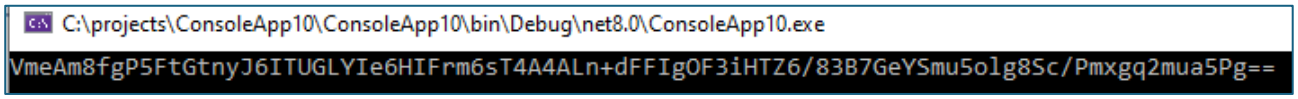
10.2. piemērā tiek izskaitļota ārējās DLL datnes jaucējfunkcijas SHA512 vērtība.


10.2. piemērs. Datnes jaucējfunkcijas vērtības izskaitļošana

```
// Modificets 7.8. piemers
using System;
using System.Text;
using System.IO;
using System.Security.Cryptography;

namespace CreateHash
{
```

```
   class Program
   {
      static void Main(string[] args)
      {
         SHA512 myHash = new SHA512Managed();  // 1. solis
         FileStream file = new FileStream(@"c:\windows\system32\User32.dll", FileMode.Open,
FileAccess.Read); //2. solis
         BinaryReader reader = new BinaryReader(file);
         myHash.ComputeHash(reader.ReadBytes((int)file.Length)); // 3. solis
         Console.WriteLine(Convert.ToBase64String(myHash.Hash)); // 4. solis
         Console.ReadLine();
      }
   }
}
```

Rezultāts



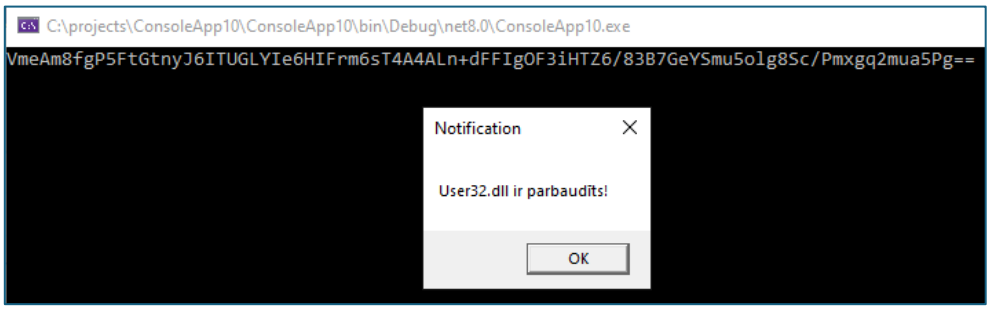
Pieņem, ka šīs datnes jaucējfunkcijas vērtība tika izskaitļota lietotnes izstrādes laikā, kad bija pārliecība par tās darbības pareizību. Turpmākajā lietotnes darbībā ir nepieciešams pārbaudīt, vai šī datne ir korekta, t. i., vai tā nav bijusi modificēta. 10.3. piemērā tiek demonstrēta datnes jaucējfunkcijas vērtības pārbaude. Atribūtā `DllImport` tiešā veidā uzdots pilns ceļš līdz datnei, tad tiek izskaitļota jaucējfunkcijas vērtība. Ja iegūtā jaucējfunkcijas vērtība nesakrīt ar agrāk iegūto jaucējfunkcijas vērtību (iepriekšējā piemērā), tiek izdots brīdinājuma paziņojums.

 10.3. piemērs. Datnes jaucējfunkcijas vērtības pārbaude

```
using System;
using System.Security.Cryptography;
using System.Text;
using System.IO;
using System.Runtime.InteropServices;
using System.Security;

class VerifyHash
{
[DllImport(@"C:\windows\system32\user32.dll")]
public static extern int MessageBox(int h, string m, string c, int type);

static void Main(string[] args)
{

string correctHash =
"VmeAm8fgP5FtGtnyJ6ITUGLYIe6HIFrm6sT4A4ALn+dFFIgOF3iHTZ6/83B7GeYSmu5olg8Sc/
Pmxgq2mua5Pg==";

// Izskaitlo jaunu hash vērtību un saglabā rindā actualHash
SHA512 newHash = new SHA512Managed();
FileStream file = new FileStream(@"C:\windows\system32\user32.dll", FileMode.Open,
FileAccess.Read);
BinaryReader reader = new BinaryReader(file);
newHash.ComputeHash(reader.ReadBytes((int)file.Length));
```

```
string actualHash = Convert.ToBase64String(newHash.Hash);
Console.WriteLine(Convert.ToBase64String(newHash.Hash));

// Ģenerē izņēmuma gadījumu,
if (correctHash != actualHash)
throw new SecurityException("User32.dll nav korekts");
// Ja hash sakrīt, viss O.K. un ar šo datni var strādāt
MessageBox(0, "User32.dll ir parbaudīts!", "Notification", 0);
}
}
```

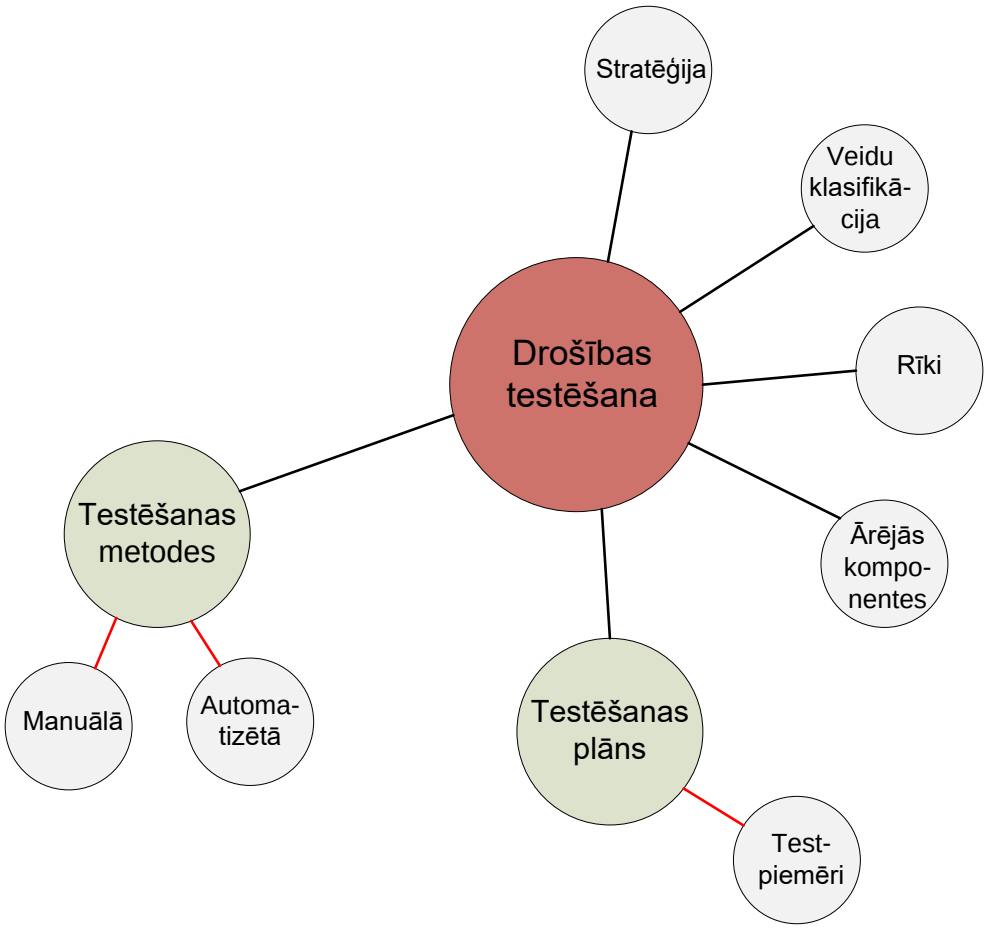
Rezultāts



Šī lietotne strādās korekti, taču ir daži aizrādījumi:

- nedrīkst paļauties uz to, ka *Visual Studio* automātiski apstrādās norādi uz datni. Tāpēc ieteicams norādīt pilnu ceļu līdz datnei ar atribūta `DllImport` palīdzību;
- ja administrators pamainīs ceļu līdz DLL datnei, lietotne nespēs to ielādēt;
- ja izstrādātājs veiks izmaiņas DLL datnē, tad pārbaudes rezultāts būs negatīvs.

10.2. attēlā parādīts nodaļā minēto svarīgāko jēdzienu koks.



10.2. att. **Desmitajā nodaļā minēto svarīgāko jēdzienu koks**

Pārbaudes testa jautājumi

1. Izstrādātāji dod Jums URL tīmekļa lietotnes formas testēšanai. Jums nav piekļuves sākuma kodam. Kāds no testēšanas veidiem ir jāizvēlas tīmekļa lietotnes testēšanai?
    a. Integritātes testēšana
    b. Baltās kastes testēšana
    c. Melnās kastes testēšana
    d. Nekas no minētā

2. Nepieciešams bloķēt atsevišķus HTTP pieprasījumus tīmekļa serverim. Kādu no rīkiem jāizmanto?
    a. Microsoft Baseline Security Analyzer
    b. IIS Lockdown
    c. URLscan Security
    d. Nekas no minētajiem

3. Kas no minētā tiek izmantots laba testa plāna izstrādē?
    a. Sadalīt lietotni fundamentālos komponentos
    b. Identificēt komponentu saskarnes
    c. Atrast drošības problēmas ar mainīgu datu palīdzību
    d. Viss minētais

4. Ir ASP.NET lietotne, kas apstrādā pieprasījumus ASP.NET tīmekļa lapā. Lietotne izmanto COM komponentu piekļuvei SQL *Server* datubāzei. Kuram no šiem komponentiem nav nepieciešama drošības testēšana?
    a. ASP.NET tīmekļa lapa
    b. COM
    c. SQL *Server*
    d. Nekam no minētā

5. Kas no minētā ir lietotnes saskarnes piemērs?
    a. *.dll* funkcijas
    b. Tīmekļa lietotnē iekļautās izpildāmās datnes
    c. TCP/IP *sockets*
    d. COM komponenti

6. Kas no minētā ir testpiemēra komponents?
    a. Testa nosacījums
    b. Ievades nosacījumu detaļas
    c. Sagaidāmais rezultāts
    d. Testpiemēra izpildei nepieciešamais laiks

11. Drošība lietotņu ieviešanas kontekstā

Microsoft drošības nodrošināšanas stratēģija SD³ (sk. 1. nodaļā) paredz veikt drošības pasākumus arī lietotnes ieviešanas laikā, t. i., pēc lietotnes uzstādīšanas klientam ir jāuztur tās aizsardzība, pievienojot sistēmas drošības uzlabojumus un veicot apdraudējumu analīzi. Drošības nodrošināšanas stratēģija lietotņu ieviešanas kontekstā nosaka to, ka darbs pie lietotnes nebeidzas līdz ar lietotnes ieviešanu ekspluatācijā. Nepieciešams uzturēt lietotnes aizsardzību klientu datoros, pieturoties noteiktām rekomendācijām.

Nodaļā apskatītas rekomendācijas lietotņu aizsardzības principa realizācijai ieviešanas laikā, noteikta nepieciešamība parakstīt asamblejas un aprakstīti stingra paraksta realizācijas paņēmieni, ieviešot .NET lietotnes.

11.1. Lietotņu ieviešanas paņēmieni

Lietotnes izstrādes procesā programmētājs realizē savus mērķus – saskarni un funkcionalitāti. Darba beigās rodas nepieciešamība kaut kādā veidā sagatavot lietotni nodošanai lietotājam, t. i., ieviest jeb izvērst (*deployment*) lietotni lietotāja datorā. Datņu tipi, kas var tikt ieviesti kopā ar lietotni, uzrādīti 11.1. tabulā.

11.1.tabula

Lietotņu ieviešanas procesā izmantojamie datņu tipi

Datnes veids	Windows lietotne/serviss	Tīmekļa lietotne/serviss
Izpildāmā datne	X	X
Dinamiskā bibliotēka DLL	X	X
.NET konfigurācijas datnes	X	X
Datu bāzes	X	X
Tīmekļa lappuses		X
Tīmekļa formas		X
Tīmekļa servisu datnes		X
XML shēmas		X

.NET lietotņu ieviešanai var izmantot šādas metodes vai tehnoloģijas:

- Utilītprogrammu *XCopy* [34];
- *Windows Installer* (.MSI) [35];
- *Cabinet-file* (.CAB) [36];
- *ClickOnce* [37].

.NET *Framework* vienkāršo ieviešanas procesu ar utilītprogrammas *XCopy* palīdzību, mehāniski iekopējot mapi, kas satur lietotnes datnes, lietotāja datorā. Šāda metode pielietojama tikai gadījumos, kad lietotne satur tikai .NET komponentus, jo tiem nav nepieciešama turpmākā reģistrācija – ASP.NET lietotnēm, tīmekļa servisiem un klienta lietotnēm.

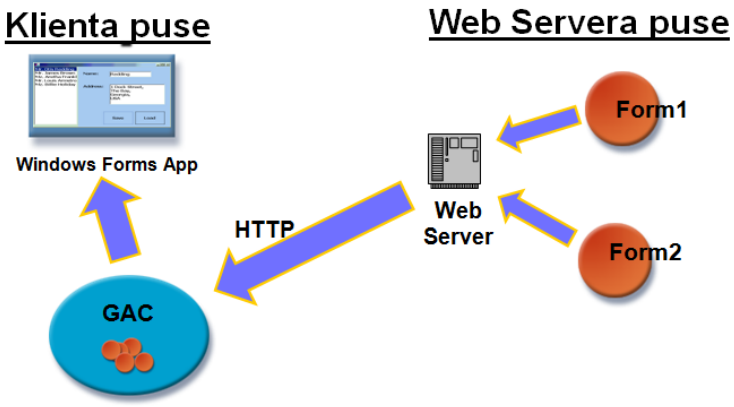
Ja lietotne izmanto daudzas asamblejas, nepieciešams specificēt uzstādīšanas parametrus un uzrādīt ievietošanas mapes nosaukumu, jāizmanto instalēšanas pakotne *Windows Installer*, kas piedāvā dažādus *Windows* lietotņu uzstādīšanas un konfigurēšanas servisus. Šī metode ir lietderīga, piemēram, gadījumos, ja lietotne satur *ActiveX* un .NET komponentus, kam nepieciešama speciāla reģistrācija. *Windows Installer* pakotni ieteicams izmantot lietotņu ieviešanas nolūkā, ja lietotnei jānodrošina šādas iespējas:

- papildu konfigurācijas iestatījumi, piemēram, NTFS drošībai;
- instalēšanas iestatījumi lietotājam;
- īsinājumikonas;

- lietotnes ievietošanas mapes nosaukums;
- .NET DLL komponentu instalēšana globālajā asambleju glabātavā (GAC).

Lietotņu ieviešanai var izmantot arī citas izplatītas metodes, piemēram, saspiesto *.CAB* (*cabinet*) datņu izmantošanu. To parasti izmanto *ActiveX* komponentu lejupielādē no Internet tīkla.

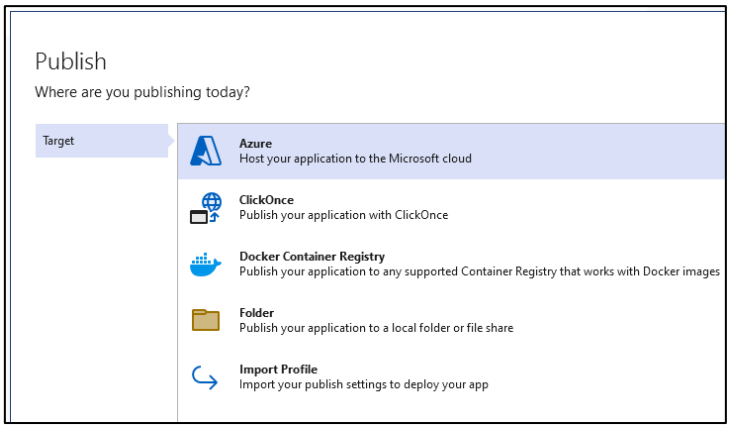
Ļoti perspektīva izrādījās jauna lietotņu ieviešanas tehnoloģija .NET vidē *No-touch* (ieviešana „bez kontakta"). Šī tehnoloģija nodrošina klientam nepieciešamo komponentu automātisku lejupielādi no servera atjaunināšanas gadījumā, neprasa instalēšanu (sk. 11.1. att.). Izstrādātājs izvieto lietotnes datnes tīmekļa serverī. Lietotājs ar HTTP pievienojas serverim. Nepieciešamās datnes tiek lejupielādētas GAC mapēs `windir>\assembly\download` un `Temporary Internet Files`. Turpmākajās lietotnes palaišanas reizēs tiek salīdzināts lietotnes versijas numurs GAC un serverī, nepieciešamības gadījumā tiek lejupielādēta pēdējā versija.



11.1. att. **Lietotņu ieviešanas „bez kontakta" darbības princips**

No-touch tehnoloģija tika modernizēta un nomainīta ar *ClickOnce*, kura atvieglo un padara vēl drošāku lietotņu ieviešanas procesu. Tā ļauj izstrādātājiem publicēt (*publish*), uzstādīt un apkalpot lietotnes.

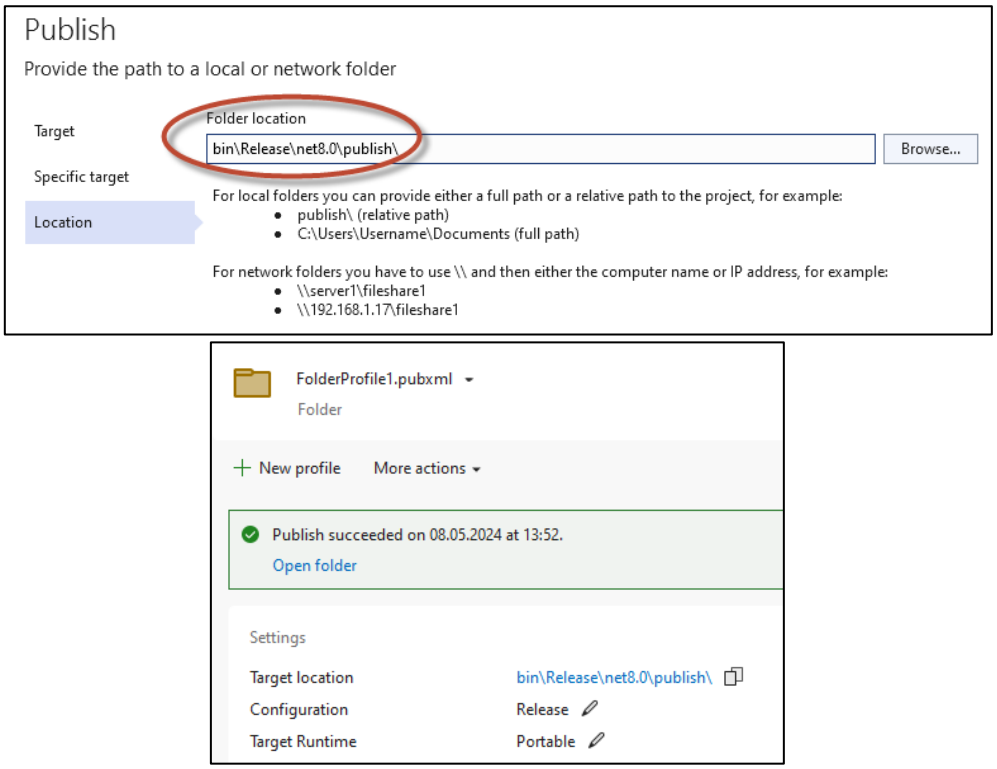
Visual Studio vidē lietotni var publicēt, secīgi izvēloties **Build / Publish Selection** (sk. 11.2. att.).



11.2. att. **Lietotņu publicēšanas vednis**

11.1. piemērs. Konsoles aplikācijas publicēšana

1. Uzraksta vienkāršu konsoles aplikāciju ar nosaukumu **tests**, kas izvada ekrānā paziņojumu „Sveika, pasaule".
2. Publicē šo lietotni ieviešanai mapē. Projekta **tests** mapē tika izveidota apakšmape **\publish**, kurā izvietota izpildāmā datne (sk. 11.3. att.).



11.3. att. **Ieviešanai sagatavotā lietotne**

Kad .NET lietotne tiek ieviesta, .NET koda piekļuves aizsardzības (CAS) mehānisms (sk. 6. nodaļu) nodrošina ierobežojošu vidi, kurā lietotne tiks izpildīta. Šāda vide ir nosaukta par .NET CAS „smilšu kasti" (*sandbox*) [38].

„Smilšu kaste" (*Sandbox*) – mehānisms, kas ļauj droši izpildīt aizdomīgas programmas. Dažkārt saka, ka tas ir virtuāls konteiners, kurā var tikt droši izpildītas neuzticamas lietotnes.

„Smilšu kastes" bieži izmanto nenotestēta vai nepazīstama koda pārbaudei, kā arī vīrusu identificēšanai. Šādiem nolūkiem tiek izdalīti stingri kontrolēti resursi, piemēram, atmiņas apgabals vai vieta cietajā diskā. Piekļuve tīklam, informācijas nolasīšana no ievadizvades iekārtām vai saskarne ar operētājsistēmu tiek stipri ierobežota vai emulēta. Tāpēc arī „smilšu kaste" tiek uzskatīta par virtualizācijas piemēru.

Ja lietotne tiek ieviesta *Internet* tīklā vai *Intranet* tīklā, tā tiek palaista „smilšu kastes" vidē. Ja lietotne tiek instalēta un palaista `My Computer` zonā, kurā visas atļaujas piešķirtas lietotnei, tā darbojas ārpus „smilšu kastes" vides.

11.2. tabulā parādīta ieviešanas tehnoloģiju saistība ar CAS.

11.2. tabula

Ieviešanas tehnoloģiju saistība ar CAS

Ieviešanas tehnoloģija	Koda piekļuves aizsardzība	Saistība ar „smilšu kasti"
XCopy *Windows Installer*	CAS piešķir pilnu uzticamību lietotnei, jo tā tiek palaista *My Computer* zonā	Ieviešamā lietotne netiek palaista „smilšu kastes" vidē tāpēc, ka lietotne tiek izpildīta lokālā datora *My Computer* zonā
Cabinet-file *ClickOnce*	Lietotnei ir CAS atļaujas, kas bāzējas uz zonas atļaujām, no kuras lietotne ir instalēta	Ieviešamā lietotne tiek palaista „smilšu kastes" vidē

Vispārīgā gadījumā lietotņu ieviešanas darbības procesā ieteicams pieturēties pie dažām rekomendācijām.

- Pati svarīgākā rekomendācija – nepieciešams uzturēt pasākumus jaunatklātās ievainojamības fiksēšanai un novēršanai.
- Nepieciešams piedāvāt klientam sistēmu, kas dod iespēju sekot notikumiem lietotnes darbības laikā. Būtu vēlams klientu savlaicīgi informēt par to, ja lietotne tiek apdraudēta.
- Lietotnei jāspēj apstrādāt dažādas iespējamās kļūdas, sniedzot par tām minimālu informāciju lietotājam, bet dodot iespēju administratoram noteikt problēmas cēloni un novērst to.

11.2. Lietotnes integritātes nodrošināšana

Lietotņu integritāti ieviešanas kontekstā nodrošina ar šādu metožu palīdzību:

- ar autentifikācijas koda parakstu (*Authenticode signing*);
- ar stingru parakstu (*Strong-name signing*).

Autentifikācijas kods – vadības ziņojumu lauks, ko piekārto datu blokam, lai identificētu ziņojumu. Autentifikācijas kodā parasti ietilpst speciāla parole, ziņojuma kārtas numurs, norāde uz pārraides laiku un datumu, kā arī cita informācija, kas atvieglo datu nesankcionētas izmantošanas un izmaiņas atklāšanu.

Autentifikācijas koda paraksta metodoloģija pievieno X.509 sertifikātu asamblejai publicētāja identitātes pārbaudei, kamēr stingrais paraksts veic lietotnes integritātes pārbaudi.

Autentifikācijas koda paraksts balstās uz šādiem standartiem:

- PKCS #7 (*Public Key Cryptography Standards*);
- PKCS #10 (*certificate request formats*);
- X.509 (*certificate specification)*;
- SHA un MD5 (*hash algorithms*).

11.3. tabulā salīdzinātas autentifikācijas koda paraksta un stingra paraksta iespējas.

11.3.tabula

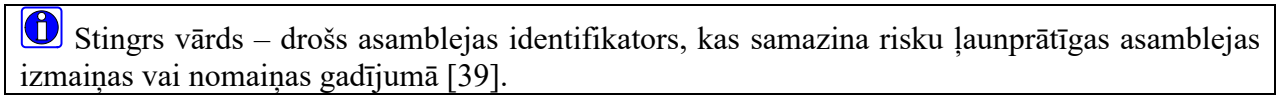
Lietotņu integritātes nodrošināšanas metožu iespējas

Iezīme	Autentifikācijas koda paraksts	Stingrs paraksts
Lietotnes integritātes pārbaude, lejupielādējot no *Internet* tīkla	+	+
Izpildāmo datņu uzturēšana (*.exe* un *.dll*)	+	+
Lietotnes darbības ilguma (*lifespan*) nodrošināšana	+	
Lietotnes publicētāja uzticamības pārbaudes nodrošināšana	+	
Saderības ar uzstādīšanas pakotnēm (*.msi* un *.cab*) pārbaudes nodrošināšana	+	
Lietotnes integritātes pārbaude, lejupielādējot *Windows* vidē		+

Turpmākajā izklāstā lielāka vērība tiks pievērsta stingra vārda parakstam, jo tas ļauj efektīvi nodrošināt lietotnes komponentu aizsardzību.

Vairākkārtīgi jau tika atzīmēts, ka lietotnes komponentu nomaiņas iespēja ir viens no būtiskajiem ievainojamību veidiem. Pēc noklusējuma asamblejas identificē viena otru tikai pēc datnes

nosaukuma, tādējādi radot iespēju ļaunprātīgam lietotājam iejaukties lietotnes izpildes procesā. .NET vidē pastāv iespēja izmantot stingrus vārdus, kas apgrūtina iespēju viltot asamblejas. Stingrs vārds ļauj identificēt asambleju pēc vairākiem faktoriem. Ja tāda asambleja tiks nomainīta vai izmainīta, .NET vide to konstatēs un noģenerēs izņēmuma stāvokli līdz tam, kad tiks palaists šis ļaunprātīgais kods.


Stingrs vārds – drošs asamblejas identifikators, kas samazina risku ļaunprātīgas asamblejas izmaiņas vai nomaiņas gadījumā [39].

Stingrs vārds sastāv no 5 daļām:

1) vārds – datnes vārds;
2) publiskā atslēga – ar RSA šifrēta publiskā atslēga, kas ļauj pārbaudīt asamblejas autentifikāciju;
3) versija – asamblejas versijas numurs formā `Major.Minor.Build.Revision`;
4) kultūra – atribūts, kas raksturo vidi, kurā tika izveidota asambleja. Parasti piešķir tukšu vērtību, lai nodrošinātu asamblejas neatkarību;
5) procesora arhitektūra – definē asamblejas formātu (MSIL vai x86).

Stingra vārda atslēgas datnes tips ir *.snk* (*Strong Name Key*), un tā satur unikālu publiskās un privātās atslēgu pāri. Var teikt, ka tādējādi šis atslēgu pāris tiek izmantots asamblejas ciparparakstam. Lai nodrošinātu šīs atslēgas datnes papildus aizsardzību, *Visual Studio* vidē to var aizsargāt ar paroli, un šajā gadījumā stingra vārda atslēgas datnes tips būs *.pfx* (*Personal Information Exchange*).

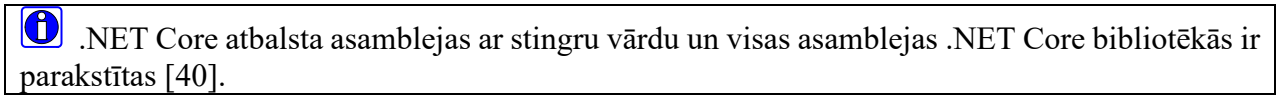
.NET izmanto ciparparakstu, lai nodrošinātu asamblejas integritāti. Paraksts tiek ģenerēts un pārbaudīts ar publiskās atslēgas kriptogrāfiju, izmantojot RSA publiskās atslēgas algoritmu un SHA-1 jaucējfunkcijas algoritmu. Ja asambleja tiek parakstīta ar stingru vārdu, izstrādātājs faktiski to paraksta ar savu privāto atslēgu. Turpmāk, ielādējot asambleju, tā tiek pārbaudīta ar atbilstošo publisko atslēgu.

Kas notiek asamblejas parakstīšanas laikā? Kad asambleja tiek kompilēta ar stingra vārda atslēgas datnes palīdzību, kompilators paraksta asambleju ar ciparparaksta palīdzību:

1) kompilators izskaitļo asamblejas satura jaucējfunkcijas vērtību (*compile-time digest*);
2) kompilators šifrē jaucējfunkcijas vērtību ar 1024 bitu privātās atslēgas palīdzību;
3) kompilators ievieto šifrēto vērtību un publisko atslēgu asamblejā.

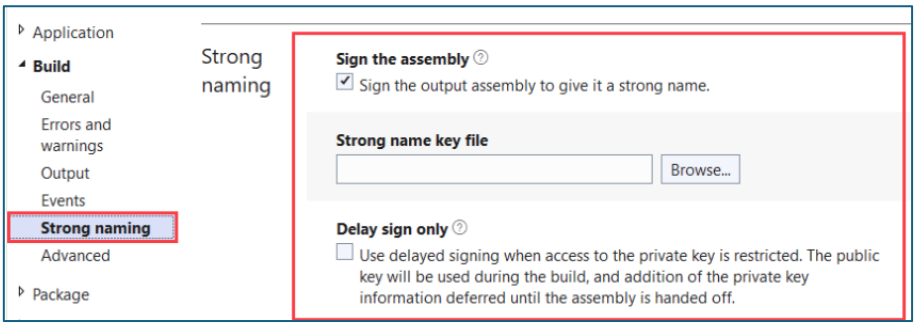
Kad lietotnei nepieciešams ielādēt parakstīto asambleju:

1) .NET asambleju ielādētājs izskaitļo pašreizējo asamblejas satura jaucējfunkcijas vērtību (*run-time digest*);
2) ielādētājs iegūst saglabāto šifrēto jaucējfunkcijas vērtību un publisko atslēgu no asamblejas;
3) ielādētājs ar publiskās atslēgas palīdzību atšifrē jaucējfunkcijas vērtību;
4) ielādētājs salīdzina pašreizējo jaucējfunkcijas vērtību ar atšifrēto vērtību (no asamblejas). Ja tās nesakrīt, asambleja ir tikusi modificēta un tās izpilde tiek pārtraukta.


.NET Core atbalsta asamblejas ar stingru vārdu un visas asamblejas .NET Core bibliotēkās ir parakstītas [40].

Tālāk minētās darbības vairāk attiecas uz vecākajām *Visual Studio* versijām. Taču var gadīties situācijas, kad programmētājam nākas parakstīt asamblejas.

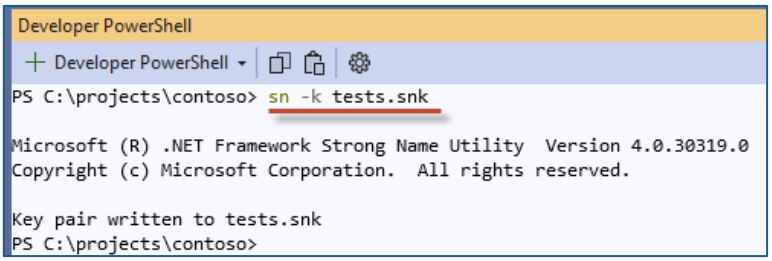
Lai parakstītu asambleju ar stingru vārdu, *Visual Studio* vidē logā **Solution Explorer** uz projekta nosaukuma uzklikšķina ar peles labo taustiņu un izvēlas **Properties**. Atvērtajā dialoga logā izvēlas **Build / Strong naming**. Ievada atslēgas datnes nosaukumu (sk. 11.7. att.).



11.7. att. **Asamblejas parakstīšana ar stingru vārdu**

Ja **Strong naming** nav aktīvs, stingrais vārds jāuzdod manuāli.

1. Atver .NET komandrindu:
   - uz projekta nosaukuma ar labo peles taustiņu izvēlas **Open in Terminal**;
   - uzraksta komandu **sn -k tests.snk** (sk. 11.8. att.).



11.8. att. **DLL parakstīšana ar stingru vārdu**

2. **Assembly** direktīvā uzraksta pilno ceļu līdz DLL datnei (sk. 11.2. piemēra 3. etapu).
3. Atkārtoti kompilē: **Build Solution**.

11.2. piemērā tiek demonstrēta stingra vārda izmantošana matemātisko funkciju bibliotēkas parakstīšanai. Pieņem, ka dota forma, kurā tiek veikta divu skaitļu saskaitīšana. Saskaitīšanas funkcija realizēta atsevišķā klasē, kurai kā argumenti tiek nodoti divi veseli skaitļi un kura atgriež rezultātu – divu skaitļu summu. Ir nepieciešams parakstīt ar stingru vārdu asambleju, kura realizē šo klasi, un pārliecināties, ka šīs asamblejas modificēšana izraisīs lietotnes izpildes kļūdu.

11.2. piemērs. Stingra vārda izmantošana matemātisko funkciju bibliotēkas parakstīšanai

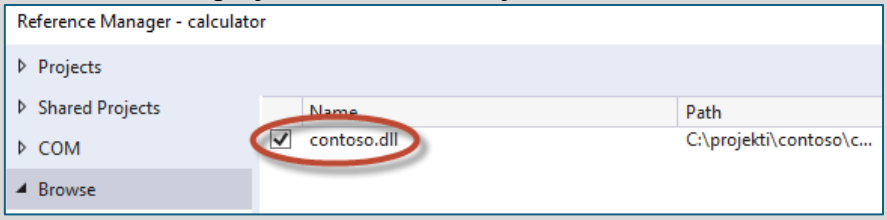
1. etaps
1.Visual Studio vidē izveido jaunu **Class Library** projektu ar nosaukumu **contoso**.
2.Programmas logā ievieto šādu kodu: **using System;** **using System.Text;** **namespace contoso** **{** **public class Math** **{** **public static int Add(int x, int y)** **{** **return x + y;** **}** **}** **}**

Te ir tikai viena statiska funkcija **Add**, kas kā argumentus pieņem divus veselus skaitļus, saskaita tos un atgriež summas vērtību.
Kompilē projektu (**Build Solution**) un aizver (tiek izveidota bibliotēka **contoso.dll**).

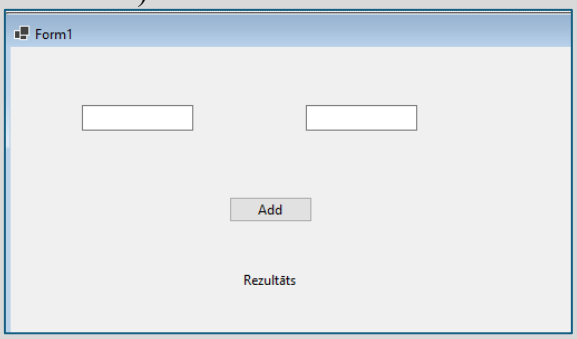
2. etaps

3. Izveido jaunu **Windows Forms App** projektu ar nosaukumu **calculator**.

4. Projektā jāpievieno atsauce uz **contoso.dll. Add / Project Reference**. Ja klases nosaukums nav redzams, tad ar **Browse** meklē projektu **contoso** un tajā izveidoto bibliotēku **contoso.dll:**



5. Formā jāievieto divi **TextBox** objekti (skaitļu ievadei), viens **Label** objekts (rezultātam) un viena poga **Button** (saskaitīšanas veikšanai).

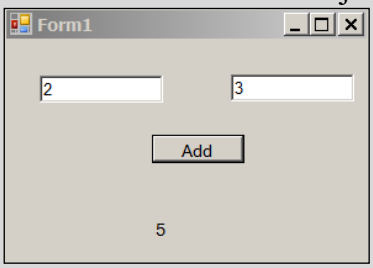


Teksta laukos tiks ievadīti divi veseli skaitļi. Nospiežot pogu **Add**, tie tiks nodoti metodei **`contoso.Math.Add`** un summa tiks atspoguļota rezultāta laukā.

6. Uzklikšķinot uz pogas **Add**, notikumā **`button1_Click`** ievieto šādu kodu:

```
int x = int.Parse(textBox1.Text);
int y = int.Parse(textBox2.Text);
int result = contoso.Math.Add(x, y);
label1.Text = result.ToString();
```

7. Jāpārliecinās, ka programma strādā korekti. Rezultātam vajadzētu būt šādam:



Ja viss kārtībā, aizver projektu.

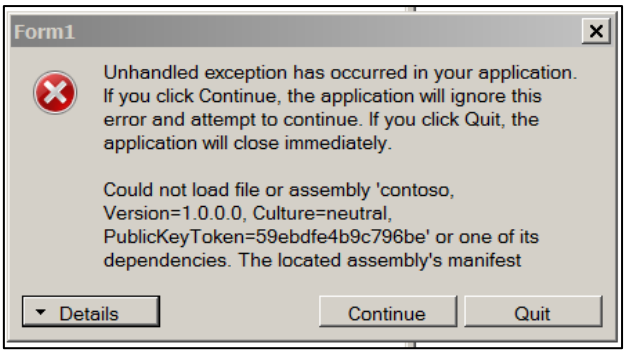
3. etaps

8. Atkal atver projektu **contoso**. Asambleja ir jāparaksta ar stingru vārdu. Manuāli veic 11.8. attēlā aprakstītās darbības. Tad programmā **Class1.cs** pievieno papildu rindas:

```
using System.Reflection;
[assembly: AssemblyKeyFileAttribute(@"C:\projects\contoso\tests.snk")]
[assembly: AssemblyDelaySignAttribute(true)]   //false ?
```

Bibliotēka **contoso.dll** tagad ir parakstīta ar stingru vārdu. Atkārtoti kompilē projektu un aizver.
4. etaps
9. Projektu **calculator** atkārtoti jāpārkompilē, lai tiktu iekļauta bibliotēka **contoso.dll** ar stingru vārdu. Pēc tam visus projektus aizver.
5. etaps
10. Ļaunprātīga lietotāja darbības imitācijai nepieciešams atkal izveidot **Class Library** projektu ar nosaukumu **contoso** (pirms tam oriģinālo projektu vajadzētu pārsaukt citā vārdā, piemēram, **contoso2**. Pirmā etapa otrajā solī uzrādītajā kodā var pamainīt atgriežamās darbības rezultātu, piemēram, **return x - y;**. Stingru vārdu šoreiz neuzdod. Visu kompilē un aizver projektu. Tiek iegūta modificēta bibliotēka **contoso.dll**, kas veic pavisam citu funkciju nekā sākotnēji paredzēts.
11. Jauniegūto bibliotēku **contoso.dll** iekopē projekta **calculator** mapē **\bin\Debug** (tādējādi pārrakstot oriģinālo bibliotēku).

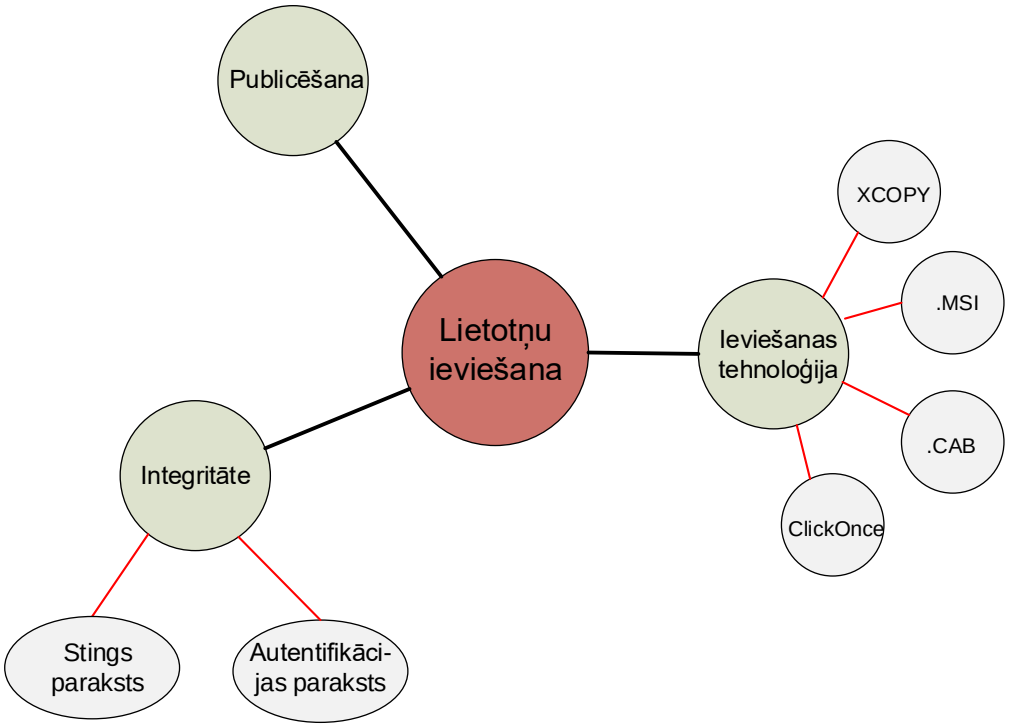
Palaižot datni *calculator.exe*, skaitļus var ievadīt, bet, noklikšķinot uz pogas **Add**, vajadzētu tikt ģenerētai izņēmuma situācijai, kā redzams 11.9. attēlā. Tādējādi konstatēts, ka ir noticis neveiksmīgs mēģinājums nomainīt oriģinālo bibliotēku, kas parakstīta ar stingru vārdu, pret modificētu.



11.9. att. **Paraksta neatbilstība oriģinālajai asamblejai ar stingru vārdu**

Nopietnākam darbam ar stingru parakstu der iepazīties ar komandrindas utilītprogrammu *Strong Name* **sn.exe**, kas tiek raksturota kā galvenais līdzeklis stingra vārda atslēgu izveidošanai un to pārvaldībai [41].

11.10. attēlā parādīts nodaļā minēto svarīgāko jēdzienu koks.



11.10. att. **Vienpadsmitajā nodaļā minēto svarīgāko jēdzienu koks**

Pārbaudes testa jautājumi

1. Tiek plānots ieviest lietotni ar 10 iezīmēm (*features*), no kurām 4 ir izvēles. Kāda no dotajām metodēm jāizvēlas lietotnes ieviešanai?
   a. ClickOnce
   b. MSI
   c. *Cabinet* datnes
   d. XCopy

2. Visas lietotnes asamblejas ir parakstītas ar stingru vārdu. Kurš no uzskaitītajiem drošības riskiem pastāvēs, ja lietotne ieviešanai tiks izvietota *Internet* tīklā?
   a. Izlikšanās (*Spoofing*)
   b. Datu falsifikācija (*Data Tampering*)
   c. Dalības noliegums (*Repudiation*)
   d. Privilēģiju paaugstināšana (*Elevation of Privileges*)

3. Tiek plānots izstrādāt programmu un padarīt to pieejamu Internet tīklā. Ir nepieciešams nodrošināt iespēju, lai klientam, kas ievieš lietotni, būtu realizēta „smilšu kastes" aizsardzība. Kura no uzskaitītajām ieviešanas metodēm būtu jāpielieto?
   a. XCopy
   b. MSI
   c. ClickOnce
   d. *Zip* arhivēšana

4. Kuru no minētajām metodēm no lietotnes publicētāja viedokļa vajadzētu izmantot, lai ieviešanas pakotnē tiktu iekļauta publicētāja uzticamības pārbaude?
   a. Stingra vārda paraksts
   b. Autentifikācijas koda paraksts
   c. Aizsardzība ar paroli

12. .NET *Core*

12.1 .NET *Core* arhitektūra

.NET *Core* ir jauna .NET *Framework* versija, kas ir bezmaksas atvērtā koda vispārējas nozīmes izstrādes platforma, ko uztur *Microsoft* [42]. To var izmantot, lai izveidotu dažāda veida lietojumprogrammas, piemēram, mobilās lietotnes, tīmekļa aplikācijas, mākoņrisinājumus, mašīnmācīšanās, spēles utt.

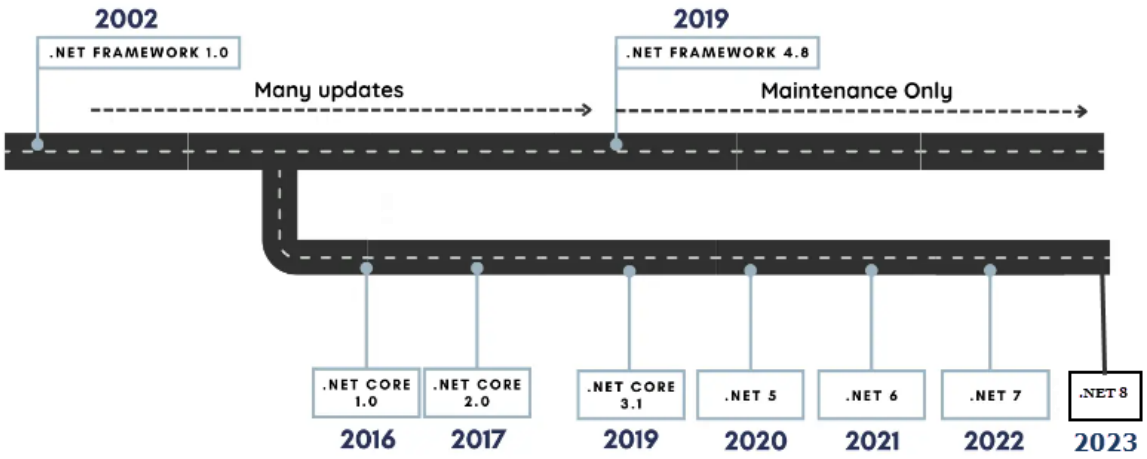
.NET *Core* ir uzrakstīts no jauna, lai padarītu to modulāru, vieglāku, ātrāku un piemērotu vairākām platformām. Tas ietver pamatfunkcijas, kas nepieciešamas, lai palaistu pamata .NET *Core* lietotnes. Citas funkcijas tiek nodrošinātas kā *NuGet* pakotnes, kuras pēc vajadzības var pievienot savai lietojumprogrammai. Tādā veidā .NET *Core* lietojumprogrammas paātrina veiktspēju, samazina atmiņas apjomu. .NET Core versijas parādītas 12.1. tabulā un 12.1. attēlā.

NET *Core* ietvars tika nosaukts par .NET 5 pēc .NET *Core* 3.1.

12.1. tabula

.NET Core versijas

Versija	*Visual Studio* versija	Izlaiduma datums	Uzturēšanas beigas
.NET 8	*Visual Studio 2022*	*Nov 14, 2023*	*Nov 10, 2026*
.NET 7	*Visual Studio 2022 v17.4*	*Nov 8, 2022*	*May 14, 2024*
.NET 6	*Visual Studio 2022*	*Nov 9, 2021*	*Nov 12, 2024*
.NET 5	*Visual Studio 2019*	*Nov 10, 2020*	*May 10, 2022*
.NET Core 3.1	*Visual Studio 2019*	*Dec 3, 2019*	*Dec 13, 2022*
.NET Core 3.0	*Visual Studio 2019*	*Sep 23, 2019*	*Mar 3, 2020*
.NET Core 2.1	*Visual Studio 2017, 2019*	*May 30, 2018*	*Aug 21, 2021*
.NET Core 2.0	*Visual Studio 2017, 2019*	*Aug 14, 2017*	*Oct 1, 2018*
.NET Core 1.0	*Visual Studio 2017*	*Jun 27, 2016*	*Jun 27, 2019*



12.1. att. **.NET attīstība pa gadiem**

Iepriekšējām .NET *Framework* versijām bija dažādi ierobežojumi. Piemēram, tās darbojās tikai *Windows* platformā. Tāpat bija jāizmanto dažādas .NET API *Windows Desktop*, *Windows Store*, *Windows Phone* un tīmekļa aplikācijām. Papildus tam .NET *Framework* bija "mašīnas mēroga" (*machine-wide framework*) ietvars. Visas tajā veiktās izmaiņas ietekmēja lietotnes, kas no tā bija atkarīgas.

Mūsdienās ir izplatītas lietotnes, kas darbojas dažādās ierīcēs – tātad ir vajadzīga vienota sistēma, kas darbojas visur. Ņemot to vērā, *Microsoft* izveidoja .NET *Core*, kuras galvenais mērķis ir padarīt .NET *Framework* saderīgu ar atvērtā koda starpplatformām, ko var izmantot dažādos risinājumos, sākot no datu centriem līdz skārienjutīgām ierīcēm.

.NET *Core* **uztur** [43]:

- atvērtā koda ietvaru: .NET *Core* ir atvērtā pirmkoda ietvars, ko uztur *Microsoft* un kas ir pieejams vietnē *GitHub* saskaņā ar MIT un *Apache* licencēm [44]. Tas ir .NET *Foundation* projekts [45];
- vairākas platformas: .NET *Core* darbojas operētājsistēmās *Windows, MacOS* un *Linux*. Katrai operētājsistēmai, kas izpilda kodu un ģenerē vienu un to pašu izvadi, ir atšķirīgs izpildlaiks;
- atbilstību dažādām arhitektūrām: izpildīts kods ar vienādu darbību dažādās instrukciju kopas arhitektūrās, tostarp x64, x86 un ARM;
- plašu lietojumprogrammu klāstu: .NET *Core* platformā var izstrādāt un palaist dažāda veida lietojumprogrammas, piemēram, mobilās lietotnes, tīmekļa aplikācijas, mākoņrisinājumi, lietu internetu, mašīnmācīšanās, spēles utt.;
- daudzvalodu atbalstu: .NET *Core* lietojumprogrammu izstrādei var izmantot C++, C#, F# un *Visual Basic* programmēšanas valodas. Var izmantot iecienītākās IDE, tostarp *Visual Studio* 2017/2019, *Visual Studio* kodu, *Sublime Text, Vim* utt.;
- modulāro arhitektūru: .NET *Core* atbalsta moduļu arhitektūras pieeju, izmantojot *NuGet* pakotnes. Ir dažādas *NuGet* pakotnes dažādām funkcijām, kuras pēc vajadzības var pievienot .NET *Core* projektam. Pat .NET *Core* bibliotēka tiek nodrošināta kā *NuGet* pakotne;
- CLI rīkus: .NET *Core* ietver CLI rīkus (komandrindas saskarne) izstrādei un nepārtrauktai integrācijai;
- elastīgu izvietošanu: .NET *Core* lietojumprogrammas var izvietot visā lietotāja vai sistēmas mērogā vai ar *Docker* konteineriem;
- saderību: ir saderīgs ar .NET *Framework* un *Mono* API, izmantojot .NET *Standard* specifikāciju [46];
- augstu veiktspēju: .NET *Core* tika optimizēts veiktspējai ar tādām funkcijām kā *Just-In-Time* (JIT) kompilācija, kas izpildes laikā pārveido kodu mašīnas instrukcijās, lai nodrošinātu labāku izpildes ātrumu;
- vienotu platformu: pēc .NET *Core* 3.1 *Microsoft* apvienoja .NET platformas, apvienojot .NET *Core*, .NET *Framework* un *Xamarin* vienā platformā ar nosaukumu ".NET", sākot no .NET 5. Šīs vienotās platformas mērķis bija nodrošināt konsekventu API dažādos lietojumprogrammu veidos;

Vairākas tehnoloģijas, kas bija pieejamas .NET *Framework* bibliotēkām, nav pieejamas lietošanai ar .NET 6+, piemēram, lietotņu domēni, attālinātā darbība un koda piekļuves drošība (CAS).

.NET *Core* **vairs neuztur** [26]:

- attālinātu darbību: .NET *Remoting* vairs netiek atbalstīts .NET 6+ versijā. .NET attālināta darbība tika identificēta kā problemātiska arhitektūra. To izmantoja saziņai starp lietojumprogrammu domēniem, kas vairs netiek atbalstīti;
- koda piekļuves drošību: *Sandbox* vairs netiek atbalstīta .NET *Framework* un tāpēc netiek atbalstīta arī .NET 6+ versijā. CAS vairs netiek uzskatīta par drošības robežu, jo .NET *Framework* un izpildlaika izpildē ir pārāk daudz gadījumu, kad tiek paaugstinātas privilēģijas;
- drošības caurspīdīgumu: līdzīgi kā CAS, drošības caurspīdīgums deklaratīvi atdala *Sandbox* kodu no drošības kritiskā koda, taču tā vairs netiek atbalstīta kā drošības

robeža. Lai palaistu procesus ar vismazāko privilēģiju kopu, jāizmanto operētājsistēmas nodrošinātās drošības robežas, piemēram, virtualizāciju, konteinerus vai lietotāju kontus;
- *System.EnterpriseServices* (COM+): nodrošināja .NET objektiem piekļuvi COM+ pakalpojumiem;
- darbplūsmu: *Windows Workflow Foundation* (WF) netiek atbalstīts .NET 6+ versijā. Kā alternatīva tiek piedāvāta *CoreWF*.

12.2 .NET *Core* un ASP.NET

Miljoniem izstrādātāju izmanto vai ir izmantojuši ASP.NET 4.x, lai izveidotu tīmekļa lietotnes. ASP.NET *Core* ir ASP.NET 4.x pārveidots dizains, tostarp arhitektūras izmaiņas, kuru rezultātā tiek iegūta vienkāršāka, modulārāka struktūra [42].

ASP.NET *Core* ir atvērtā koda modulāra tīmekļa lietojumprogrammu sistēma [47]. Tas ir ASP.NET pārprojektējums, kas apvieno iepriekš atsevišķās ASP.NET MVC un ASP.NET *Web* API vienā programmēšanas modelī. Neskatoties uz to, ka tas ir jauns ietvars, tai ir augsta saderības pakāpe ar ASP.NET. ASP.NET *Core* sākotnēji darbojās gan tikai *Windows* .NET *Framework* vidē, tomēr atbalsts .NET *Framework* tika pārtraukts, sākot ar ASP.NET *Core* 3.0.

.ASP.NET *Core* komponentes:
- *Entity Framework* (EF) *Core* [48].
- *Identity Core* [49].
- MVC *Core* [50], (12.1. piemērs).
- *Razor Core* [51], (12.2. piemērs).
- *SignalR* [52].
- *Blazor* [53], (12.2. piemērs).
- *Kestrel web server* [54].

Būtiskākajām komponentēm dots īss skaidrojums. Realizācijas piemēri jāpēta dotajās atsaucēs [48] – [54].

Entity Framework ir atvērtā koda objektu un relāciju kartēšanas ORM (*object–relational mapping*) ietvars ADO.NET (darbam ar datu bāzēm).

Identity ir API, kas atbalsta lietotāja saskarnes (UI) pieteikšanās funkcionalitāti. Pārvalda lietotājus, paroles, profila datus, lomas, pilnvaras un daudz ko citu.

MVC (*Model-View-Controller*) ir programmatūras projektēšanas modelis, ko parasti izmanto lietotāja saskarņu izstrādei, kas satur trīs elementus:
- modeli – informācijas iekšējās reprezentācijas skatu;
- saskarni – sniedz informāciju lietotājam un pieņem to no lietotāja;
- kontrolieri – programmatūru, kas savieno modeli un saskarni.

Tradicionāli izmanto grafiskajās lietotāja saskarnēs (GUI). Šis modelis kļuva populārs tīmekļa lietojumprogrammu izstrādē. Populārajām programmēšanas valodām ir MVC ietvari, kas atvieglo modeļa ieviešanu.

Razor padara uz lapām orientētu scenāriju kodēšanu vieglāku un produktīvāku, nekā izmantojot kontrolierus un skatus.

SignalR ir bezmaksas atvērtā pirmkoda programmatūras bibliotēka ASP.NET, kas ļauj servera kodam nosūtīt asinhronus paziņojumus klienta puses tīmekļa lietojumprogrammām. Bibliotēkā ir iekļauti servera un klienta puses *JavaScript* komponenti.

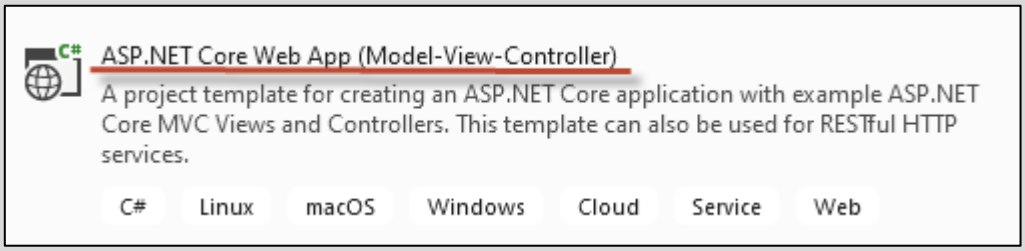
Blazor ir .NET tīmekļa ietvars, kas atbalsta gan servera puses atveidošanu (*rendering*), gan klienta interaktivitāti vienā programmēšanas modelī.

Kestrel ir starpplatformu tīmekļa serveris ASP.NET *Core*. *Kestrel* ir ieteicamais serveris, un tas pēc noklusējuma ir konfigurēts ASP.NET *Core* projektu veidnēs.

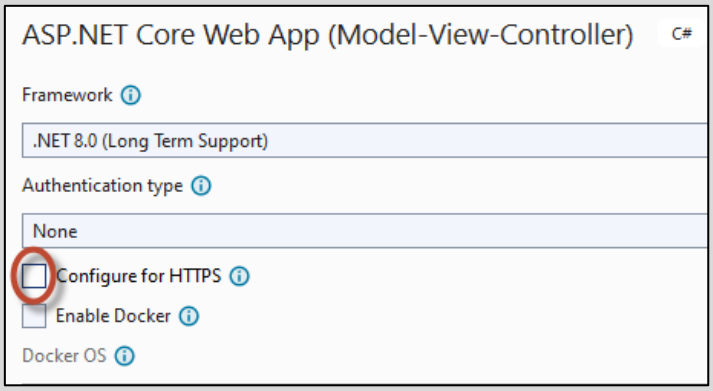
Turpmāk doti daži .ASP.NET *Core* komponentu izmantošanas piemēri.

12.1. piemērs. Vienkāršs ASP.NET *Core* MVC projekts, kas izvada paziņojumu "Sveika pasaule"

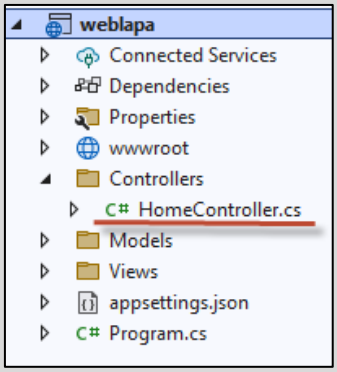
1.*Visual Studio* vidē izveido jaunu projektu ar nosaukumu **weblapa**.



2. Konfigurē projektu:



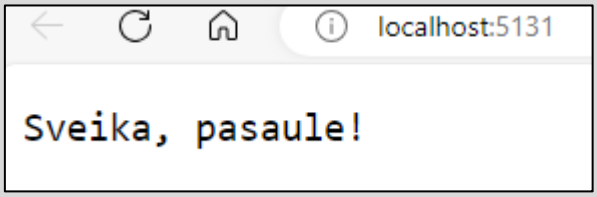
3. **Controllers** sadaļā atver datni **HomeController.cs**:



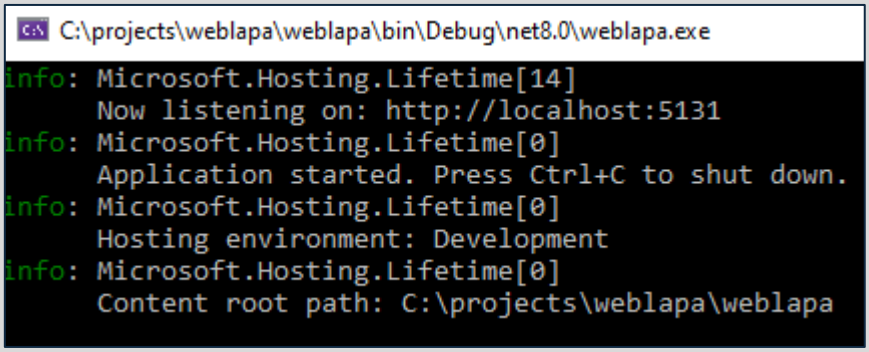
4. Tur esošo kodu nomaina ar šo:

```
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace sveiki.Controllers
{
    public class HomeController : Controller
    {
        public string Index()
        {
            return "Sveika, pasaule!";
        }
    }
}
```

5. ASP.NET *Core* tīmekļa lietojumprogrammas tiek pašas mitinātas iekšējā tīmekļa serverī ar nosaukumu "Kestrel". Tātad nav jāizmanto IIS *Express* vai cits tīmekļa serveris. Tomēr *Visual Studio* sniedz iespēju izmantot IIS **Express**, ja ir vajadzība. Noklikšķinot uz nolaižamās bultiņas ar "http", lai izmantotu vēlamo tīmekļa serveri (šajā gadījumā tiek izmantots tikai *Kestrel*).
Palaižot projektu, tiek iegūts šāds rezultāts:

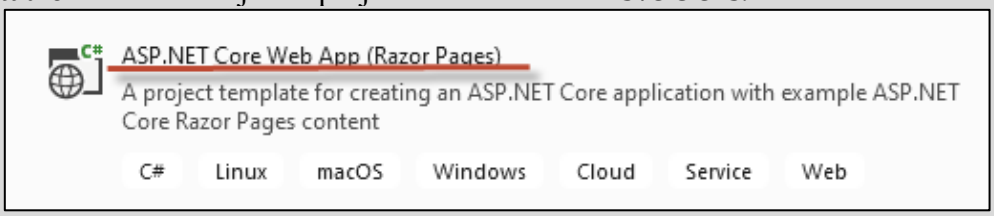


6. Kopā ar pārlūkprogrammu atveras arī terminālis (komandu logu sistēmā). Tas ir paredzēts iekšējam tīmekļa serverim *Kestrel*. Tas parāda URL un citu informāciju. Nospiežot taustiņu kombināciju Ctrl + C, tīmekļa serveris tiks izslēgts, un lietojumprogramma pārtrauks darboties.

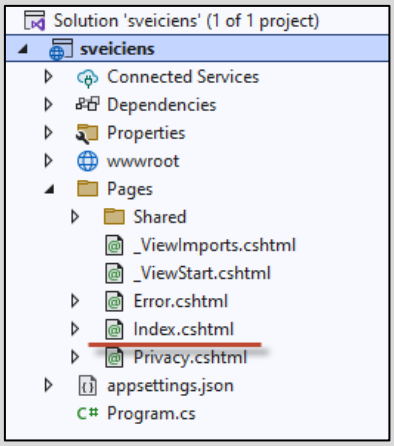


12.2. piemērs. Vienkāršs *Razor* projekts, kas izvada paziņojumu "Sveika pasaule"

1.*Visual Studio* vidē izveido jaunu projektu ar nosaukumu **sveiciens**.

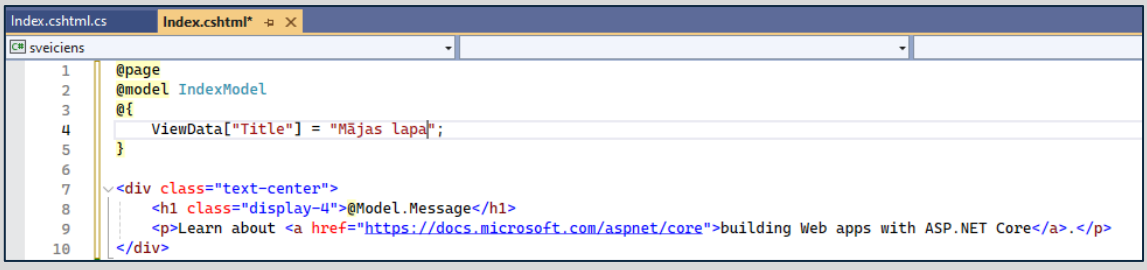


2. Mapē **Pages** izvēlas datni **index.cshtml**:

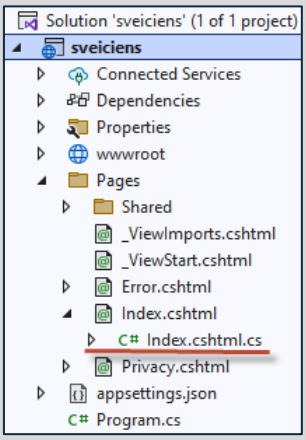


3. Tur esošo kodu nomaina ar šo:

```
@page
@model IndexModel
@{
   ViewData["Title"] = "Mājas lapa";
}

<div class="text-center">
   <h1 class="display-4">@Model.Message</h1>
   <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>
</div>
```



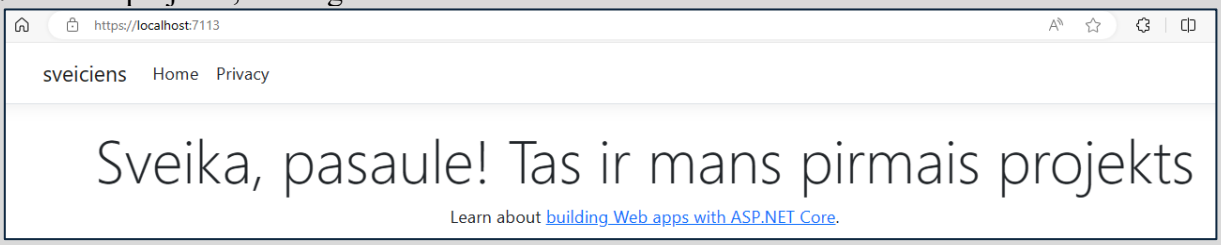
4. Izvēlas datni **index.cshtml.cs**:



5. Tur esošo kodu nomaina ar šo:

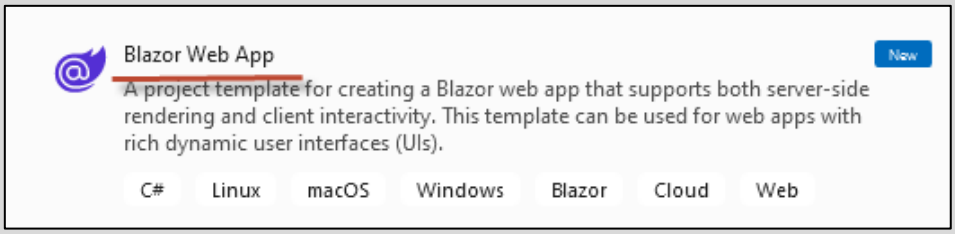
```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace sveiciens.Pages
{
    public class IndexModel : PageModel
    {
        private readonly ILogger<IndexModel> _logger;

        public string Message { get; set; }
        public IndexModel(ILogger<IndexModel> logger)
        {
            _logger = logger;
        }

        public void OnGet()
        {
            Message = "Sveika, pasaule! Tas ir mans pirmais projekts";
        }
        public void OnPost()
        {

        }

    }
}
```

6. Palaižot projektu, tiek iegūts šāds rezultāts:

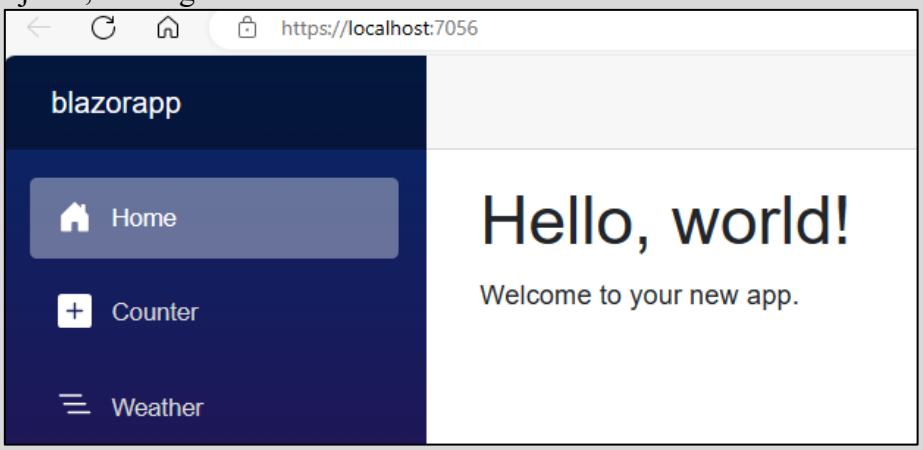


12.3. piemērs. ASP.NET *Core Blazor* projekta demonstrācijas iesākums

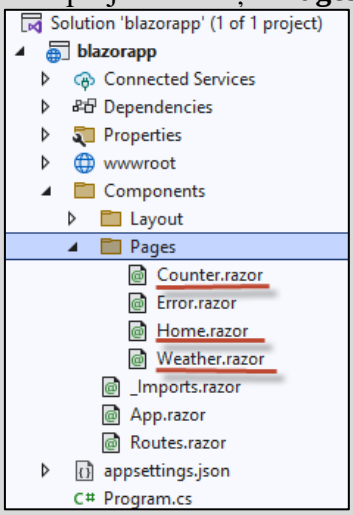
1.*Visual Studio* vidē izveido jaunu projektu **Blazor Web App** ar nosaukumu **blazorapp**.



2. Palaižot projektu, tiek iegūts šāds rezultāts:



3. Sākuma lapas iestatījumus var mainīt projekta sadaļas **Pages** attiecīgajās datnēs:



4. Datnē **Home.razor** var, piemēram, nomainīt apsveikumu "Hello, world":

```
@page "/"

<PageTitle>Home</PageTitle>

<h1>Hello, world!</h1>

Welcome to your new app.
```

5. Datnē **Counter.razor** tiek uzskaitīts lapas apmeklējumu skaits:

```
@page "/counter"
@rendermode InteractiveServer

<PageTitle>Counter</PageTitle>

<h1>Counter</h1>

<p role="status">Current count: @currentCount</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

@code {
    private int currentCount = 0;

    private void IncrementCount()
    {
        currentCount++;
    }
}
```

6. Datnē **Weather.razor** tiek simulēta asinhrona datu par laika apstākliem apstrāde:

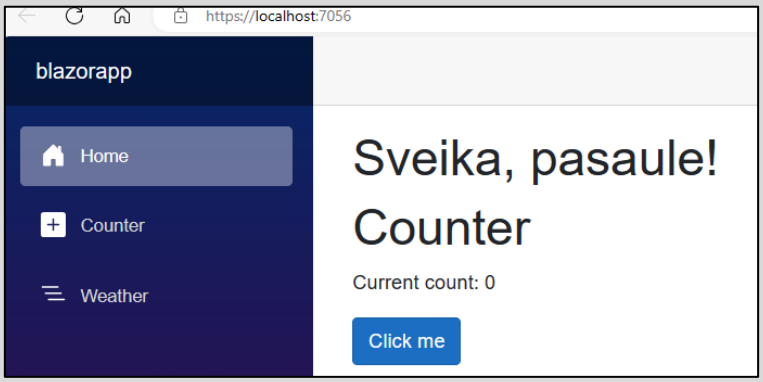
```
@page "/weather"
@attribute [StreamRendering]

<PageTitle>Weather</PageTitle>

<h1>Weather</h1>

<p>This component demonstrates showing data.</p>

@if (forecasts == null)
{
    <p><em>Loading...</em></p>
}
else
{
    <table class="table">
        <thead>
            <tr>
                <th>Date</th>
                <th>Temp. (C)</th>
                <th>Temp. (F)</th>
                <th>Summary</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var forecast in forecasts)
            {
                <tr>
                    <td>@forecast.Date.ToShortDateString()</td>
                    <td>@forecast.TemperatureC</td>
                    <td>@forecast.TemperatureF</td>
                    <td>@forecast.Summary</td>
                </tr>
            }
        </tbody>
    </table>
}

@code {
```

```
    private WeatherForecast[]? forecasts;

    protected override async Task OnInitializedAsync()
    {
        // Simulate asynchronous loading to demonstrate streaming rendering
        await Task.Delay(500);

        var startDate = DateOnly.FromDateTime(DateTime.Now);
        var summaries = new[] { "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching" };
        forecasts = Enumerable.Range(1, 5).Select(index => new WeatherForecast
        {
            Date = startDate.AddDays(index),
            TemperatureC = Random.Shared.Next(-20, 55),
            Summary = summaries[Random.Shared.Next(summaries.Length)]
        }).ToArray();
    }

    private class WeatherForecast
    {
        public DateOnly Date { get; set; }
        public int TemperatureC { get; set; }
        public string? Summary { get; set; }
        public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);
    }
}
```

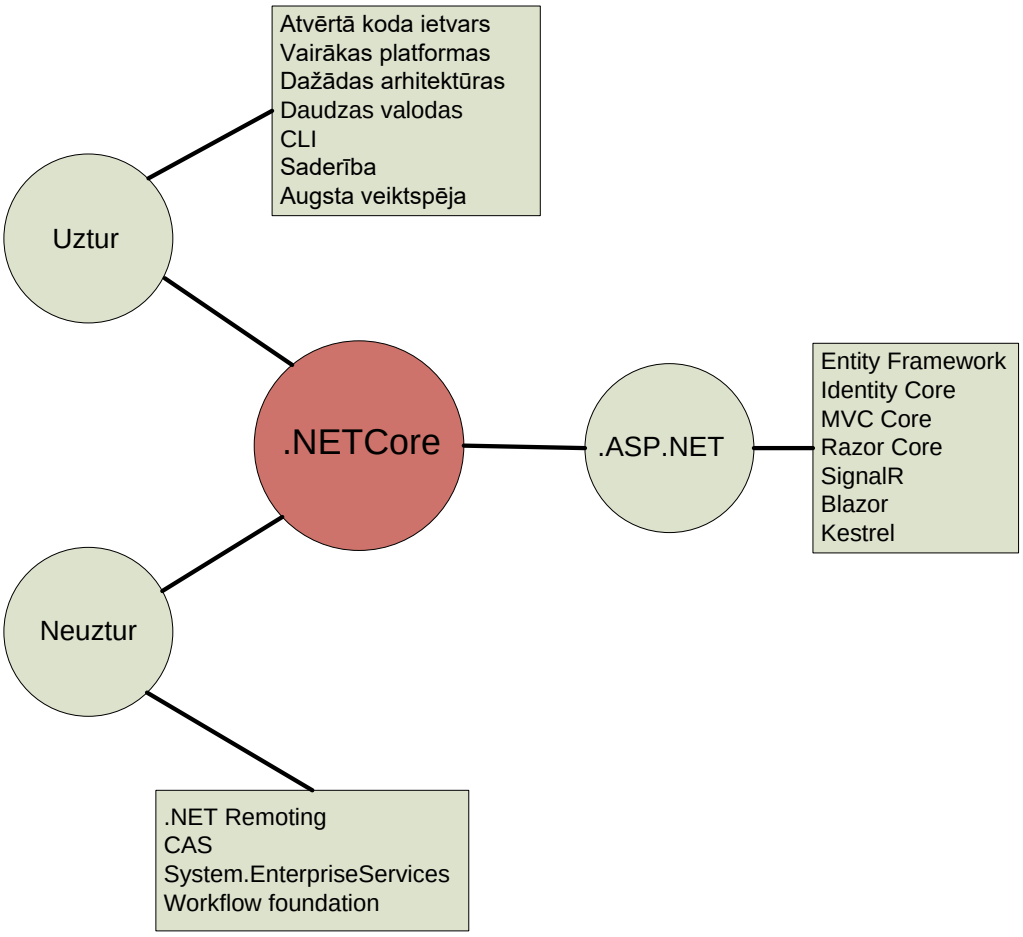
7. Eksperimenta nolūkā datnē **Home.razor** var nomainīt apsveikumu un iznest **Counter** uz titullapu:

```
@page "/"

<PageTitle>Home</PageTitle>

<h1>Sveika, pasaule!</h1>

<Counter />
```

8. Tiks iegūts šāds rezultāts:



12.2. attēlā parādīts nodaļā minēto svarīgāko jēdzienu koks.



12.2. att. **Divpadsmitajā nodaļā minēto svarīgāko jēdzienu koks**

13. Pamatjēdzienu kopsavilkums

ACL (*Access Control List*) – piekļuves vadības saraksts, t. i., datoru lietotāju saraksts, kas norāda, kuriem lietotājiem ir atļauta piekļuve kādam no datoru resursiem un kā lietotāji šo resursu var izmantot.
Atļauja (*permission*) – noteikumi, kas saistīti ar kāda datoru tīkla vai vairāklietotāju sistēmas konta izmantošanu un kas nosaka, kādi datoru tīkla vai sistēmas resursi ir pieejami konta lietotājam.
Atšifrēšana (*decryption*) – šifrēta ziņojuma apstrāde, ko veic tā saņēmējs, lai atklātu ziņojuma sākotnējo saturu.
Audits (*secure audit*) – neatkarīga datoru tīkla funkcionēšanas novērošana un analīze, lai noteiktu izmantoto drošības līdzekļu pietiekamību, to atbilstību pieņemtajiem drošības noteikumiem un datu apstrādes procedūrām, kā arī lai konstatētu drošības pārkāpumus, izstrādātu ieteikumus drošības pārbaudes līdzekļu un drošības procedūru uzlabošanai.
Autentificēšana (*authentication*) – ziņojuma, datu avota un datu saņēmēja autentiskuma apstiprināšana, kā arī lietotāja identitātes pārbaudes procedūra.
Autentifikācijas kods (*authenticode*) – vadības ziņojumu lauks, ko piekārto datu blokam, lai identificētu ziņojumu. Autentifikācijas kodā parasti ietilpst speciāla parole, ziņojuma kārtas numurs, norāde uz pārraides laiku un datumu, kā arī cita informācija, kas atvieglo datu nesankcionētas izmantošanas un izmaiņas atklāšanu.
Blazor – jauna .ASP.NET *Core* tehnoloģija, kas atbalsta gan servera puses atveidošanu, gan klienta interaktivitāti vienā programmēšanas modelī.
Buferis (*buffer*) – atmiņa īslaicīgai datu glabāšanai, ko pārsūta starp divām ierīcēm, lai izlīdzinātu to darbības ātrumus, signālu līmeņus vai nodrošinātu asinhronu pārraidi.
CA (*Certificate Authority*) – sertificēšanas institūcija ir uzticama organizācija, kas izsniedz sertifikātus, ar to apliecinot, ka attiecīgajai personai pieder dotā publiskā atslēga, tādējādi apstiprinot viņas identitāti.
Ciparparaksts (*Digital Signature*) – dati, kas pievienoti datu blokam vai arī ir iegūti, tos kriptogrāfiski pārveidojot, un kas ļauj datu saņēmējam pārliecināties par datu bloka integritāti un datu avota autentiskumu, kā arī nepieļauj to viltošanu.
Ciparsertifikāts (*Digital Certificate*) – elektronisks dokuments, kurš apstiprina publiskās atslēgas piederību kādai personai vai citai vienībai. Ciparsertifikāts satur lietotāja publisko atslēgu un vārdu, kā arī lietotāja sertificēšanas institūcijas parakstu. Tas parasti satur arī derīguma termiņu, sertificētāja vārdu, lietotāja e-pasta adresi u. c. informāciju. Ciparsertifikāts, kuru izdevis uzticams sertificētājs (CA), ļauj pārliecināties, ka sūtījumu parakstītājs lieto to publisko atslēgu, kas tam būtu jālieto.
DACL (*Discretionary ACL*) – īpašnieka piekļuves vadības saraksts. Piekļuves vadības saraksts sastāv no atsevišķiem ierakstiem – ACE ievadnēm (*Access Control Entry*).
Dalības noliegums (*repudiation*) – situācija, kurā viens no datoru tīkla lietotājiem neatzīst savu līdzdalību datu apmaiņas procesā vai kādā tā daļā.
Deklaratīvie priekšraksti (*declarative*) – nosaka .NET *Framework* videi veikt drošības pasākumu pārbaudi pirms metodes izpildes. Tas ir visdrošākais veids ierobežot piekļuvi kodam, jo izpildes vide veic drošības pārbaudi pirms koda izpildes.
Identificēšana (*identification*) – operācija, kas datu apstrādes sistēmās nosaka lietotāja vai lietojumprocesa identitāti. Pēc identificēšanas parasti seko autentificēšana.
Identitāte (*identity*) – .NET *Framework* objekts, kas iekapsulē informāciju par lietotāju.

Iekapsulēšana – datu struktūru un atsevišķu funkciju apvienošana vienā objektā, lai novērstu nevēlamus blakusefektus sistēmas darbībā. Izmantojot iekapsulēšanu, lielas objektorientētas programmas kļūst labāk lasāmas, jo visi dati un ar tiem saistītie kodi ir atrodami vienā vietā.

Imperatīvie priekšraksti (*imperative*) – tiek noteikti kodā un var tikt izmantoti detalizētākam piekļuves ierobežojumam koda sastāvdaļām, t. i., ierobežo piekļuvi atsevišķiem metodes komponentiem.

Integritāte (*integrity*) – sistēmu, programmu un datu aizsardzība pret netīšu vai ļaunprātīgu bojāšanu vai pārveidošanu.

Izlikšanās (*spoofing*) – metodes, ko izmanto nesankcionētai piekļuvei datoriem. Pārkāpējs sūta ziņojumu datoram, izmantojot IP adresi, kura norāda, ka ziņojums nāk no pārbaudītas pieslēgvietas. Lai to izdarītu, kramplauzim vispirms jāatrod šāda IP adrese, pēc tam jāmaina pakešu galvenes tā, lai radītu pārliecību, ka paketes tiek sūtītas no šīs pieslēgvietas.

Izolēta glabātava (*isolated storage*) – slēgta datņu sistēma .NET *Framework* pārvaldībā, ko ierobežo lietotājs, asambleja vai domēns.

Jaucējfunkcija (*hash function*) – algoritms, kas pārveido dažāda garuma tekstu fiksēta garuma izvadē. Jaucējfunkcijas izmanto, lai veidotu, piem., ciparparakstus vai lai pārvērstu lietotājam nozīmīgu identifikatoru vai atslēgu par norādi, kādā noteiktas struktūras (tabulas) vietā šo informāciju var atrast.

Koda piekļuves aizsardzība (*Code Access Security – CAS*) – drošības sistēma, kas ļauj administratoram un izstrādātājam veikt lietotnes pilnvarošanu. Tā balstās ne uz lietotāju identificēšanu, bet uz izpildāmā koda identificēšanu (kodam atļautās operācijas un pieejamie resursi ir ierobežoti saskaņā ar drošības apliecinājumiem).

Konts (*user account*) – drošības mehānisms, ko izmanto, lai kontrolētu piekļuvi datoru sistēmai vai tīklam. Lietotāja kontu nosaka un pārrauga sistēmas administrators. Lietotāja kontā ietilpst informācija par lietotāja paroli, tā tiesībām un, iespējams, cita informācija, kas ļauj identificēt lietotāju.

Lietotāja autentificēšana (*authentication of user*) – procedūra, kas vairāklietotāju sistēmās un datoru tīklos ļauj pārbaudīt, vai lietotājs atbilst uzrādītajam identifikatoram. Par identifikatoriem var kalpot, piem., dažādas paroles, šifrēšanas atslēgas u. c. Lietotāja autentificēšanu izmanto, lai aizsargātu datu apstrādes sistēmas no nesankcionētas piekļuves, reģistrētu lietotāju un izvēlētos tam atbilstošu apkalpošanas režīmu.

MVC (*Model-View-Controller*) – programmatūras projektēšanas modelis, ko parasti izmanto lietotāja saskarņu izstrādei.

Pakalpojumu atteikums (*denial of service*) – sankcionētas piekļuves nenodrošināšana tīkla resursiem vai operāciju izpildes laika nepieļauta novilcināšana datoru tīklos.

Pārpilde (*overflow*) – noteiktu datu vai programmu uzglabāšanai datorā paredzētā atmiņas apjoma pārsniegšana.

Pilnvaras (*principal*) – .NET *Framework* objekts, kas iekapsulē informāciju par lietotāja lomām. Pilnvaras veido identitātes un lomu kompozīcija.

Pilnvarošana *(authorization)* – pilnvaru piešķiršana kādai personai vai personu grupai noteiktu darbību izpildei un resursu izmantošanai.

Privātā atslēga (*Private Key*) – slepena atslēga, ko publiskās atslēgšifrēšanas procesā lietotājs neatklāj nevienam, bet izmanto, lai apzīmētu nosūtītos ziņojumus, kā arī atšifrētu saņemtos ziņojumus, kas šifrēti, lietojot publisko atslēgu.

Publiskā atslēga (*Public Key*) – atslēga, ko publiskās atslēgšifrēšanas procesā lietotājs izplata saviem potenciālajiem korespondentiem un ko šie korespondenti izmanto, lai šifrētu lietotājam adresētos ziņojumus un atšifrētu lietotāja signatūru, kas šifrēta ar lietotāja privāto atslēgu.

Razor – jauna .ASP.NET *Core* tehnoloģija, kas padara uz lapām orientētu scenāriju kodēšanu vieglāku un produktīvāku.

SACL (*System ACL*) – sistēmas piekļuves vadības saraksts. Apraksta darbības ar objektiem, kas pakļauti auditēšanai.

Skripts *(script)* – instrukciju virkne, kas nosaka, kā programmai jāveic kāda specifiska procedūra, piem., ieiešana elektroniskā pasta sistēmā. Dažādās programmās skripta iespējas ir jau iebūvētas, dažus skriptus veido automātiski, pierakstot, kādus taustiņus un komandu izvēlnes lietotājs izmanto procedūru laikā. Globālā tīmekļa kontekstā skripts ir programma, kas atrodas kādā tīmekļa serverī un apstrādā no pārlūkprogrammas saņemtos pieprasījumus.

Starpvietņu skriptošana XSS *(cross-site scripingt)* – XSS (*Cross Site Scripting*) uzbrukums ir uzbrukuma veids tīmekļa aplikācijām. Uzbrucējs izmanto XSS, lai iemānītu savu izveidoto skriptu gala lietotājam. Tā kā lietotāja pārlūkprogramma uzskata, ka skripts ir saņemts no uzticama informācijas avota un tai nav nekādas iespējas uzzināt to, ka skriptam nedrīkst uzticēties, programma palaidīs skriptu, tādējādi nodrošinot tam piekļuvi jebkurām lietotāja sīkdatnēm (*cookies*), sesiju datiem un citai informācijai, kuru lietotāja pārlūkprogrammā izmanto tīmekļa lapā. Ar šādu skriptu palīdzību iespējams pārrakstīt HTML lapas saturu.

Stingrs vārds (*Strong Name*) – drošs asamblejas identifikators, kas samazina risku ļaunprātīgas asamblejas izmaiņas vai nomaiņas gadījumā.

Simetriskā atslēgšifrēšana (*symmetric-key cryptography*) – šifrēšanas sistēma, kurā atšķirībā no publiskās atslēgšifrēšanas ziņojuma nosūtītājs un ziņojuma saņēmējs ziņojuma šifrēšanai un atšifrēšanai izmanto vienu un to pašu šifrēšanas atslēgu. Simetriskā atslēgšifrēšana ir vienkāršāka un ātrāka nekā publiskā atslēgšifrēšana, bet, lai izmantotu šo šifrēšanas sistēmu, jāatrod drošs veids, kā abām informācijas apmaiņā iesaistītajām pusēm apmainīties ar šifrēšanas atslēgām. Publiskajā atslēgšifrēšanā privātā šifrēšanas atslēga nekad netiek pārsūtīta.

Smilšu kaste (*Sandbox*) – mehānisms, kas ļauj droši izpildīt aizdomīgas programmas. Dažkārt saka, ka tas ir virtuāls konteiners, kurā var tikt droši izpildītas neuzticamas lietotnes.

SOAP (*Simple Object Access Protocol*) – vienkāršais objektu piekļuves protokols. Tas ir protokols, ar kura palīdzību *Internet* tīklā un citās izkliedētās skaitļošanas vidēs var sazināties un kopīgi darboties dažādu platformu un operētājsistēmu programmatūra, sūtot cita citai XML formātā strukturētus datus. Ir specificēts, kā šāda apmaiņa var notikt ar HTTP palīdzību.

SQL injekcija (*injection*) – datorsistēmu apdraudējuma veids, kurā uzbrucējs ar klienta tiesībām mēģina piekļūt datubāzei, kas ir dotās tīmekļa lapas vai aplikācijas pamatā. Mūsdienās SQL injekcija ir vispopulārākais aplikācijas līmeņa uzbrukuma veids. SQL injekcijas pamatā ir nepareizi uzrakstītas programmas vai tīmekļa aplikācijas, kas lietotājam ļauj veikt datubāzes vaicājumus, kuras nav paredzējis programmatūras vai mājaslapas izstrādātājs. Šai situācijai pamatā parasti ir nepietiekama lietotāju ievadītās informācijas pārbaude, kuras rezultātā netiek filtrēti SQL īpašie simboli.

Testēšana (*testing*) – programmatūras un aparatūras darbības pārbaude, izmantojot testdatus. Testēšanas mērķis var būt defekta atklāšana, tā atrašanās vietas lokalizēšana vai arī testējamā objekta dinamisko parametru (piemēram, ātrdarbības) noskaidrošana.

Tests (*test*) – speciāli izveidota datu kopa, ko izmanto, lai pārbaudītu datora vai sistēmas darbības atbilstību paredzētajām prasībām. Kā testu var izmantot datus, kas iegūti sistēmas iepriekšējās darbības rezultātā vai arī mākslīgi radīti šīs pārbaudes veikšanai.

Testpiemērs (*test case*) – konkrētam mērķim izstrādāta testa ievaddatu, izpildes noteikumu un sagaidāmo rezultātu kopa, lai dotu iespēju pārbaudīt testējamā objekta atbilstību iepriekš definētajām prasībām.

Tīmekļa pakalpojumi jeb **servisi** (*Web services*) – pakalpojumi, kas tiek sniegti tīmeklī. Tie ir starptīkla pakalpojumu speciālgadījumi, lai arī bieži vien tiek ar tiem jaukti. Tā, piemēram, e-pasts ir starptīkla, bet nav tīmekļa pakalpojums, savukārt tīmekļa saskarne e-pasta aplūkošanai ir gan tīmekļa, gan arī starptīkla pakalpojums. Vairumā gadījumu tīmekļa pakalpojumi ir pieejami, izmantojot tīmekļa pārlūku, tomēr tas nevar būt par formālu kritēriju to identificēšanai; piemēram, FTP vietnes iespējams aplūkot arī vairumā tīmekļa pārlūku, tomēr FTP nav tīmekļa pakalpojums.

Šifrēšana (*Encryption*) – datu un ziņojumu apstrādes process, ko veic datu sagatavotājs vai ziņojuma nosūtītājs, lai datus vai ziņojuma saturu nodrošinātu pret nesankcionētu izmantošanu. Lai šādus datus vai ziņojuma saturu varētu izmantot, jāveic tā atšifrēšana.

URL (*Uniform Resource Locator*) – vienotais resursu vietrādis. Tā ir adrese, kas pārlūkprogrammā norāda, kur var atrast kādu konkrētu *Internet* tīkla resursu. Vienveida resursu vietrādim varētu būt, piem., šāda struktūra: *http://www.edzi.lza.lv/history.htm*. Vietrāža pirmā daļa (līdz dubultajai slīpsvītrai) norāda piekļuves metodi (šajā piemērā – hiperteksta transporta protokolu). Teksts starp dubulto slīpsvītru un vienkāršo slīpsvītru norāda serveri, bet teksts aiz vienkāršās slīpsvītras – direktoriju vai datni.

Lomās bāzēta aizsardzība (*Role-Based Security - RBS*) – lietotāju autentificēšanas un pilnvarošanas process, kas nosaka to, kādus resursus lietotājs ir pilnvarots izmantot. Loma (*role*) – definē darba funkcijas, kas tiek noteiktas pilnvarošanas procesā.

XML (*eXtensible Markup Language*) – programmēšanas valoda, kas speciāli izstrādāta darbam ar tīmekļa dokumentiem. Valoda XML nodrošina globālā tīmekļa izstrādātajiem iespēju radīt savas personiskās birkas (kodus), lai nodrošinātu tādu funkciju izpildi, ko nevar nodrošināt ar valodas HTML starpniecību. Atšķirībā no valodas HTML, kuras saites norāda tikai uz vienu dokumentu, valoda XML nodrošina saites, kurās ir atsauces uz vairākiem dokumentiem.

Izmantotās literatūras un avotu saraksts

1. Microsoft Official Course 2840A: Implementing Security for Applications, Microsoft Corporation.
2. Northrup T. MCAD/MCSD Self-Paced Training Kit: Implementing Security for Applications with Microsoft® Visual Basic® .NET and Microsoft Visual C#® .NET, 2004. https://archive.org/details/mcadmcsdselfpace00nort/page/n1/mode/2up
3. Buffer overflow. https://owasp.org/www-community/vulnerabilities/Buffer_Overflow
4. SQL injection. https://owasp.org/www-community/attacks/SQL_Injection
5. Cross site scripting. https://owasp.org/www-community/attacks/xss/
6. Google Uses 20 Canonicalization Signals. https://ahrefs.com/blog/canonicalization/
7. A Look Inside the Security Development Lifecycle at Microsoft. https://learn.microsoft.com/en-us/archive/msdn-magazine/2005/november/a-look-inside-the-security-development-lifecycle-at-microsoft
8. STRIDE Threat Modeling: What You Need to Know. https://www.softwaresecured.com/post/stride-threat-modeling
9. DREAD Threat Modeling. https://threat-modeling.com/dread-threat-modeling/
10. Janca T. Alice and Bob Learn Application Security. Wiley, 2020.
11. Kohnfelder L. Designing Secure Software: A Guide for Developers. No Starch Press, 2021.
12. Abdul-Rahman A. Application Security Program Guide: Building a Comprehensive Application and Product Security Program. 2023.
13. Borg O. Mastering Security+: A Journey Through 500 Questions SY0-701/SY0-601. 2024.
14. Shostack A. Threat Modeling: Designing for Security. Wiley.
15. Howard, Michael and LeBlanc, David. *Writing Secure Code, 2d edition*. Redmond, WA: Microsoft Press, 2002.
16. What is discretionary access control (DAC)? https://nordlayer.com/learn/access-control/discretionary-access-control/
17. Access control lists. https://learn.microsoft.com/en-us/windows/win32/secauthz/access-control-lists
18. Katz J., Lindell Y. Introduction to modern cryptography. Second edition. 2015. CRC Press. https://eclass.uniwa.gr/modules/document/file.php/CSCYB105/Reading%20Material/%5BJonathan_Katz%2C_Yehuda_Lindell%5D_Introduction_to_Mo%282nd%29.pdf
19. Friedl J. Mastering Regular Expressions. 3rd Edition. 2006, O'Reilly. https://terrorgum.com/tfox/books/masteringregularexpressions.pdf
20. Posadas M. Mastering C# and .NET Framework. Packt Publishing, 2016.
21. Lock A. ASP.NET Core in Action, Third Edition. Manning, 2023.
22. Bock J. .*NET Security*. Berkeley, CA: Apress LP, 2010. http://dx.doi.org/10.1007/978-1-4302-0846-4
23. Role based security. http://www.dranieh.com/NETSecurity/RoleBasedSecurity.htm
24. .NET Code Access Security (CAS). https://www.c-sharpcorner.com/UploadFile/84c85b/net-code-access-security-cas/
25. Code Access Security. https://www.albahari.com/nutshell/CodeAccessSecurity.pdf
26. .NET Framework technologies unavailable on .NET. https://learn.microsoft.com/en-us/dotnet/core/porting/net-framework-tech-unavailable
27. ASP.NET security overview. https://learn.microsoft.com/en-us/troubleshoot/developer/webapps/aspnet/development/security-overview
28. .NET Remoting. https://en.wikipedia.org/wiki/.NET_Remoting
29. .NET Remoting Architecture. https://learn.microsoft.com/en-us/previous-versions/dotnet/netframework-1.1/2e7z38xb(v=vs.71)?redirectedfrom=MSDN
30. SOAP Web Services Tutorial: What is SOAP Protocol? https://www.guru99.com/soap-simple-object-access-protocol.html
31. Nunit. https://nunit.org/
32. Microsoft Baseline Security Analyzer. https://learn.microsoft.com/en-us/previous-versions/cc184923(v=msdn.10)?redirectedfrom=MSDN
33. The Open source security testing methodology manual. https://www.isecom.org/OSSTMM.3.pdf
34. XCopy. https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/xcopy
35. Windows Installer. https://learn.microsoft.com/en-us/windows/win32/msi/windows-installer-portal
36. Cabinet Files. https://learn.microsoft.com/en-us/windows/win32/msi/cabinet-files
37. ClickOnce security and deployment. https://learn.microsoft.com/en-us/visualstudio/deployment/clickonce-security-and-deployment?view=vs-2022

38. Sandbox (computer security). https://en.wikipedia.org/wiki/Sandbox_(computer_security)
39. How to: Sign an assembly with a strong name. https://learn.microsoft.com/en-us/dotnet/standard/assembly/sign-strong-name
40. Strong Name Signing. https://github.com/dotnet/runtime/blob/main/docs/project/strong-name-signing.md
41. Strong name. https://learn.microsoft.com/en-us/archive/msdn-magazine/2006/july/clr-inside-out-using-strong-name-signatures
42. Overview of ASP.NET Core. https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0
43. .NET Core Overview. https://www.tutorialsteacher.com/core/dotnet-core
44. MIT License. https://github.com/dotnet/runtime/blob/main/LICENSE.TXT
45. NET foundation. https://dotnetfoundation.org/
46. .NET Standard. https://learn.microsoft.com/en-us/dotnet/standard/net-standard?tabs=net-standard-1-0
47. ASP.NET documentation. https://learn.microsoft.com/lv-lv/aspnet/core/?view=aspnetcore-8.0
48. Entity Framework Core. https://learn.microsoft.com/en-us/ef/core/
49. Introduction to Identity on ASP.NET Core. https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-8.0&tabs=visual-studio
50. Overview of ASP.NET Core MVC. https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-8.0
51. Introduction to Razor Pages in ASP.NET Core. https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-8.0&tabs=visual-studio
52. Introduction to SignalR. https://learn.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr
53. ASP.NET Core Blazor. https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-8.0
54. Kestrel web server in ASP.NET Core. https://learn.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel?view=aspnetcore-8.0